Matrix Completion Problem

The Netflix Challenge

The Problem:

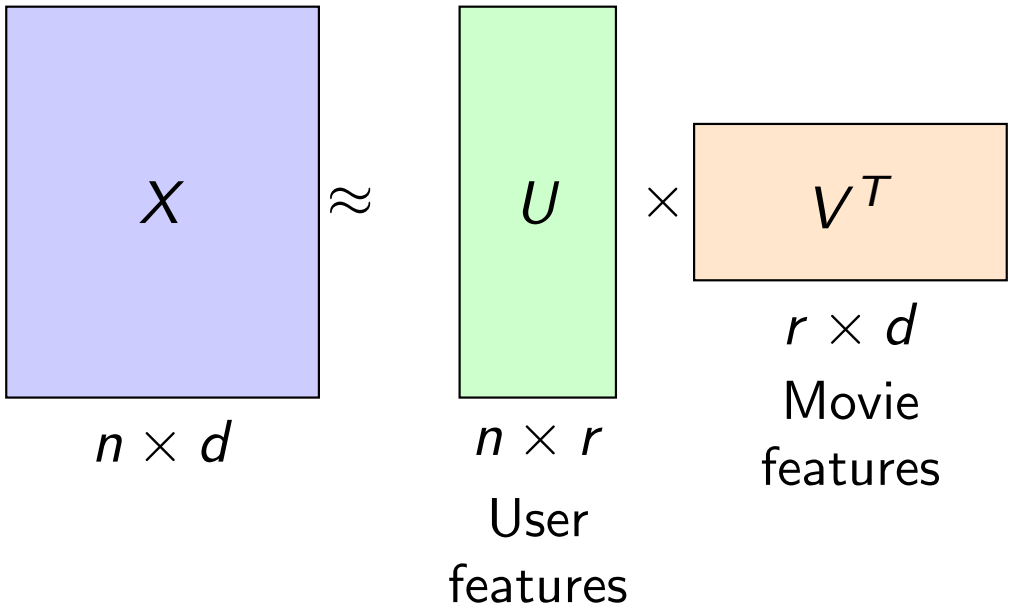
- Netflix has millions of users
- Thousands of movies/shows
- Most users rate only a small fraction of content
- Goal: Predict missing ratings

Why it matters:

- Better recommendations
- Increased user engagement
- $1M prize in 2009!

Key Insight: Low-Rank Structure

Intuition: User preferences are driven by a small number of latent factors



Examples of latent factors:

- Genre preferences (action, comedy, drama)
- Era preferences (classic vs. modern)
- Audience type (family, adult, teen)

Key assumption: $r \ll \min(n, d)$

Low-rank assumption

Low-rank assumption

For $n_1 \times n_2$ matrix M of rank r, assume the $\min(n_1, n_2) \gg r$.

- Why this assumption is needed?
- For simplicity, think about an $n \times n$ matrix M of rank r.
- It has $(2n - r)r$ degrees of freedom.
  - Degree of freedom is calculated by counting parameters in the SVD.
    - (The number of singular values)
      $+$ (degree of freedom of left singular vectors)
      $+$ (degree of freedom of right singular vectors)
      $= r + ((2n - r - 1)r)/2 + ((2n - r - 1)r)/2$
  - Considerably smaller than n^2.

Matrix Completion: Formal Setup

Given:

- A partially observed matrix with entries on index set $\Omega \subset \{1, \ldots, n\} \times \{1, \ldots, d\}$
- Observations: $\{X_{ij} : (i, j) \in \Omega\}$
- $|\Omega| = m$ (number of observed entries)

Goal: Recover the complete matrix $X \in \mathbb{R}^{n \times d}$ of rank r where

$$r \ll \min(n, d)$$

Assumption: The true underlying matrix has low rank, reflecting the presence of latent structure in the data.

