Riemannian Gradient Descent Methods for Graph-Regularized Matrix Completion

Shuyu Dong[†], P.-A. Absil[†], K. A. Gallivan[‡*]

[†]ICTEAM Institute, UCLouvain, Belgium
[‡]Florida State University, U.S.A.

December 2, 2019

Abstract

Low-rank matrix completion is the problem of recovering the missing entries of a data matrix by using the assumption that the true matrix admits a good low-rank approximation. Much attention has been given recently to exploiting correlations between the column/row entities to improve the matrix completion quality. In this paper, we propose preconditioned gradient descent algorithms for solving the low-rank matrix completion problem with graph Laplacian-based regularizers. Experiments on synthetic data show that our approach achieves significant speedup compared to an existing method based on alternating minimization. Experimental results on real world data also show that our methods provide low-rank solutions of similar quality in comparable or less time than the state-of-the-art method.

Keywords: matrix completion; graph information; Riemannian optimization; low-rank optimization
MSC: 90C90, 53B21, 15A83

1 Introduction

Low-rank matrix completion arises in applications such as recommender systems, forecasting and imputation of data; see [38] for a recent survey. Given a data matrix with missing entries, the objective of matrix completion can be formulated as the minimization of an error function of a matrix variable with respect to the data matrix restricted to its revealed entries. In various applications, the data matrix either has a rank much lower than its dimensions or can be approximated by a low-rank matrix. As a consequence, restricting the rank of the matrix variable in the matrix completion objective not only corresponds to a reasonable assumption for successful recovery of the data matrix but also reduces the model complexity.

In certain situations, besides the low-rank constraint, it is useful to add a regularization term to the error function, in order to favor other structural properties related to the true data matrix. This regularization term can often be built from side information, that is, any information associated with row and column indices of the data matrix. Recent efforts in

[*]Email:{`shuyu.dong, pa.absil`}`@uclouvain.be`, `kgallivan@fsu.edu`. This work was supported by the Fonds de la Recherche Scientifique – FNRS and the Fonds Wetenschappelijk Onderzoek – Vlaanderen under EOS Project no 30468160. The first author is supported by the FNRS through a FRIA scholarship. Part of this work was performed while the third author was a visiting professor at UCLouvain, funded by the Science and Technology Sector, and with additional support by the Netherlands Organization for Scientific Research.

exploiting side information for matrix completion include inductive low-rank matrix completion [53, 23, 58] and graph-regularized matrix completion [61, 26, 59, 42, 55]. In particular, graph-regularized matrix completion involves designing the regularization term using graph Laplacian matrices that encode pairwise similarities between the row and/or column entities. In other words, the pairwise similarities between the entries of the data matrix are modeled using an undirected graph. Depending on the application, the graph information is available through the connections between the data entities or can be inferred from the data itself.

Rao et al. [42] addressed the task of graph-regularized matrix completion by a low-rank factorization problem of the following form,

$$\underset{(G,H)\in\mathbb{R}^{m\times k}\times\mathbb{R}^{n\times k}}{\text{minimize}} \frac{1}{2}\sum_{(i,j)\in\Omega}\left((GH^T)_{ij}-M^\star_{ij}\right)^2+\lambda_{\mathrm{r}}\mathrm{Tr}\left(G^T\Theta^{\mathrm{r}}G\right)+\lambda_{\mathrm{c}}\mathrm{Tr}\left(H^T\Theta^{\mathrm{c}}H\right), \tag{1}$$

where $M^\star\in\mathbb{R}^{m\times n}$ is the data matrix to be completed, k is the maximal rank of the low-rank model, Ω is the set of revealed entries and $\lambda_{\mathrm{r}}\geq 0$ and $\lambda_{\mathrm{c}}\geq 0$ are parameters. The matrices $\Theta^{\mathrm{r}}:=I_m+L^{\mathrm{r}}$ and $\Theta^{\mathrm{c}}:=I_n+L^{\mathrm{c}}$ with given graph Laplacian matrices $L^{\mathrm{r}}\in\mathbb{R}^{m\times m}$ and $L^{\mathrm{c}}\in\mathbb{R}^{n\times n}$. The graph Laplacian matrices L^{r} and L^{c} incorporate the pairwise correlations or similarities between the columns or rows of the data matrix $M^\star$. Indeed, observe that the graph Laplacian-based penalty terms in (1) in the form of $\mathrm{Tr}\left(F^TLF\right)$ can be written as

