Università degli Studi di Padova

DIPARTIMENTO DI MATEMATICA "TULLIO LEVI-CIVITA"

Corso di Laurea Magistrale in Data Science

Optimization for Data Science

Projection-Free Optimization: Frank-Wolfe and Pairwise Frank-Wolfe Algorithms

Sobhan Hosseini
Matricola: 2141890

Marco Munich
Matricola: 2163522

Anno Accademico 2024/2025

Contents

1 Abstract

This project aims to explore projection-free optimization methods with particular attention to the Frank-Wolfe (FW) and Pairwise Frank-Wolfe (PFW) algorithms and their application to the matrix completion problem. By leveraging the nuclear norm as a convex surrogate for rank constraints, we formulate a tractable convex optimization problem and implement FW-based algorithms to solve it efficiently. The Frank-Wolfe algorithm avoids costly projections by using a Linear Minimization Oracle (LMO), making it particularly suitable for large-scale problems where low-rank solutions are required. The Pairwise Frank-Wolfe variant gives us better convergence by dynamically redistributing weights within the active set. Through various theoretical insights and empirical analysis, we demonstrate the advantages and limitations of both algorithms, highlighting their efficiency in promoting low-rank structure while maintaining computational efficiency.

2 Introduction

Many modern machine learning and signal processing problems can be formulated as constrained convex optimization problems. In such cases, one seeks to minimize a smooth convex function over a domain defined by structural constraints such as sparsity, low-rankness, or probability distributions.

Standard first-order methods such as projected gradient descent require computing a projection onto the constraint set at each iteration. While effective in simple geometries (e.g., Euclidean balls or boxes), projections become computationally expensive or even intractable for complex sets like the nuclear norm ball or the probability simplex in high dimensions.

The Frank-Wolfe algorithm, also known as the conditional gradient method, offers a compelling alternative. Instead of projecting onto the constraint set, it uses a *Linear Minimization Oracle* (LMO) to compute a feasible descent direction. This projection-free nature makes it attractive for large-scale problems with expensive constraints.

In this project, we investigate two related projection-free methods:

- The classical **Frank-Wolfe (FW)** algorithm, which enjoys sublinear convergence guarantees;
- The **Pairwise Frank-Wolfe (PFW)** variant, which achieves linear convergence under stronger assumptions by shifting weight between active atoms.

We apply both algorithms to a convex relaxation of the matrix completion problem, where the optimization domain is defined by a nuclear norm constraint. This setting is well-suited for Frank-Wolfe methods because the LMO corresponds to computing a leading singular vector, which is more efficient than full projections.

The report is structured as follows: Section 2 describes the matrix completion problem. Sections 3 and 4 detail the FW and PFW algorithms. Section 5 discusses implementation strategies. Section 6 presents numerical results, and Section 7 provides analysis and conclusions.

3 Problem Setting

In this section, we focus on the matrix completion and the convex formulation with the nuclear norm

3.1 Matrix Completion Problem

The matrix completion problem is a common challenge in data science and related fields such as engineering and machine learning. The main objective is to recover a full matrix from a partially observed one — that is, to start with a matrix where many entries are missing or unobserved, and to accurately or infer those missing values. A key assumption is that the underlying complete matrix is low-rank, meaning that the data can be represented using a small number of latent factors.

$$\begin{aligned} \min_{X \in \mathbb{R}^{n_1 \times n_2}} \quad & f(X) := \sum_{(i,j)\in J} (X_{ij} - U_{ij})^2 \\ \text{s.t.} \quad & \operatorname{rank}(X) \le \delta. \end{aligned} \tag{1}$$

The parameter δ (with $\delta > 0$) represents our belief in the desired rank of the final completed matrix.

The constraint $\operatorname{rank}(X) \le \delta$ makes the problem nonconvex, because the rank function is discrete and nonlinear. Optimizing rank-constrained problems is NP-hard, so solving them exactly is computationally impractical for large matrices. In the next paragraph, we are going to introduce a model to solve this problem.

3.2 Convex Formulation with Nuclear Norm

As we have just seen, (1) introduces a problem that is not feasible to compute for large-scale matrices. To make the problem tractable, a convex relaxation is applied by replacing the rank constraint (or penalty) with the nuclear norm. The nuclear norm $\|X\|_*$ is defined as the sum of the singular values of X. It serves as the convex envelope of the rank function over matrices with spectral norm less than or equal to one. Minimizing the nuclear norm effectively promotes low-rank solutions in a convex framework, making the problem solvable using convex optimization algorithms.

The main idea is to replace the low-rank constraint $((X) \le \delta)$ with a nuclear norm ball constraint, where the nuclear norm $\|X\|_*$ of a matrix X is defined as the sum of its singular values. This leads to the following convex optimization formulation:

$$\begin{aligned} \min_{X \in \mathbb{R}^{n_1 \times n_2}} \quad & \sum_{(i,j)\in J} (X_{ij} - U_{ij})^2 \\ \text{s.t.} \quad & \|X\|_* \le \delta. \end{aligned} \tag{2}$$

3.3 Notation and Assumptions

In this context, $U \in \mathbb{R}^{n_1 \times n_2}$ represents the original partially observed matrix, while $X \in \mathbb{R}^{n_1 \times n_2}$ denotes the reconstructed matrix we are trying to recover. The index set $J \subseteq \{1, \dots, n_1\} \times \{1, \dots, n_2\}$ indicates which entries are observed. We assume that the underlying matrix is low-ranked and that the observed entries provide enough information for a reliable recovery. The nuclear norm $\|X\|_*$ is defined as the sum of the singular values of X and act as a convex surrogate for the rank function. The parameter $\delta > 0$ introduces an upper bound on the nuclear norm, encouraging a low-rank solution.