Notation and Setup

Key objects:

- $X \in \mathbb{R}^{n \times d}$: True complete matrix (rank r)
- Ω: Set of observed indices
- $\mathcal{P}_\Omega(X)$: Projection onto observed entries

$$[\mathcal{P}_\Omega(X)]_{ij} = \begin{cases} X_{ij} & \text{if } (i,j) \in \Omega \\ 0 & \text{otherwise} \end{cases}$$

- Sampling fraction: $\rho = \frac{m}{nd} = \frac{|\Omega|}{nd}$

Challenge: This is an ill-posed problem without additional assumptions!

- Infinitely many matrices agree with observed entries
- Need to exploit low-rank structure

Ideal Estimator: Rank Minimization

Natural approach: Find the lowest-rank matrix consistent with observations

Rank Minimization Problem

$$\begin{aligned} \underset{M \in \mathbb{R}^{n \times d}}{\text{minimize}} \quad & \text{rank}(M) \\ \text{subject to} \quad & \mathcal{P}_\Omega(M) = \mathcal{P}_\Omega(X) \end{aligned}$$

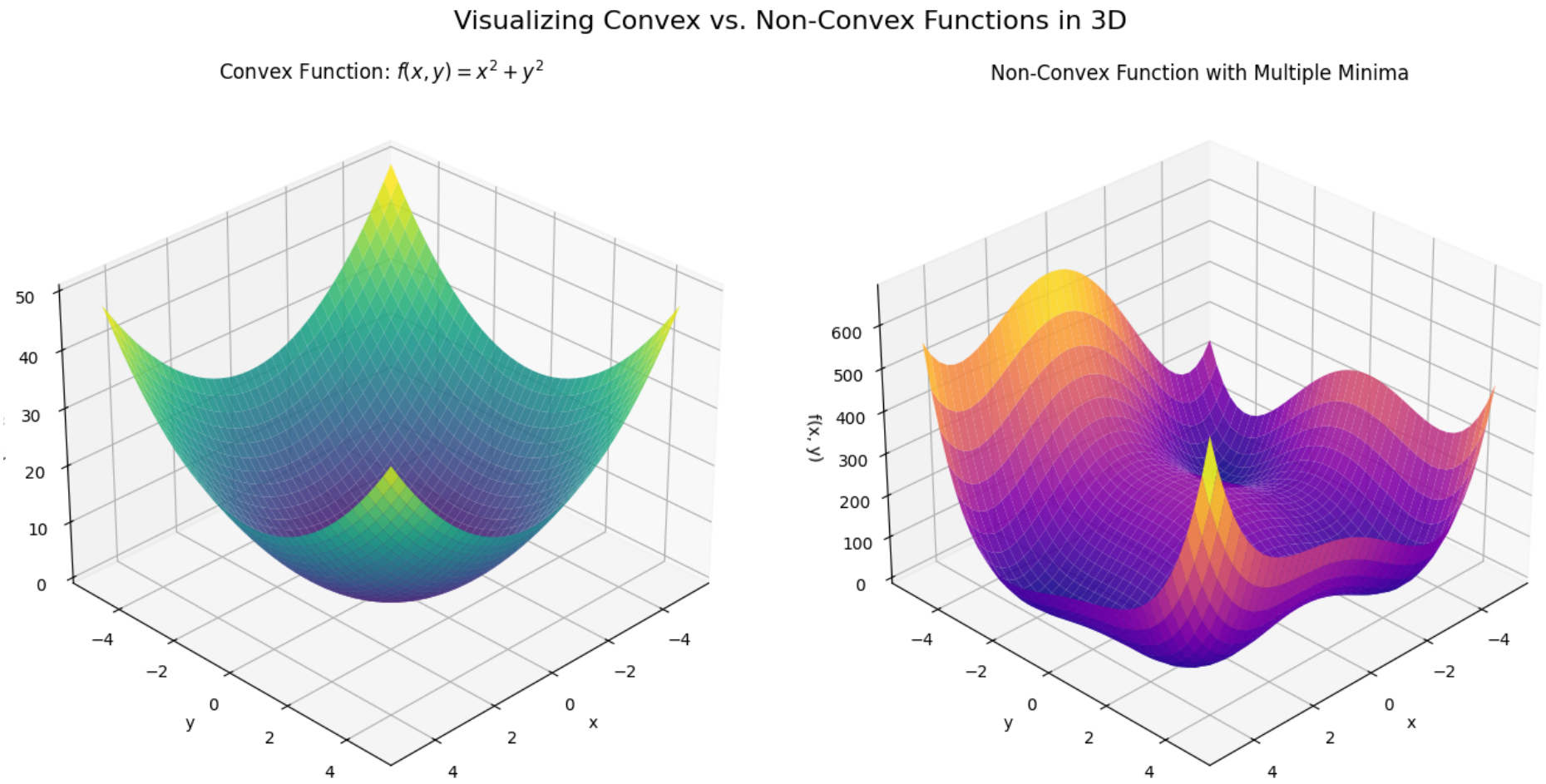
Equivalently:

$$\underset{M \in \mathbb{R}^{n \times d}}{\text{minimize}} \quad \text{rank}(M) \quad \text{s.t.} \quad M_{ij} = X_{ij} \text{ for all } (i,j) \in \Omega$$

Problem: This is NP-hard!

- Rank function is non-convex
- Combinatorial in nature
- No efficient algorithms for general case

Non-convex vs Convex I



Non-convex vs Convex II

Convex.

- \+ Single global minimum
- \+ Simple to optimize; guaranteed convergence to global minimum
- – Limited complexity: not suitable for real-world problems.

Non-convex.

- \+ Suitable for modeling landscaopes related to real-world problems.
- – Multiple local minima
- – Convergence challanges: optimization methods can be stuck in local minima.

Nuclear Norm: The Convex Relaxation

Key idea: Replace rank with nuclear norm!

Nuclear Norm Definition

The **nuclear norm** (or trace norm) of a matrix M is:

$$\|M\|_* = \sum_{i=1}^{\min(n,d)} \sigma_i(M)$$

where $\sigma_1(M) \geq \sigma_2(M) \geq \cdots \geq \sigma_{\min(n,d)}(M) \geq 0$ are the singular values of M.

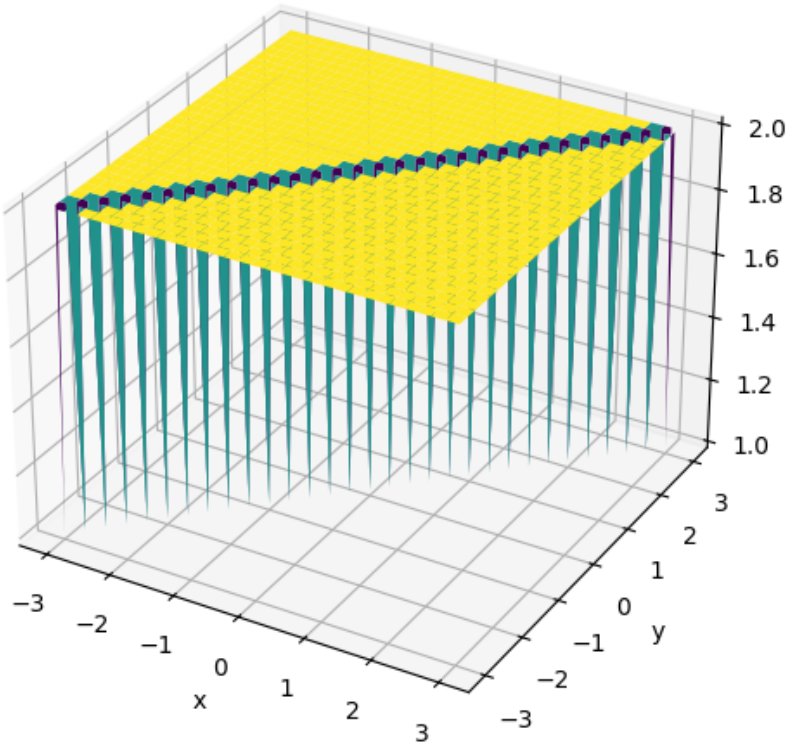
Properties:

- Convex function
- Sum of singular values = trace of $\sqrt{M^T M}$
- ℓ_1 norm of the singular value vector

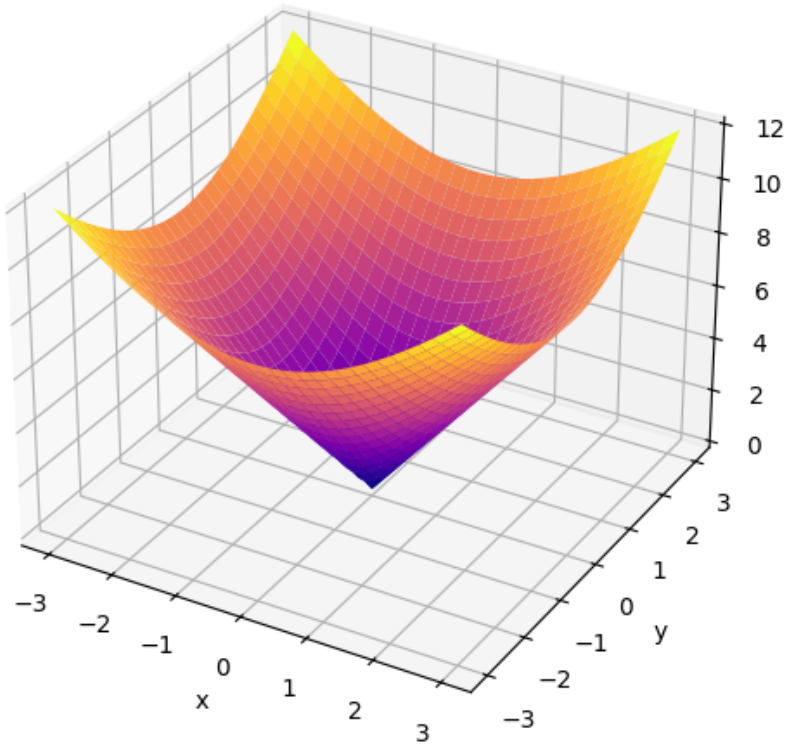
Rank vs Nuclear norm

$$M = [[\sqrt{4}x\ (x+y)],[(x+y)\ \sqrt{4}y]]$$

Rank of Matrix (Non-Convex)



Nuclear Norm (Convex Approximation)



Nuclear Norm Intuition

Analogy with sparse vectors:

Goal	Non-convex	Convex relaxation
Sparse vector	$\|x\|_0 = \#\{\text{non-zeros}\}$	$\|x\|_1 = \sum_i \|x_i\|$
Low-rank matrix	$\text{rank}(M)$	$\|M\|_* = \sum_i \sigma_i(M)$

Why it works:

- ℓ_1 norm promotes sparsity in vectors
- Nuclear norm promotes sparsity in singular values
- Sparsity in singular values = low rank!

Convex Formulation

Nuclear Norm Minimization

$$\begin{aligned} \underset{M \in \mathbb{R}^{n \times d}}{\text{minimize}} \quad & \|M\|_* \\ \text{subject to} \quad & \mathcal{P}_\Omega(M) = \mathcal{P}_\Omega(X) \end{aligned}$$

Advantages:

- Convex optimization problem
- Can be solved efficiently (though n, d large)
- Strong theoretical guarantees
- Exact recovery under mild conditions

When Does Nuclear Norm Work ? Theory I

Candes and Recht 2008:

Random Orthogonal Model

$$M = \sum_{k=1}^{r} \sigma_k u_k v_k^T$$

- $\{u_k\}_{k=1}^r$ is selected uniformly at random among all families of orthonormal vectors.
- The same for $\{v_k\}$.
- That is: M has an intrinsic low rank.