$$\mathrm{Tr}\left(F^TLF\right)=\sum_{(i,j)\in\mathcal{E}}W_{ij}\|F_{i,:}-F_{j,:}\|_2^2, \tag{2}$$

where W is the graph adjacency matrix such that $L=\mathrm{Diag}(W\mathbf{1})-W$. The right hand-side term above suggests that the graph Laplacian-based penalty term promotes low-rank solutions that show pairwise similarities according to the given graph. Rao et al. [42] related the graph-based regularization term in (1) to a generalized nuclear norm (a weighted atomic norm [8]) and then found a close connection between the matrix factorization model (1) and a convex optimization formulation involving the generalized nuclear norm of the matrix variable $X=GH^T\in\mathbb{R}^{m\times n}$. Moreover, they [42, §5] derived an error bound for the generalized nuclear-norm minimization problem, which can be smaller than that of the standard nuclear norm minimization problem if the graph Laplacian matrices are sufficiently informative with respect to the pairwise similarities between the columns/rows of $M^\star$. In this previous work, an instance (GRALS) of the alternating minimization method is developed for solving the problem (1).

In this paper, we propose to solve the graph-regularized matrix factorization problem (1) by using Riemannian gradient descent and conjugate gradient methods. Our proposed algorithms are motivated by the following consideration. Optimization methods on the matrix product space $\mathbb{R}^{m\times k}\times\mathbb{R}^{n\times k}$ for matrix factorization models have been observed to efficiently provide good quality solutions to matrix completion problems, in spite of the non-convexity due to the factorization model $(G,H)\mapsto GH^T$. Theoretical support for this observation can be found in [49, 16, 29]. Unlike alternating minimization (*e.g.* GRALS [42]), both Euclidean gradient descent and our proposed methods update the two matrix factors simultaneously, and do not require setting stopping criteria for subproblem solvers as in alternating minimization methods. Furthermore, by exploiting non-Euclidean geometries of the set of low-rank matrices in relation with the matrix product space $\mathbb{R}^{m\times k}\times\mathbb{R}^{n\times k}$, our algorithms use descent directions based on what can be seen as scaled gradients [33, 37] in the matrix product space. Moreover, as in [33], the particular structure of the objective function makes it possible to resort to exact line minimization along the descent direction. We show that the resulting gradient descent algorithms have an iteration complexity bound akin to the Euclidean gradient method, but that faster convergence behaviors are observed with these proposed methods, compared to their counterparts with respect to the Euclidean geometry.

We test the graph-regularized matrix completion model for matrix recovery tasks on both synthetic and real datasets. We compare our proposed algorithms with a state-of-the-art method (GRALS [42]), a baseline alternating minimization (AltMin) method and Euclidean gradient descent and conjugate gradient methods. We observe that the proposed methods enjoy faster or similar convergence behaviors compared to the state-of-the-art method and faster convergence behavior than the rest of the baseline methods tested. Moreover, the convergence behavior of the proposed algorithms is observed to be more robust against balancing issues that may arise with the asymmetric factorization model in (1), compared to their counterparts with respect to the Euclidean geometry. For completeness, we also compare empirically the graph-regularized matrix completion model with two low-rank matrix completion models: the matrix factorization model without regularization and the maximum-margin matrix factorization [48, 44] in terms of recovery error. On both synthetic and real datasets, when the graph Laplacian matrices are properly constructed from features of the data matrix (with missing entries), the graph-regularized matrix completion model is found to yield solutions with superior recovery qualities compared to the other two models.

A precursor of this work can be found in the short conference paper [14].

2 Related Work and Discussions

Matrix completion models. The graph-regularized matrix completion problem (1) is a generalization to the Maximum-Margin Matrix Factorization (MMMF) problem [48, 44],

$$\underset{(G,H)\in\mathbb{R}^{m\times k}\times\mathbb{R}^{n\times k}}{\text{minimize}} \frac{1}{2}\sum_{(i,j)\in\Omega}\left((GH^T)_{ij}-M^\star_{ij}\right)^2+\frac{\lambda}{2}\left(\|G\|_F^2+\|H\|_F^2\right). \tag{3}$$