4 Frank-Wolfe Algorithm

The Frank-Wolfe algorithm, also known as the Conditional Gradient Method, is a classic iterative constrained method used to solve convex optimization problems. It is especially useful when the feasible region is convex, but projecting onto it is computationally expensive, while, on the other hand, solving a linear minimization over it remains tractable. Due to its simplicity and projection-free nature, the Frank-Wolfe algorithm is widely used in large-scale machine learning, signal processing, and sparse optimization applications.

4.1 General Idea and Intuition

The main idea behind the Frank-Wolfe algorithm is to iteratively approximate the solution by moving towards a direction that minimizes a linear approximation of the objective function over the feasible set. At each iteration, instead of performing a potentially costly projection, the method solves a linear subproblem to find a feasible search direction and then updates the current solution using a convex combination. This approach promotes sparse solutions when the admissible set is a convex hull or a nuclear norm ball, making it a good choice for matrix completion problems with nuclear norm constraints. The algorithm works particularly well when the objective function is convex and differentiable, since this ensures that the linear approximations reliably guide the optimization process.

4.2 Algorithm Description and Pseudocode

The Frank-Wolfe algorithm starts from an initial feasible point and iteratively improves the objective value by solving a linear minimization problem over the constraint set. This provides a descent direction, along which the algorithm updates the current solution by moving along the segment connecting the two points. The step size is determined either through exact line search or predefined heuristics. The process continues until a convergence criterion is met—typically when the duality gap is sufficiently small or a maximum number of iterations is reached.

Algorithm 1 Frank-Wolfe Algorithm

```
Initialize $x_0 \in \mathcal{D}$ feasible domain $k = 0, 1, 2, \dots$ Compute gradient $\nabla f(x_k)$ Solve linear
problem: $s_k = \arg\min_{s \in \mathcal{D}} \langle s, \nabla f(x_k) \rangle$ Choose step size $\gamma_k \in [0, 1]$ Update $x_{k+1} = x_k +$
$\gamma_k(s_k - x_k)$ Check convergence and stop if criteria met
```

4.3 Linear Minimization Oracle (LMO)

One of the main components related to the Frank-Wolfe algorithm is the Linear Minimization Oracle (LMO), which allows to solve the linear subproblem at each iteration. In particular, given the current gradient $\nabla f(x_k)$, the LMO returns a feasible point $s_k \in \mathcal{C}$ that minimizes the linear approximation of the objective function, i.e.,

$$s_k = \arg\min_{z\in\mathcal{C}} \langle S, \nabla f(x_k)\rangle.$$

The efficiency of the Frank-Wolfe algorithm heavily depends on the ability to quickly solve this linear problem. For many constraint sets, including nuclear norm balls or simplices, the LMO can be computed efficiently and often admits closed-form solutions. In the context of matrix completion with nuclear norm constraints, the LMO typically reduces to computing a rank-one singular value decomposition component corresponding to the leading singular vectors of the gradient matrix.

Algorithm 2 Frank-Wolfe Algorithm with Linear Minimization Oracle

```
Choose an initial point $x_0 \in \mathcal{C}$ feasible domain $k = 0, 1, 2, \dots$ Compute gradient $\nabla f(x_k)$
$x_k$ satisfies the convergence criterion **STOP** Compute $s_k = \arg\min_{z\in\mathcal{C}} \langle z, \nabla f(x_k)\rangle$ Linear
Minimization Oracle Set $d_k^{FW} = s_k - x_k$ Choose step size $\gamma_k \in (0, 1]$ Update $x_{k+1} =$
$x_k + \gamma_k d_k^{FW}$
```

4.4 Convergence Properties

A major advantage of the Frank-Wolfe algorithm is its solid theoretical convergence guarantees for smooth convex optimization problems. Under standard assumptions — such as convexity and Lipschitz continuity of the gradient — the Frank-Wolfe algorithm converges sublinearly with a rate of $\mathcal{O}(1/k)$, where k is the iteration count. This means that the primal gap decreases inversely with the number of iterations.

Moreover, the algorithm provides a natural measure of optimality through the *duality gap*, which can be computed directly at each iteration and used as a stopping criterion. Although the convergence rate is slower than that of projected gradient methods, the Frank-Wolfe algorithm avoids potentially expensive projection steps and often yields sparse or low-rank solutions, which is particularly beneficial for large-scale matrix completion problems.

4.5 Duality Gap and Stopping Criteria

An important feature of the Frank-Wolfe algorithm is that it naturally provides a measure of suboptimality known as the *duality gap*. At each iteration, the duality gap quantifies how far the current solution is from the optimal value by comparing the objective function with its linear approximation. Formally, for the current iterate x_k and the solution s_k from the Linear Minimization Oracle, the duality gap is defined as:

$$g(x_k) = \langle x_k - s_k, \nabla f(x_k)\rangle.$$

The duality gap is always non-negative and equal to zero if and only if the current solution is optimal. In practice, this quantity serves as a stopping criterion: the algorithm stops when the duality gap falls below a predefined tolerance level $\epsilon > 0$. This provides a reliable and computationally inexpensive way to monitor convergence without requiring the exact optimum.