When Does Nuclear Norm Work ? Theory II

Theorem

Let $M \in \mathbb{R}^{n_1 \times n_2}$ be a rank r matrix sampled from the random orthogonal model. Let $n = \max(n_1, n_2)$. Suppose we observe m entries of M uniformly at random. Then, there are constants c and C such that if

$$m \geq C n^{5/4} r \log n$$

then the minimizer to the Nuclear norm minimization is unique and equal to M with probability $1 - cn^{-3}$.

When Does Nuclear Norm Work ? Theory III

Implications:

- Under the hypothesis of the Theorem, there is a unique low-rank matrix which is consistent with the observed entries.
- This matrix can be recovered by a convex optimization algorithm.
- So the nuclear norm relaxation is formally equivalent to the NP hard problem.

Computational Challenge

Problem: Even though convex, directly solving nuclear norm minimization is expensive for large matrices.

Solution: Singular Value Thresholding (SVT) Algorithm

- Iterative algorithm
- Exploits problem structure
- Only requires SVD of sparse matrices
- Efficient for large-scale problems

Key Ingredient: Soft-Thresholding of Singular Values

Scalar soft-thresholding:

$$\mathcal{S}_\tau(x) = \text{sign}(x) \cdot \max(|x| - \tau, 0) = \begin{cases} x - \tau & \text{if } x > \tau \\ 0 & \text{if } |x| \leq \tau \\ x + \tau & \text{if } x < -\tau \end{cases}$$

Matrix soft-thresholding: Let $M = U\Sigma V^T$ be the SVD with $\Sigma = \text{diag}(\sigma_1, \ldots, \sigma_r)$