The MMMF problem (3) is related to the nuclear norm-based [5, 43, 7] convex program for low-rank matrix completion [31]

$$\min_{X\in\mathbb{R}^{m\times n}} \frac{1}{2}\sum_{(i,j)\in\Omega}\left(X_{ij}-M^\star_{ij}\right)^2+\lambda\|X\|_* \tag{4}$$

via the relation

$$\|X\|_* = \min_{G,H:GH^T=X}\frac{1}{2}\left(\|G\|_F^2+\|H\|_F^2\right).$$

As shown by Hastie et al. [20], any solution to (3) is also solution to the convex program (4), provided that $k \geq \operatorname{rank}(M^\star)$. Since the MMMF problem searches for a pair of matrix factors of rank smaller than or equal to k, which is usually much smaller than the matrix dimensions (m and n), its computational cost and memory requirements are significantly reduced compared to the nuclear norm-based convex program (4).

Optimization methods in related work. To the best of our knowledge, the state-of-the-art method for graph-regularized matrix completion is the AltMin-type algorithm GRALS [42]. For an alternating minimization method (*e.g.* [42]), one must deal with Sylvester-type equations for solving the subproblem with respect to the low-rank factor G (resp. H) when a graph-based regularization term $\operatorname{Tr}\left(G^T\Theta^{\mathrm{r}}G\right)$ (resp. $\operatorname{Tr}\left(H^T\Theta^{\mathrm{c}}H\right)$) is introduced in the objective function. Take the fully observed case $\Omega = [\![m]\!]\times[\![n]\!]$ in [42] for example, the subproblem of (1) with respect to the factor H corresponds to the following Sylvester equation

$$HG^TG+\lambda_c\Theta^{\mathrm{c}}H = M^\star G. \tag{5}$$

In the matrix completion scenario ($\Omega \subsetneq [\![m]\!]\times[\![n]\!]$), solving the subproblem of (1) with respect to the factor H corresponds to solving an equation similar to (5), where the constant matrices involved in the equation depend on the positions of the revealed entries in Ω. Equivalently, the subproblem with respect to H (resp. G) can be rewritten as a linear least-squares problem (see (81) in Appendix A.8) with respect to the vectorization of the factor H^T of dimension nk (resp. vectorization of G^T of dimension mk). The Hessian operator of this least-squares problem is not block diagonal due to the fact that the original subproblem corresponds to a Sylvester-type equation. Hence the least-squares problem of each alternating minimization step for (1) cannot be decomposed into m or n separate linear systems in dimension k. GRALS [42] solves each of the two least-squares problems by using a linear conjugate gradient (CG) solver.

AltMin-type algorithms have proven to be very efficient in solving bi-convexity problems, such as matrix factorization, nonnegative matrix factorization [52, 54], dictionary learning [39, 30], low-rank matrix completion, and in particular have also been proven to converge linearly for the low-rank matrix completion problem (without regularization) [24, 18]. On the other hand, there are inconveniences with AltMin-like algorithms in practice. The parameters that control the stopping criteria of the solver for each alternating least squares problem determines the trade-off between the accuracy of the solution and the time efficiency of the AltMin method, but there is no apparent way to set them to achieve the best trade-off once and for all kinds of data. This can be seen in one of our experiments (see Figure 3). GRALS [42], as an instance of the AltMin method with well-tuned parameters and additional stopping criteria in its subproblem solvers, may suffer from significant drop in efficiency when certain properties of the data matrix change: a change simply of the "scale" of the data matrix, which can be measured by $\|M^\star\|_F$ for example, changes significantly the performance of GRALS with a fixed set of stopping-criteria parameters.

3 Notation, definitions and problem statement

For $m \in \mathbb{N}^*$, we denote the set of integers $\{1, \ldots, m\}$ by $[\![m]\!]$. An undirected graph $\mathcal{G}$, which is determined by a set $\mathcal{V}$ of nodes, a set of (undirected) edges $\mathcal{E} \subset \mathcal{V} \times \mathcal{V}$ and edge weights $W \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$, is denoted as $\mathcal{G} = (\mathcal{V}, \mathcal{E}, W)$. The graph adjacency matrix W is symmetric because $\mathcal{G}$ is undirected. In addition, we consider adjacency matrices with nonnegative coefficients:

$$W_{i_1,i_2} = W_{i_2,i_1} \begin{cases} > 0 & \text{if } (i_1, i_2) \in \mathcal{E} \\ = 0 & \text{otherwise.} \end{cases} \tag{6}$$