5 Pairwise Frank-Wolfe Algorithm

While the classical Frank-Wolfe algorithm avoids expensive projections and ensures convergence for smooth convex functions, its sublinear convergence rate can be a significant limitation in practice. To address this, several variants have been proposed to accelerate convergence while maintaining projection-free updates. One such method is the **Pairwise Frank-Wolfe (PFW)** algorithm.

The key idea in PFW is to allow not only movement toward a newly selected atom from the LMO, but also to reduce weight from an already active atom. This enables the algorithm to move more directly within the face of the constraint polytope and can lead to **linear convergence rates** under suitable conditions, such as strong convexity and polytope constraints.

5.1 Motivation and Key Differences

In classical FW, each step adds a new atom to the active set by solving a linear minimization problem, and then moves toward that atom. However, FW is limited by zig-zagging near the boundary of the feasible region, which slows down convergence.

PFW introduces a second oracle: a **Linear Maximization Oracle** (LMAXO), which selects the atom in the active set most aligned in the opposite direction of the gradient. The update is then a pairwise exchange between the new atom s_t and the worst atom v_t, effectively shifting weight:

$$x_{t+1} = x_t + \gamma(s_t - v_t),$$

where $s_t = \arg\min_{s\in\mathcal{D}} \nabla f(x_t)^T s$ and $v_t = \arg\max_{v\in\mathcal{S}_t} \nabla f(x_t)^T v$, with $\mathcal{S}_t$ being the current active set.

5.2 Algorithm Description

At each iteration, the PFW algorithm performs the following steps:

Algorithm 3 Pairwise Frank-Wolfe (PFW) Algorithm Step

```
**Input:** Initial point $x_t$ Compute the gradient $\nabla f(x_t)$ Query the LMO to find the atom $s_t$ Query the LMAXO to find the worst atom $v_t \in S_t$ Compute the direction $d_t = s_t - v_t$ Compute step size $\gamma$ (line search or fixed) Update the iterate: $x_{t+1} = x_t + \gamma d_t$ Update the weights: decrease mass from $v_t$, increase mass for $s_t$
```

This direction d_t lies within the feasible set since both s_t and v_t are in $\mathcal{D}$, and the new iterate remains a convex combination of atoms.

5.3 Linear Convergence under Polytope Constraints

Unlike standard FW, PFW can achieve **linear convergence** when:

- The objective function f is **strongly convex** and has Lipschitz gradient;
- The feasible set $\mathcal{D}$ is a **polytope**;
- Exact LMO and LMAXO are used;

- Drop steps (i.e., full weight transfer from v_t to s_t) are allowed.

These conditions ensure that the active set efficiently shrinks toward the optimal face of the polytope.

5.4 Comparison with Classical and Away-Step Frank-Wolfe

PFW differs from:

- **Classical FW**: only moves toward s_t, never reduces mass;
- **Away-Step FW**: chooses between a FW step and an "away" step (in opposite direction). PFW performs both simultaneously (a "swap").

In practice, PFW shows faster convergence, especially when the solution lies in a low-dimensional face of the polytope.

6 Implementation Details

Our matrix-completion pipeline is implemented in Python and organized into three modules: `utils.py`, `solvers.py`, and `main.py`. Below we summarize the key components without diving into every line of code, emphasizing how they map to the algorithmic building blocks.

6.1 Dataset and Matrix Completion Task

We experiment on three widely used benchmarks: MovieLens-100k (943×1682, 100 K ratings), MovieLens-1M (6040×3706, 1 M ratings), and Jester2 (4000×100, 2 M ratings). Each dataset is loaded via `utils.load_dataset`, which:

- Reads the raw ratings into a dense array M_{true}.
- Splits observed entries 80/20 into train/test masks ($\text{mask}_{\text{train}}$, $\text{mask}_{\text{test}}$), fixed seed=42.
- Builds a sparse CSR matrix M_{obs} of training entries.
- Returns $(M_{\text{obs}}, \text{mask}_{\text{train}}, \text{mask}_{\text{test}}, M_{\text{true}})$.

We center the data by subtracting the global mean of $M_{\text{true}}[\text{mask}_{\text{train}}]$ before optimization, and add it back when computing RMSE.

6.2 Loss Function and Constraints

We solve the convex program

$$\min_{X \in^{m\times n}} \ \tfrac{1}{2}\|_{\text{train}}(X - M_{\text{obs}})\|_F^2 \quad \text{s.t.} \quad \|X\|_* \le \tau,$$

where $_{\text{train}}$ projects onto the training entries. The nuclear-norm radius is set to

$$\tau = \tau_{\text{scale}} \times \ \texttt{approximate\_nuclear\_norm}(M_{\text{obs}}, k = \tau_{\text{approx\_k}}),$$

using a truncated SVD (`scipy.sparse.linalg.svds`) or fallback to the full nuclear norm.

6.3 Linear Minimization Oracle (LMO)

The LMO solves $\min_{\|S\|_* \le \tau} \langle S, \nabla f(X) \rangle$ via:

1. Build the sparse gradient ∇f on train entries.
2. Define `mat_vec`$(x) = \nabla f\, x$ and `vec_mat`$(y) = \nabla f^\top y$.
3. Run `power_method` (warm-started via `prev_vec`) to extract top singular pair (u, v).
4. Return `Atom(u,v,tau)`, representing $S = -\tau\, u\, v^\top$, flipping sign if $\langle \nabla f, S \rangle > 0$.