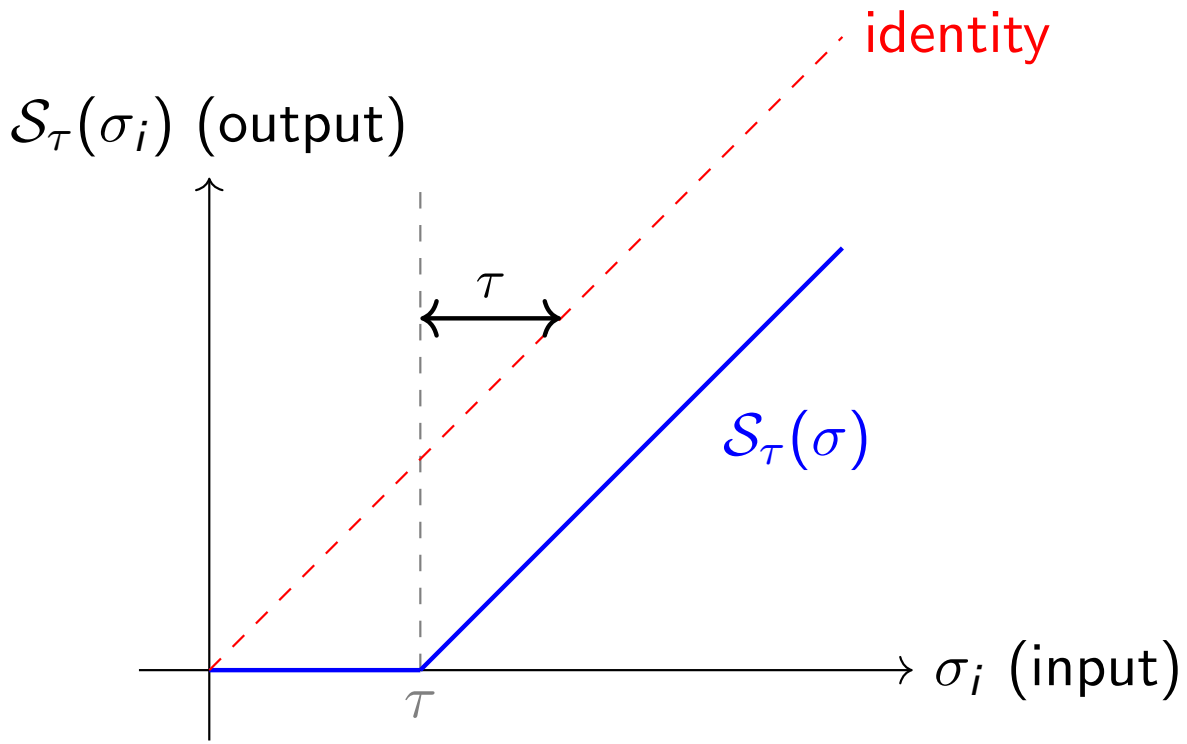
Singular Value Thresholding Operator

$$\mathcal{D}_\tau(M) = U\mathcal{S}_\tau(\Sigma)V^T = \sum_{i=1}^{r} \mathcal{S}_\tau(\sigma_i) u_i v_i^T$$

where $\mathcal{S}_\tau(\Sigma) = \text{diag}(\mathcal{S}_\tau(\sigma_1), \ldots, \mathcal{S}_\tau(\sigma_r))$

Interpretation: Shrinks singular values by τ, sets small ones to zero

Soft-Thresholding Visualization



Singular values below τ are set to zero

Effect: Promotes low rank by eliminating small singular values

Why Soft-Thresholding?

Connection between $\mathcal{D}_\tau$ and $\tau\|\cdot\|_*$

The soft-thresholding operator $\mathcal{D}_\tau$ can be written also as:

$$\mathcal{D}_\tau(Y) = \arg\min_M \left\{ \frac{1}{2}\|M - Y\|_F^2 + \tau\|M\|_* \right\}$$

This means:

- Soft-thresholding solves a regularized least-squares problem
- Balances fidelity to Y with nuclear norm penalty
- Closed-form solution via SVD!

Computation:

1. Compute SVD: $Y = U\Sigma V^T$
2. Threshold: $\Sigma' = \mathcal{S}_\tau(\Sigma)$
3. Reconstruct: $\mathcal{D}_\tau(Y) = U\Sigma' V^T$

SVT Algorithm: Setup

Reformulation: Consider the unconstrained problem

$$\underset{M}{\text{minimize}} \quad \tau\|M\|_* + \frac{1}{2}\|\mathcal{P}_\Omega(M) - \mathcal{P}_\Omega(X)\|_F^2$$

for appropriate choice of $\tau > 0$.

Equivalent constrained form:

$$\begin{aligned} \underset{M}{\text{minimize}} \quad & \tau\|M\|_* \\ \text{subject to} \quad & \mathcal{P}_\Omega(M) = \mathcal{P}_\Omega(X) \end{aligned}$$

The SVT algorithm solves this using an iterative projection method.

SVT Algorithm: The Iteration

Singular Value Thresholding Algorithm

Input: Observations $\mathcal{P}_\Omega(X)$, parameter $\tau > 0$, step size $\delta \in (0, 2)$

Initialize: $Y_0 = c_o \delta P_\Omega(X)$, $k = 0$

Repeat until convergence:

1. Compute: $M_k = \mathcal{D}_\tau(Y_k)$
2. Update: $Y_{k+1} = Y_k + \delta \mathcal{P}_\Omega(X - M_k)$
3. $k \leftarrow k + 1$

Output: M_k

Interpretation:

- Step 1: Shrink singular values (promote low rank)
- Step 2: Project back to agreement with observations
- Alternates between rank minimization and data fidelity

SVT Algorithm: Detailed Steps

Step-by-step breakdown:

Step 1: Singular value thresholding

- Compute SVD: $Y_k = U_k \Sigma_k V_k^T$
- Apply soft-threshold: $M_k = U_k \mathcal{S}_\tau(\Sigma_k) V_k^T$
- This gives a low-rank approximation to Y_k

Step 2: Gradient update

- Compute residual: $R_k = X - M_k$ on Ω
- Update: $Y_{k+1} = Y_k + \delta \mathcal{P}_\Omega(R_k)$
- Move in direction that reduces constraint violation
- δ controls step size (typically $\delta = 1.2$)

Key: Only need to compute SVD once per iteration, and only for top singular vectors!