Throughout this paper, the graph Laplacian matrix of a graph $\mathcal{G}$, denoted by L, is defined as

$$L = \operatorname{diag}(W\mathbf{1}) - W. \tag{7}$$

Thus we denote the graph as $\mathcal{G} = (\mathcal{V}, \mathcal{E}, W)$ or $\mathcal{G} = (\mathcal{V}, \mathcal{E}, L)$ indifferently. The graph Laplacian matrix defined by (6)–(7) is positive semi-definite (*e.g.* [11, 47]). We denote by $\Lambda = \operatorname{Diag}(\lambda_1, \ldots, \lambda_{|\mathcal{V}|})$ the diagonal matrix containing the eigenvalues of L in increasing order: $0 = \lambda_1 \leq, \ldots, \leq \lambda_{|\mathcal{V}|}$. For a graph Laplacian matrix $L \in \mathbb{R}^{n\times n}$ and any matrix $F \in \mathbb{R}^{n\times k}$, the following quantity, denoted and defined as

$$\mathcal{S}_{2,L}(F) := \sqrt{\operatorname{Tr}(F^T L F)}, \tag{8}$$

is a semi-norm of F. Moreover, by extending the calculations in [11, §1.4] from a vector $f \in \mathbb{R}^n$ to $F \in \mathbb{R}^{n\times k}$, the semi-norm (8) satisfies the characterization (2). For a matrix $M \in \mathbb{R}^{m\times n}$, we model the row index set of M by a graph $\mathcal{G}^{\mathrm{r}} = (\mathcal{V}^{\mathrm{r}}, \mathcal{E}^{\mathrm{r}}, L^{\mathrm{r}})$, where $\mathcal{V}^{\mathrm{r}} = [\![m]\!]$ and the superscript r signifies that the graph encodes row-wise correlations. Similarly, the graph that models the column-wise correlations of M is denoted as $\mathcal{G}^{\mathrm{c}} = (\mathcal{V}^{\mathrm{c}}, \mathcal{E}^{\mathrm{c}}, L^{\mathrm{c}})$. For a real-valued symmetric matrix Θ, the symbols $\lambda_{\max}(\Theta)$ and $\lambda_{\min}(\Theta)$ denote the largest and smallest eigenvalues. The notation $\Theta \succeq 0$ (resp. $\Theta \succ 0$) signifies that Θ is positive semi-definite (resp. positive definite). The largest and smallest singular values of a matrix X are denoted by $\sigma_{\max}(X)$ and $\sigma_{\min}(X)$ respectively. The Euclidean inner product and norm for the product space $\mathbb{R}^{m\times k} \times \mathbb{R}^{n\times k}$, denoted as $\langle \cdot, \cdot \rangle$ and $\| \cdot \|$ respectively, are defined as

$$\begin{aligned} \langle x, y \rangle &= \operatorname{Tr}\left(G_x^T G_y\right) + \operatorname{Tr}\left(H_x^T H_y\right), && \text{(9a)} \\ \|x\| &= \sqrt{\langle x, x \rangle}, && \text{(9b)} \end{aligned}$$

for any pair of points $x = (G_x, H_x)$, $y = (G_y, H_y) \in \mathbb{R}^{m\times k} \times \mathbb{R}^{n\times k}$.

Problem statement. As stated in Section 1, the purpose of this paper is to solve (1), which we rewrite as

$$\underset{(G,H)\in\mathbb{R}^{m\times k}\times\mathbb{R}^{n\times k}}{\text{minimize}} \frac{1}{2}\|P_\Omega(GH^T - M^\star)\|_F^2 + \frac{\alpha}{2}\left(\operatorname{Tr}\left(G^T\Theta^{\mathrm{r}}G\right) + \operatorname{Tr}\left(H^T\Theta^{\mathrm{c}}H\right)\right), \tag{10}$$

where P_Ω is the projection onto the subspace of sparse matrices with nonzeros restricted to the index set Ω. The first term in (10) is an equivalent expression of the first term of (1). As for the second term, it corresponds to the other terms of (1) with a parameterization that we find more convenient: $\Theta^{\mathrm{r}} = I_m + \gamma_{\mathrm{r}} L^{\mathrm{r}}$ and $\Theta^{\mathrm{c}} = I_n + \gamma_{\mathrm{c}} L^{\mathrm{c}}$. Setting $\gamma_{\mathrm{r}} = 0$ and $\gamma_{\mathrm{c}} = 0$ turns off the graph regularization, leaving us with the MMMF model (3). Setting $\alpha = 0$ turns off all regularization terms, leading to an unregularized matrix factorization problem

$$\min_{(G,H)\in\mathbb{R}^{m\times k}\times\mathbb{R}^{n\times k}} f_\Omega(G,H) := \frac{1}{2}\|P_\Omega(GH^T - M^\star)\|_F^2. \tag{11}$$