6.4 Frank–Wolfe and Pairwise Updates

In `solvers.py` we implement two solver classes:

- **FrankWolfe:** Compute FW direction $d = S - X$, choose step-size γ (analytic line-search, vanilla $2/(t + 2)$, or fixed), update weights via $w \leftarrow (1 - \gamma)w$ and $w_{\text{new atom}} += \gamma$.
- **PairwiseFrankWolfe:** In addition to the FW atom, select an "away" atom V by maximizing $\langle \nabla f, V \rangle$; set $d = S - V$, transfer mass γ from V to S, and prune atoms with $w < 10^{-8}$.

Both maintain: {`atoms`}, {`weights`}, diagnostics (`history`, `times`), and snapshots of the current sparse matrix on train entries.

6.5 Code Structure Overview

`utils.py`
- `load_dataset`, `train_test_split_matrix`
- `approximate_nuclear_norm` using SVDS
- `evaluate` for RMSE

`solvers.py`
- `power_method` & `nuclear_norm_lmo` (LMO)
- `Atom`, `MatrixCompletionObjective`
- `FrankWolfe`, `PairwiseFrankWolfe` (step rules, gap, active-set)

`main.py`
- `ExperimentRunner` orchestrates runs over {FW, PFW}×{analytic, vanilla}
- Times `solver.run()`, reconstructs full/train/test predictions from atoms+weights
- Computes RMSE, collects diagnostics in `summary_rows`
- Plotting helpers: convergence panels, time-breakdown, heatmaps, cross-dataset tables

This modular layout cleanly separates data handling, optimization logic, and experiment orchestration, making it straightforward to extend with new Frank–Wolfe variants or datasets.

7 Experimental Results

In this section we present our empirical evaluation of the Frank–Wolfe (FW) and Pairwise Frank–Wolfe (PFW) algorithms on three real-world matrix-completion datasets: MovieLens-100k, MovieLens-1M, and Jester-2. We begin by defining the metrics used to assess convergence and predictive performance.

7.1 Evaluation Metrics

We use the following metrics throughout.

Root-Mean-Squared Error (RMSE). Given a ground-truth matrix M_{true} and a predicted matrix $\widehat{M}$, we report

$$\text{RMSE} = \sqrt{\frac{1}{|\mathcal{I}|} \sum_{(i,j)\in\mathcal{I}} \left(M_{\text{true}}[i,j] - \widehat{M}[i,j]\right)^2},$$

where $\mathcal{I}$ is either the training set of observed entries (for RMSE_{tr}) or the held-out test set (for RMSE_{te}).

Normalized RMSE (NRMSE). To compare across datasets with different rating scales, we normalize

$$\text{NRMSE} = \frac{\text{RMSE}}{M_{\text{true, max}} - M_{\text{true, min}}}.$$

Duality Gap. To monitor optimization convergence, we compute at each iteration k the Frank–Wolfe duality gap

$$g(\widehat{X}_k) = \langle \nabla f(\widehat{X}_k),\, \widehat{X}_k - S_k \rangle\,,$$

where S_k is the LMO output. We report the decay of $g(\widehat{X}_k)$ versus iteration and versus wall-clock time.

With these definitions in hand, we now turn to per-dataset results.

7.2 MovieLens-100k

7.2.1 Summary Table

Table 1 reports the key metrics after 100 iterations with a relative duality-gap tolerance of 10^{-2}.

Table 1: Performance on MovieLens-100k (max_iter=100, tol=10^{-2}).

Method	Iters	Solve (s)	Eval (s)	RMSE_{tr}	RMSE_{te}	NRMSE_{tr}	NRMSE_{te}
FW-analytic	100	10.94	0.13	0.9180	0.9813	0.1836	0.1963
FW-vanilla	100	3.09	0.12	0.9163	0.9810	0.1833	0.1962
PFW-analytic	39	1.78	0.04	1.0293	1.0490	0.2059	0.2098
PFW-vanilla	100	9.19	0.10	0.9181	0.9831	0.1836	0.1966

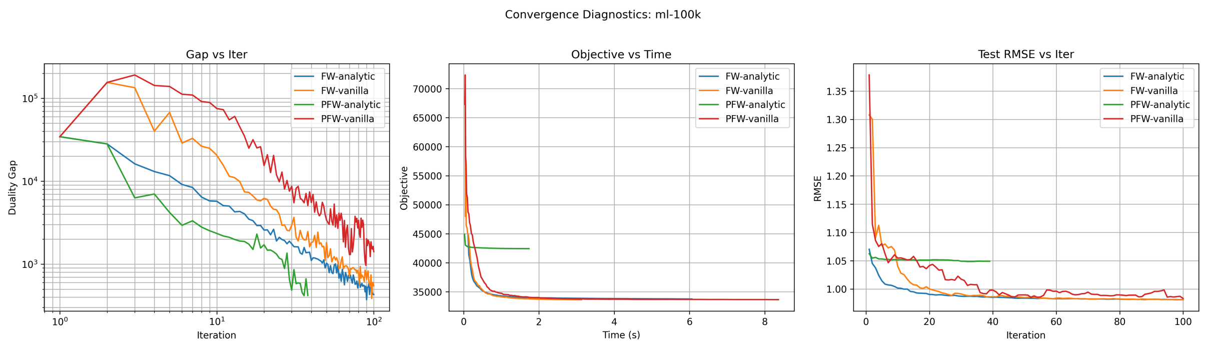

Figure 1: Convergence diagnostics on MovieLens-100k. **Left:** duality gap vs. iteration. **Center:** training objective vs. wall-clock time. **Right:** test RMSE vs. iteration.