SVT: Convergence Properties

Theoretical guarantees:

Theorem (Cai, Candès, Shen 2010)

For $\delta \in (0, 2)$, the SVT algorithm converges to the optimal solution of the nuclear norm minimization problem.

Convergence rate:

- Linear convergence: $\|M_k - M^*\|_F \leq C\rho^k$ for some $\rho < 1$
- Rate depends on δ and problem conditioning
- Best choice: $\delta \approx 1.2$ to 1.5 (empirically)

Stopping criterion:

$$\frac{\|\mathcal{P}_\Omega(M_k) - \mathcal{P}_\Omega(X)\|_F}{\|\mathcal{P}_\Omega(X)\|_F} < \epsilon$$

for desired tolerance ϵ (e.g., $\epsilon = 10^{-4}$).

SVT: Computational Complexity

Per iteration cost:

1. **SVD computation:** $O(r^2(n+d))$ using partial SVD
   - Only need top r singular vectors
   - Can use iterative methods (Lanczos, power method)
   - Much cheaper than full SVD: $O(\min(n^2 d, nd^2))$
2. **Projection update:** $O(m)$ where $m = |\Omega|$
   - Only update observed entries
   - Sparse operation

Overall complexity:

$$O(K \cdot r^2(n+d))$$

where K is the number of iterations.

Scalability: Efficient for large n, d when r is small!

SVT: Key Properties

Advantages:

- **Simple implementation:** Only requires SVD and basic operations
- **Memory efficient:** Can work with sparse representations
- **Scalable:** Handles large matrices when rank is small
- **Guaranteed convergence:** Provably converges to optimal solution
- **Flexible:** Easy to add constraints or regularization

Practical considerations:

- Choice of τ: Cross-validation or theoretical bounds
- Warm start: Initialize with previous solution
- Adaptive δ: Can adjust step size during iteration
- Early stopping: Don't need exact convergence

SVT: Parameter Selection

Choosing τ:

1. **Theoretical choice:**

$$\tau = \gamma\sqrt{nd}$$

where $\gamma \approx 5$ to 10 (rule of thumb)

2. **Cross-validation:**
   - Hold out some observed entries
   - Select τ minimizing validation error
   - Grid search over $\tau \in [1, 10\sqrt{nd}]$

3. **Adaptive:**

$$\tau_k = \tau_0 \cdot \alpha^k$$

Decrease τ over iterations (continuation method)

Choosing δ: Typical values $\delta \in [1.0, 1.5]$

- Larger δ: Faster initial progress
- Smaller δ: More stable convergence

SVT: Practical Algorithm

Algorithm 1 Practical SVT Implementation

Require: $\mathcal{P}_\Omega(X)$, τ, δ, ϵ, $k_{\max}$

Initialize: $Y \leftarrow c_0 \delta P_\Omega(X)$, $k \leftarrow 0$
while $k < k_{\max}$ **do**
  $[U, \Sigma, V] \leftarrow \text{partialSVD}(Y, r_k)$ {Top r_k components}
  $\Sigma' \leftarrow \max(\Sigma - \tau, 0)$ {Soft-threshold}
  $M \leftarrow U \Sigma' V^T$
  $R \leftarrow \mathcal{P}_\Omega(X - M)$ {Residual on Ω}
  if $\|R\|_F / \|\mathcal{P}_\Omega(X)\|_F < \epsilon$ **then**
    break {Converged}
  end if
  $Y \leftarrow Y + \delta R$
  $r_{k+1} \leftarrow \text{rank}(M) + 1$ {Adaptive rank}
  $k \leftarrow k + 1$
end while
return M

Alternative Algorithms

Other popular methods:

Algorithm	**Key Idea**
OptSpace	Low-rank factorization on manifold
SOFT-IMPUTE	Iterative soft-thresholding with warm starts
FPCA	Fixed point iteration with continuation
APGL	Accelerated proximal gradient (Nesterov)
LMaFit	Low-rank matrix fitting (non-convex)
ALS	Alternating least squares (factorization)

Comparison with SVT:

- SVT: Simple, guaranteed convergence, may be slower
- OptSpace: Very fast, but needs good initialization
- SOFT-IMPUTE: Similar to SVT, good practical performance
- Non-convex methods: Faster but no guarantees

SVT vs. Factorization Methods

SVT (Convex)

- \+ Global optimum
- \+ Theoretical guarantees
- \+ No initialization needed
- – Slower convergence
- – SVD per iteration
- – Memory for full M

Factorization (Non-convex)

- \+ Very fast
- \+ Memory efficient
- \+ No SVD needed
- – Local minima
- – Initialization matters
- – Rank must be specified

Recommendation:

- Small/medium problems: Use SVT
- Large-scale: Consider factorization or SOFT-IMPUTE
- When guarantees matter: Use SVT

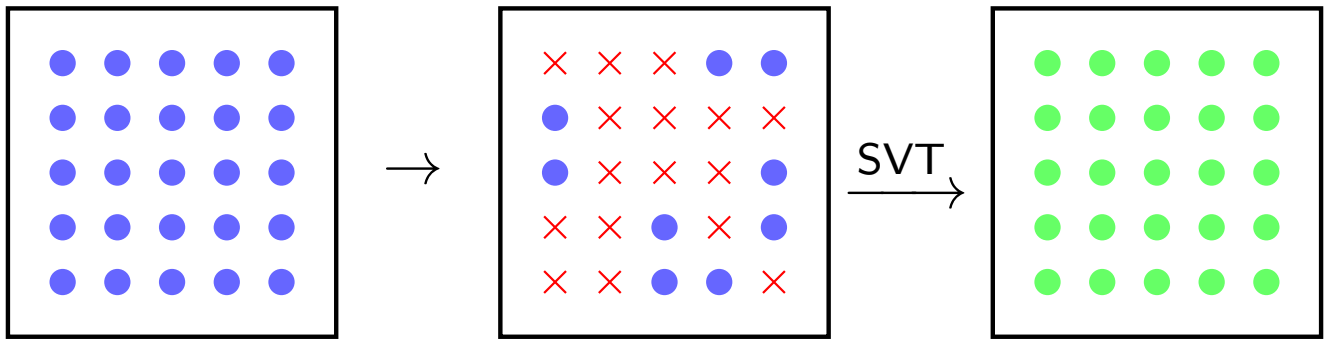
Applications of Matrix Completion

1. **Computer Vision**
   - Image inpainting
   - Video completion
   - Background subtraction
2. **System Identification**
   - Missing sensor data
   - Network topology inference
   - Wireless sensor networks
3. **Bioinformatics**
   - Gene expression data
   - Protein interaction networks
   - Drug response prediction

Applications (continued)

4. **Quantum State Tomography**
   - Reconstruct quantum states
   - Reduce measurement complexity
5. **Collaborative Filtering**
   - Product recommendations
   - Social network analysis
   - Ad targeting
6. **Financial Data**
   - Portfolio optimization with missing returns
   - Risk assessment
   - Market prediction

Example: Image Inpainting



Key insight: Natural images have low-rank structure when properly arranged (e.g., patch-based representations).

Summary: Key Takeaways

1. **Problem:** Complete partially observed low-rank matrix
   - Motivated by recommendation systems
   - Assumes rank $r \ll \min(n, d)$
2. **Approach:** Convex relaxation
   - Replace rank with nuclear norm
   - Provable recovery guarantees
3. **Algorithm:** Singular Value Thresholding
   - Iterative soft-thresholding of singular values
   - Computationally efficient for low-rank matrices
   - Guaranteed convergence
4. **Applications:** Widespread in machine learning and signal processing