The choice of the rank parameter k is *a priori* unknown for low-rank matrix approximation problems such as (10). Common approaches include model selection via cross-validation and rank adaptive methods [34, 60]. In this paper, we focus on the setting where k is smaller than or equal to the optimal rank.[1] Observe that (10) is guaranteed to have a solution whenever $\alpha > 0$ since the objective function is continuous and coercive.

4 Optimization on the product space $\mathbb{R}^{m\times k} \times \mathbb{R}^{n\times k}$

In this section, we introduce a Riemannian gradient descent and a Riemannian conjugate gradient method for problem (10). Since the search space $\mathbb{R}^{m\times k} \times \mathbb{R}^{n\times k}$ is just a vector space, these methods can be interpreted as preconditioned gradient methods. However, we call them "Riemannian" because the chosen preconditioners are inspired from known Riemannian metrics on the set of rank-k m-by-n matrices, as we now explain.

In the main problem model (10), the product GH^T is an $m \times n$ matrix of rank smaller than or equal to k, and such matrices form the following nonlinear matrix space

$$\mathcal{M}_{\leq k} := \left\{X \in \mathbb{R}^{m\times n},\ \operatorname{rank}(X) \leq k\right\}.$$

In particular, when the regularization parameter α in (10) reduces to 0, the model (10) can be directly identified with the following optimization problem on $\mathcal{M}_{\leq k}$ via the matrix factorization model $\mathbb{R}^{m\times k} \times \mathbb{R}^{n\times k} \mapsto \mathcal{M}_{\leq k} : (G,H) \mapsto X = GH^T$,

$$\min_{X\in\mathcal{M}_{\leq k}} \frac{1}{2}\|P_\Omega(X - M^\star)\|_F^2. \tag{12}$$

[1] In real-world applications, it usually suffices to set up a rank value that is orders of magnitude smaller than m and n in order to work with an "underestimated" rank, that is, smaller than or equal to the optimal rank.

We refer to the model (12) and (11) as the unregularized matrix completion model, in contrast to the graph-regularized model (10). A recent survey [50] provides advances on optimization methods on the low-rank matrix space $\mathcal{M}_{\leq k}$.

Geometric elements on $\mathbb{R}^{m\times k}\times\mathbb{R}^{n\times k}$. The tangent space at a point $x\in\mathbb{R}^{m\times k}\times\mathbb{R}^{n\times k}$ is the Cartesian product of the tangent spaces of the two element matrix spaces: $T_x\left(\mathbb{R}^{m\times k}\times\mathbb{R}^{n\times k}\right)\simeq\mathbb{R}^{m\times k}\times\mathbb{R}^{n\times k}$.

Given two tangent vectors $\xi,\eta\in T_x\left(\mathbb{R}^{m\times k}\times\mathbb{R}^{n\times k}\right)$ at x, we consider the following two metrics,

- The *right-invariant* metric:

$$g_x(\xi,\eta)=\mathrm{Tr}\left(\xi_G^T\eta_G\left(G^TG+\delta I_k\right)^{-1}\right)+\mathrm{Tr}\left(\xi_H^T\eta_H\left(H^TH+\delta I_k\right)^{-1}\right); \tag{13}$$

- The *preconditioned* metric:

$$g_x(\xi,\eta)=\mathrm{Tr}\left(\xi_G^T\eta_G\left(H^TH+\delta I_k\right)\right)+\mathrm{Tr}\left(\xi_H^T\eta_H\left(G^TG+\delta I_k\right)\right), \tag{14}$$

where $\delta>0$ is a constant parameter. For both (13) and (14), g_x is a well defined inner product. The condition $\delta>0$ is required for (13) and (14) to remain well defined and positive definite when G or H does not have full column rank. Note that, if we set $\delta=0$ and restrict the search space to the product space of full column-rank matrices $\mathbb{R}_*^{m\times k}\times\mathbb{R}_*^{n\times k}$, then (13) and (14) reduce to known metrics that induce metrics on $\mathcal{M}_{\leq k}$. Specifically, (13) reduces to the right-invariant metric proposed in [32, 35], and (14) reduces to a metric proposed by Mishra et al. [33], which is specially adapted to the matrix factorization loss function (11). To see this, it suffices to note that the diagonal blocks of the Hessian (see Appendix A.9) of f_Ω in (11) correspond to the following linear transformation

$$(\xi_G,\xi_H)\quad\mapsto\quad\left(P_\Omega(\xi_GH^T)H,P_\Omega(G\xi_H^T)^TG\right). \tag{15}$$