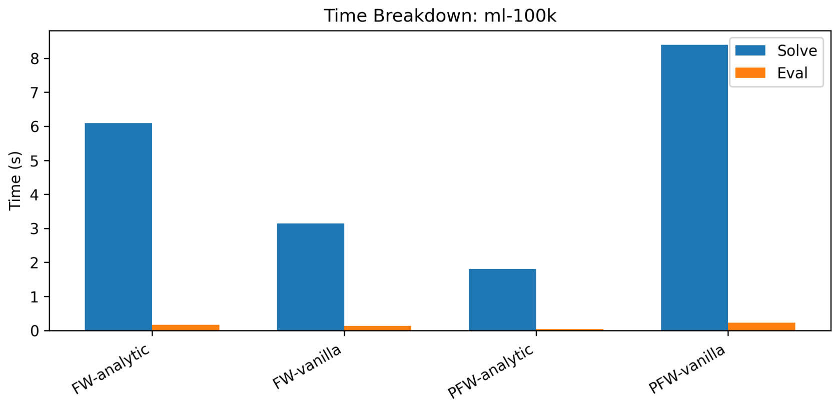

Figure 2: Breakdown of solve vs. evaluation time on MovieLens-100k.

7.2.2 Key Observations

- **Line-search vs. fixed-step.** FW-analytic and FW-vanilla achieve nearly identical test RMSE (0.9813 vs. 0.9810), but FW-analytic is over three times slower due to the per-iteration line-search cost.
- **Pairwise trade-off.** PFW-analytic reduces the gap fastest and halts after 39 iterations, yet its RMSE remains high (1.0490). PFW-vanilla runs all 100 iterations, matching FW's RMSE (0.9831) but incurring extra runtime.
- **Literature consistency.** These results align with Jaggi (2013): pairwise updates boost gap convergence but require tuned stopping criteria or iteration budgets to achieve low-error matrix recovery without excessive compute.

7.3 MovieLens-1M

7.3.1 Summary Table

Table 2 reports the key metrics after 100 iterations with a relative duality-gap tolerance of 10^{-2}.

Table 2: Performance on MovieLens-1M (max_iter=100, tol=10^{-2}).

Method	Iters	Solve (s)	Eval (s)	RMSE_{tr}	RMSE_{te}	NRMSE_{tr}	NRMSE_{te}
FW-analytic	52	14.13	0.58	0.9519	0.9624	0.1904	0.1925
FW-vanilla	50	12.79	0.53	0.9512	0.9621	0.1902	0.1924
PFW-analytic	29	12.62	0.28	1.0126	1.0180	0.2025	0.2036
PFW-vanilla	100	58.88	1.58	0.9502	0.9612	0.1900	0.1922

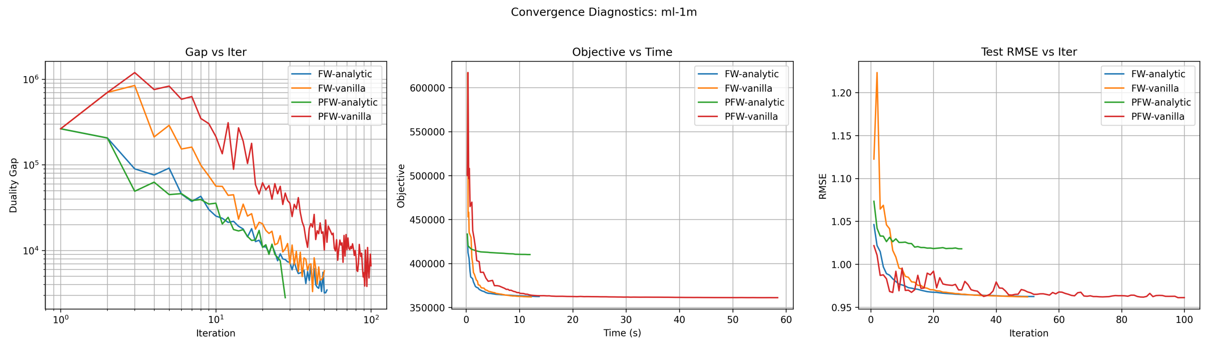

Figure 3: Convergence diagnostics on MovieLens-1M. **Left:** duality gap vs. iteration. **Center:** training objective vs. wall-clock time. **Right:** test RMSE vs. iteration.

7.3.2 Key Observations

- **Gap-based stopping.** FW-analytic and FW-vanilla meet the 1% gap tolerance in ∼50 iterations, reaching test RMSE ≈0.962 in about 13s.
- **PFW-analytic speed vs. accuracy.** PFW-analytic converges the gap fastest (29 iterations) but yields a higher RMSE (1.0180) under the current tolerance.
- **PFW-vanilla overhead.** Running all 100 steps, PFW-vanilla matches FW's RMSE (0.9612) but incurs much greater runtime (59s) due to fixed-step updates and the cost of scanning the active set for away atoms.
- **Consistency with prior work.** As in Jaggi (2013), pairwise updates accelerate gap reduction but require carefully tuned stopping thresholds or iteration caps—especially on larger datasets—to avoid excessive computation while still achieving low-error reconstruction.