Definition 4.1. *For a point $x\in\mathbb{R}^{m\times k}\times\mathbb{R}^{n\times k}$, the gradient of f at x is the unique vector in $T_x\left(\mathbb{R}^{m\times k}\times\mathbb{R}^{n\times k}\right)$, denoted as $\mathrm{grad}f\,(x)$, such that*

$$g_x(\xi,\mathrm{grad}f\,(x))=Df(x)[\xi],\forall\xi\in T_x\left(\mathbb{R}^{m\times k}\times\mathbb{R}^{n\times k}\right). \tag{16}$$

Based on the metric (13) and Definition 4.1, the gradient $\mathrm{grad}f\,(x)$, denoted as QRight-inv hereafter, is

$$\mathrm{grad}f\,(G,H)=\left(\partial_Gf\,(G,H)\,(G^TG+\delta I_k),\partial_Hf\,(G,H)\,(H^TH+\delta I_k)\right). \tag{17}$$

Based on the metric (14) and Definition (4.1), the gradient of $\mathrm{grad}f\,(x)$, denoted as Qprecon hereafter, at x is

$$\mathrm{grad}f\,(x)=\left(\partial_Gf\,(G,H)\,(H^TH+\delta I_k)^{-1},\partial_Hf\,(G,H)\,(G^TG+\delta I_k)^{-1}\right). \tag{18}$$

4.1 Algorithms

In this subsection, we introduce our algorithms and their elements such as computation of the gradient of the objective function of (10) and stepsize selection. We consider two basic algorithms (Algorithms 4.1 and 4.2) for optimization on $\mathbb{R}^{m\times k}\times\mathbb{R}^{n\times k}$. Computational details of these algorithms are given in Appendices A.1–A.6.

Initialization. A widely used initialization method is the so-called spectral initialization (*e.g.* [28, 27, 49]) to construct the initial low-rank variable x^0. This consists of computing (U_0, S_0, V_0) by the k-SVD of the zero-filled matrix $M_0 := P_\Omega(M^\star)$ and then defining the initial point $x^0 := (G^0, H^0)$ as follows,

$$(G^0, H^0) = (U_0 S_0^{1/2}, V_0 S_0^{1/2}). \tag{19}$$

Algorithm 4.1 Riemannian Gradient Descent (RGD)

```
Input: Function $f : \mathbb{R}^{m\times k} \times \mathbb{R}^{n\times k} \mapsto \mathbb{R}$, an initial point $x^0 \in \mathbb{R}^{m\times k} \times \mathbb{R}^{n\times k}$, and tolerance
    parameter $\epsilon > 0$.
Output: $x^t$.
1: $t \leftarrow 0$.
2: Compute gradient: $\mathrm{grad} f\left(x^t\right)$.                # See QRIGHTINV (17) or QPRECON (18)
3: while $\|\mathrm{grad} f\left(x^t\right)\| > \epsilon$ do
4:     For $\eta^t = -\mathrm{grad} f\left(x^t\right)$, find stepsize $s_t$ such that $(s_t, \eta^t)$ satisfy (23) or (24)
                                                              # See Algorithm A.6 for (23)
5:     Update: $x^{t+1} = x^t + s_t \eta^t$.
6:     $t \leftarrow t + 1$.
7:     Compute gradient: $\mathrm{grad} f\left(x^t\right)$.
8: end while
```

Gradient and descent directions. First, the computation of the Euclidean gradient of the objective function of (10) at $x := (G, H) \in \mathbb{R}^{m\times k} \times \mathbb{R}^{n\times k}$ is as follows. We have

$$Df(x)[\xi] = \mathrm{Tr}\left(\xi_G^T S H + \xi_H^T S^T G\right) + \alpha\left(\mathrm{Tr}\left(\xi_G^T \Theta^{\mathrm{r}} G\right) + \mathrm{Tr}\left(\xi_H^T \Theta^{\mathrm{c}} H\right)\right),$$

where $S = P_\Omega(GH^T - M^\star)$. From the identity

$$Df(x)[\xi] = \mathrm{Tr}\left(\xi_G^T \partial_G f(x)\right) + \mathrm{Tr}\left(\xi_H^T \partial_H f(x)\right),$$

we deduce that the components of the Euclidean gradient $\nabla f(x)$ are

$$\partial_G f(x) = SH + \alpha\Theta^{\mathrm{r}}\, G, \tag{20a}$$
$$\partial_H f(x) = S^T G + \alpha\Theta^{\mathrm{c}}\, H. \tag{20b}$$

Subsequently, the computation of the Riemannian gradient $\mathrm{grad} f(x)$ with respect to the metric (13) is based on (20) and the gradient setting QRIGHTINV (17). In this case, Algorithm 4.1 and Algorithm 4.2 are later referred to as `Qrightinv RGD` and `Qrightinv RCG` respectively. Similarly, Algorithms 4.1–4.2 are later referred to as `Qprecon RGD` and `Qprecon RCG` respectively based on (20) and the gradient setting QPRECON (18). Detailed steps for these computations are given in Algorithm A.1.

In Algorithm 4.1, the descent direction at iteration t is the negative gradient: $\eta^t = -\mathrm{grad} f\left(x^t\right)$. In Algorithm 4.2, the conjugate descent direction is defined as

$$\eta^t = -\mathrm{grad} f\left(x^t\right) + \beta_t \eta^{t-1} \tag{21}$$

for $t \geq 1$, where β_t is determined by a nonlinear CG rule, such as the Fletcher-Reeves rule

$$\beta_t = \frac{g_{x^t}\left(\mathrm{grad} f\left(x^t\right), \mathrm{grad} f\left(x^t\right)\right)}{g_{x^{t-1}}\left(\mathrm{grad} f\left(x^{t-1}\right), \mathrm{grad} f\left(x^{t-1}\right)\right)}, \tag{22}$$

depending on the local geometry of $\mathbb{R}^{m\times k} \times \mathbb{R}^{n\times k}$. A full description of the computation of the RCG descent directions is given in Appendix A.1, see Algorithms A.3 and A.5.

Algorithm 4.2 Riemannian Conjugate Gradient (RCG)

```
Input: Function $f : \mathbb{R}^{m\times k} \times \mathbb{R}^{n\times k} \mapsto \mathbb{R}$, an initial point $x^0 \in \mathbb{R}^{m\times k} \times \mathbb{R}^{n\times k}$ and $\epsilon > 0$.
Output: $x^t$.
1: $t \leftarrow 0$.
2: Compute gradient: $\mathrm{grad}f\left(x^t\right), \eta^t = -\mathrm{grad}f\left(x^t\right)$.        # See QRIGHTINV (17) or QPRECON (18)
3: while $\|\mathrm{grad}f\left(x^t\right)\| > \epsilon$ do
4:     Compute the conjuage descent direction $\eta^t$ by (21).        # See Algorithm A.3
5:     Find step size $s_t$ such that $\left(s_t, \eta^t\right)$ satisfy (23) or (24).  # See Algorithm A.6 for (23)
6:     Update: $x^{t+1} = x^t + s_t\eta^t$.
7:     $t \leftarrow t + 1$.
8: end while
```

Stepsize selection. In Algorithms 4.1–4.2, a stepsize s_t must be selected at each iteration for the update step along a descent direction $\eta^t \in T_{x^t}\left(\mathbb{R}^{m\times k} \times \mathbb{R}^{n\times k}\right)$. For this purpose, a common approach is to carry out a line search procedure by using backtracking with respect to the Armijo condition,

$$f(x^t) - f\left(x^t + s\eta^t\right) \geq \sigma s g_{x^t}\left(-\mathrm{grad}f\left(x^t\right), \eta^t\right). \tag{23}$$