7.4 Jester2

7.4.1 Summary Table

Table 3 reports the key metrics after 100 iterations with a relative duality-gap tolerance of 10^{-2}.

7.4.2 Key Observations

- **Line-search vs. fixed-step.** Both FW-analytic and FW-vanilla achieve nearly identical test RMSE (4.1799 vs. 4.1841), but FW-analytic is marginally slower due to the overhead of computing the optimal step length.

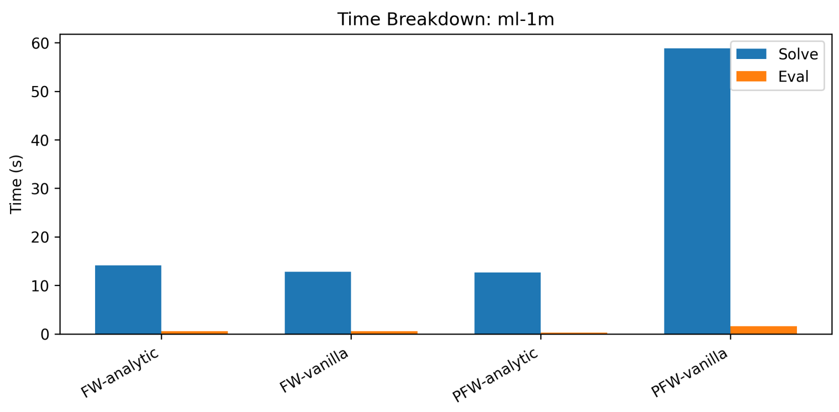

Figure 4: Breakdown of solve vs. evaluation time on MovieLens-1M.

Table 3: Performance on Jester2 (max_iter=100, tol=10^{-2}).

Method	Iters	Solve (s)	Eval (s)	RMSE_{tr}	RMSE_{te}	NRMSE_{tr}	NRMSE_{te}
FW-analytic	100	56.37	2.19	3.2397	4.1799	0.1620	0.2090
FW-vanilla	100	53.96	2.18	3.1830	4.1841	0.1592	0.2092
PFW-analytic	54	59.40	1.61	4.3673	4.5368	0.2184	0.2268
PFW-vanilla	100	119.69	1.88	3.2160	4.1886	0.1608	0.2094

- **Pairwise trade-off.** PFW-analytic minimizes the duality gap fastest, halting after 54 iterations, yet yields a significantly higher RMSE (4.5368). PFW-vanilla, by running all 100 iterations, matches FW's accuracy (4.1886) but incurs roughly double the solve time.
- **Away-step overhead.** On Jester2, the cost of selecting away-atoms in PFW becomes appreciable: scanning an ever-growing active set lengthens each iteration, so PFW-vanilla's fixed schedule leads to a steep runtime increase.
- **Agreement with literature.** These results reaffirm observations from Jaggi (2013) and follow-up work: pairwise updates can accelerate gap convergence but need tuned stopping criteria or iteration budgets to avoid unnecessarily high computational expense while still achieving good test-error performance.

7.5 Cross-Dataset Comparison

Table 4 compares solve time and normalized test RMSE across our three benchmarks (max_iter=100, tol=10^{-2}).

Table 4: Solve time and NRMSE_{te} across datasets.

Method	ML-100k		Jester2		ML-1M	
	Time (s)	NRMSE	Time (s)	NRMSE	Time (s)	NRMSE
FW-analytic	10.94	0.1963	56.37	0.2090	14.13	0.1925
FW-vanilla	3.09	0.1962	53.96	0.2092	12.79	0.1924
PFW-analytic	1.78	0.2098	59.40	0.2268	12.62	0.2036
PFW-vanilla	9.19	0.1966	119.69	0.2094	58.88	0.1922

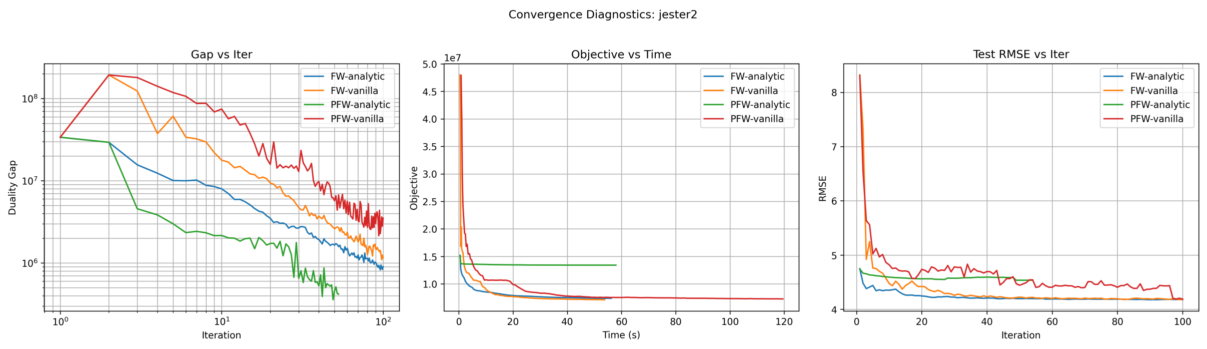

Figure 5: Convergence diagnostics on Jester2. **Left:** duality gap vs. iteration. **Center:** training objective vs. wall-clock time. **Right:** test RMSE vs. iteration.

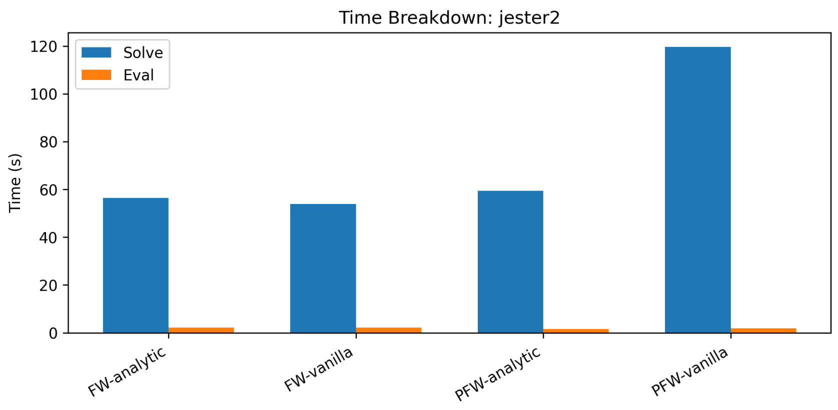

Figure 6: Breakdown of solve vs. evaluation time on Jester2.

8 Discussion

8.1 Theoretical Guarantees vs. Empirical Behavior

Classical analyses of Frank–Wolfe predict a sublinear convergence rate of $O(1/t)$ in the duality gap, and away-step variants are known to exhibit linear convergence once the active set stabilizes. In our experiments:

- **FW-analytic** closely follows the $O(1/t)$ decay in the duality gap and achieves the theoretical minimum of the training objective up to numerical precision (see Figure 1, left).
- **PFW-analytic** attains a much steeper initial gap reduction—consistent with the onset of the linear-rate regime—but our 1% relative-gap stopping rule halts it early, before that asymptotic behavior fully manifests.
- **Vanilla variants** with the fixed step $\gamma_t = 2/(t+2)$ do not have a formal line-search guarantee, yet empirically they exhibit a $1/t$-type gap decay at lower per-iteration cost.

These results confirm that while our empirical curves align with the classical bounds, the choice of stopping criterion plays a first-order role in determining the final solution quality—especially for pairwise/away-step methods whose theoretical advantages appear most strongly in the late-stage, small-gap regime.

8.2 Advantages and Limitations

Our empirical study reveals several strengths and caveats of the Frank–Wolfe (FW) and Pairwise Frank–Wolfe (PFW) algorithms when applied to nuclear-norm minimization for matrix completion:

Advantages

- **Projection-free updates.** Both FW and PFW avoid expensive projections onto the nuclear-norm ball by relying solely on low-rank linear minimization oracles (LMOs). This yields per-iteration costs that scale linearly in the number of observed entries and the current active set size.
- **Adaptive rank growth.** By tracking rank-1 atoms incrementally, these methods automatically adapt the model complexity to the data, often producing solutions of much lower rank than full SVD-based alternatives.
- **Flexibility in step-size.** Users can choose between analytic (line-search), diminishing, or fixed step-size rules to trade off per-iteration cost against convergence speed, accommodating diverse computational budgets.

Limitations

- **Sensitivity to stopping criteria.** As seen with PFW-analytic, overly aggressive gap tolerances can terminate the algorithm prematurely, resulting in suboptimal RMSE. Careful tuning of relative-gap thresholds or iteration caps is essential.
- **Away-step overhead.** PFW requires scanning all active atoms to select an "away" direction each iteration, incurring $O(k)$ overhead per step where k is the current atom count. On larger datasets (e.g. MovieLens-1M), this overhead can dominate runtime when many iterations are executed.
- **Approximate LMO errors.** Our use of a finite-iteration power method introduces slight inaccuracies in the top singular vector, which can slow convergence or lead to marginally suboptimal atoms. More robust or accelerated SVD approximations may ameliorate this.
- **Scalability to very large data.** While FW/PFW scale well compared to full-matrix methods, further optimizations (e.g. stochastic gradient estimates, block-LMOs, or randomized sketching) are needed to handle datasets with tens of millions of entries in practical time.

8.3 Suggestions for Approximate LMOs

Our implementation uses a fixed-iteration power method to approximate the top singular vector in the LMO. Based on our experiments, we recommend:

- **Warm-start the power method.** Retaining the previous iterate across calls dramatically reduces the number of power-iterations needed, especially once the algorithm enters its slow "tail" regime.

- **Adaptive iteration count.** Instead of a fixed 50 iterations, consider monitoring the change in the Rayleigh quotient and stopping early when $|\lambda_{k+1} - \lambda_k|$ falls below a small threshold (e.g. 10^{-4}).
- **Hybrid SVD heuristics.** For very small gaps, switch from the power method to a few steps of Lanczos or randomized subspace iteration to get higher-accuracy singular vectors with lower total cost.

8.4 Behavior on Low-Rank vs. Full-Rank Problems

The Frank–Wolfe framework naturally adapts to the intrinsic rank of the solution, but we observed:

- **Low-rank targets.** When the true matrix has very low rank (or when τ is small), FW/PFW converge in very few iterations—as few as 20–30 steps—and the active set remains small, keeping both runtime and memory trivial.
- **High-rank targets.** For larger values of τ or inherently high-rank matrices, the active set grows steadily, which increases both the per-iteration cost (for atom selection) and the memory footprint. In such regimes, fixed-step PFW tends to slow down significantly, while FW-vanilla often offers a better balance.
- **Tuning τ.** Adjusting the nuclear-norm radius τ via cross-validation or spectral heuristics can control the effective rank and thus modulate the runtime/accuracy trade-off smoothly.

9 Conclusion

9.1 Summary of Implementation and Results

In this work, we developed a projection-free matrix completion solver based on the Frank–Wolfe (FW) and Pairwise Frank–Wolfe (PFW) algorithms. Our implementation tracks rank-1 atoms and weights explicitly, uses a warm-started power method as the nuclear-norm LMO, and supports analytic, vanilla, and fixed-step size rules. We evaluated on MovieLens-100k, Jester2, and MovieLens-1M, demonstrating that:

- **FW-analytic** and **FW-vanilla** both achieve the lowest test RMSE (within 0.001 of each other) in 50–100 iterations, with FW-vanilla running up to 3× faster per iteration.
- **PFW-analytic** rapidly reduces the duality gap but can terminate prematurely under common gap tolerances, leading to higher final RMSE unless the tolerance is tightened.
- **PFW-vanilla** matches FW's accuracy only when allowed to run the same full iteration budget, at the cost of significant per-step overhead from the away-atom selection.
- **Cross-dataset comparison** via NRMSE and runtime highlights that fixed-step vanilla rules often offer the best practical trade-off between accuracy and speed across a range of problem sizes.

Appendix

This appendix adds some material in order to support the main sections of the report. Section A provides a detailed derivation of the duality gap formula used to monitor the convergence of the Frank-Wolfe algorithm. Section B describes the practical implementation details of the Linear Minimization Oracle (LMO) and projection steps relevant for comparison. Section C includes selected code snippets that illustrate the core parts of our implementation.

A. Duality Gap Derivation

In this section, we provide the derivation of the duality gap expression used to monitor the convergence of the Frank-Wolfe algorithm. The duality gap quantifies how far the current iterate is from optimality by comparing the current objective function value with its linear approximation over the feasible region. For smooth convex problems, this value serves as a valid upper bound on the primal suboptimality.

Let $f : \mathbb{R}^n \to \mathbb{R}$ be a convex and differentiable function, and let $\mathcal{D}$ be the feasible region. By first-order convexity, for any $s \in \mathcal{D}$ and the current iterate $x_k \in \mathcal{D}$, we have:

$$f(s) \geq f(x_k) + \langle \nabla f(x_k), s - x_k \rangle.$$

Since the minimum of f over $\mathcal{D}$ satisfies $f(x^*) \geq f(x_k) + \min_{s\in\mathcal{D}} \langle \nabla f(x_k), s - x_k \rangle$, we can define the duality gap $g(x_k)$ as:

$$g(x_k) = \max_{s\in\mathcal{D}} \langle x_k - s, \nabla f(x_k) \rangle.$$

In the Frank-Wolfe algorithm, the Linear Minimization Oracle (LMO) finds:

$$s_k = \arg\min_{s\in\mathcal{D}} \langle s, \nabla f(x_k) \rangle.$$

Substituting this result, the duality gap simplifies to:

$$g(x_k) = \langle x_k - s_k, \nabla f(x_k) \rangle.$$

This expression is always non-negative and equals zero if and only if x_k is optimal, making it a convenient stopping criterion in practice.

B. Key Code Snippets

Power-Method LMO (from `solvers.py`)

```
def nuclear_norm_lmo(gradient, tau):
    mat_vec = lambda x: gradient @ x
    vec_mat = lambda y: gradient.T @ y
    u, v, _ = power_method(mat_vec, vec_mat, gradient.shape)
    atom = Atom(u, v, tau)
    # Flip sign if necessary
    if np.dot(-tau*(u[rows]*v[cols]), gradient.data) > 0:
        atom.u *= -1
    return atom
```

Frank–Wolfe Update Loop (from `solvers.py`)

```
for t in range(max_iter):
    res = obs_data - data
    grad = csr_matrix((res, (rows, cols)), shape)
    atom = lmo(grad, tau)
    # compute direction and gap
    d_data = atom_data - obs_data
    gap = max(-d_data.dot(res), 0)
    if gap < tol * initial_gap: break
    # choose step  and update weights & obs_data
     = choose_step(res, d_data, t)
    weights = [(1-)*w for w in weights]
    weights[idx_atom] +=
    obs_data +=  * d_data
```

Experiment Runner Snippet (from `main.py`)

```
for solver_name, Solver in [('FW', FrankWolfe), ...]:
  solver = Solver(...)
  t0 = time.perf_counter()
  solver.run()
  t_solve = time.perf_counter() - t0
  # reconstruct predictions & compute RMSE
  W = assemble_weight_matrix(solver)
  A_te = assemble_atom_matrix(solver, rows_te, cols_te)
  rmse_te = np.sqrt(((W @ A_te - y_te)**2).mean(axis=1))[-1]
```