For an arbitrary initial step size $s_t^0 > 0$ in the Armijo line search procedure (Algorithm A.6), the number of backtracking steps is problem dependent and corresponds to the number of times when the objective function is evaluated. Therefore, finding a good guess for the initial step size is crucial to reducing the computational cost required by the line search procedure at each iteration. Recent works [57, 3] on convergence analysis of gradient-based methods on manifolds suggest that if the vector field of Riemannian gradients of f satisfies a Lipschitz-like regularity property with a numerical constant $L_f > 0$, then one can find a good initial guess by setting $s_t^0 = 1/L_f$ or even a fixed stepsize $s_t := 1/L_f$. In the specific case with the graph-regularized matrix completion model (10), we show in Lemma 5.1 that the objective function f of (10) is Lipschitz continuously differentiable: depending on a point $x^0 \in \mathbb{R}^{m\times k} \times \mathbb{R}^{n\times k}$, one can find a Lipschitz constant of f in the sublevel set $\{x : f(x) \leq f(x^0)\}$. One can estimate this Lipschitz constant by computing, as in [34],

$$\widetilde{L_f} = \frac{\|\nabla f(x) - \nabla f(y)\|}{\|x - y\|}$$

for two points $x, y \in \mathbb{R}^{m\times k} \times \mathbb{R}^{n\times k}$ randomly chosen.

Alternatively, one can estimate the stepsize s_t via line minimization (*e.g.* [33, 51]): On a point $x \in \mathbb{R}^{m\times k} \times \mathbb{R}^{n\times k}$, we compute the stepsize defined as follows,

$$s_t^* = \underset{s\geq 0}{\operatorname{argmin}}\, f(G + s\eta_G, H + s\eta_H), \tag{24}$$

for a given *descent* direction $\eta \in T_x\left(\mathbb{R}^{m\times k} \times \mathbb{R}^{n\times k}\right)$. We choose to use the stepsize (24) for the numerical experiments in this paper. The solution s_t^* to (24) is obtained by selecting the minimizer from the real positive roots of the derivative of the quartic function of (24), which is a polynomial of degree 3 and can be computed easily. The computational cost of this procedure is of the same order as the computation of the Riemannian gradient (17) or (18). Computational details of (24) are given in Appendix A.4. In Section 5, we will show convergence properties of Algorithm 4.1 using the stepsize (24).

4.2 The regularization parameters

In this section, we discuss how to choose suitable values for the parameters of problem (10). Note that the model (10) with $\alpha > 0$ and at least one strictly positive value for γ_{r} and γ_{c} is referred to as a graph-regularized matrix completion (GRMC) model. When $\alpha > 0$ and $(\gamma_{\mathrm{r}}, \gamma_{\mathrm{c}}) = \mathbf{0}$, model (10) reduces to a MMMF model (3). When $(\alpha, \gamma_{\mathrm{r}}, \gamma_{\mathrm{c}}) = \mathbf{0}$, model (10) reduces to the unregularized matrix completion model (11), which is referred to as the MC model hereafter. Depending on the properties of the data (synthetic and real datasets), and for the graph Laplacian matrices $L^{\mathrm{r}}, L^{\mathrm{c}}$ given, we have two types of regularization schemes: (i) fixed parameter values and (ii) two-phase regularization scheme.

In the fixed-parameter scheme, we choose the parameter values following a standard cross validation procedure (see, *e.g.*, [25, §5.1.3]). We first generate a collection of parameter settings with random samples drawn from a range of values in $I_\alpha \times I_{\gamma_{\mathrm{r}}} \times I_{\gamma_{\mathrm{c}}} \subset \mathbb{R}^3_+$ with the uniform distribution in log scale, and then we select the parameter setting that has the best mean validation score in terms of RMSE (48).

Algorithm 4.3 Two-phase matrix completion using graph-based regularization (2-phase GRMC)

```
Input: Regularization parameter $\alpha > 0$. Number of iterations $T, S > 0$ for the two phases.
    An initial point $x^0 \in \mathbb{R}^{m\times k} \times \mathbb{R}^{n\times k}$. A tolerance parameter (for Phase 2) $\epsilon > 0$.
Output: $x^*, X^*$.
1: Initialize with $x^0 \in \mathbb{R}^{m\times k} \times \mathbb{R}^{n\times k}$ using (19).
2: $t \leftarrow 0$.
3: Phase 1: $x^*_\alpha$ = GRMC Solver($f_\alpha, x^0, \infty$) for at most $T$ iterations. # See Algorithms A.4 or A.5.
4: Phase 2: start from $x^*_\alpha$ and find $x^*$ = GRMC solver($f_\Omega, x^*_\alpha, \epsilon$) for at most $S$ iterations.
```

Apart from setting a fixed value to the parameter α in (10), we also consider a multi-phase regularization scheme by solving problems in the form of (10) with a decreasing series of values for α. In particular, we consider a two-phase regularization scheme, shown in Algorithm 4.3. In line 3 and line 4, the `GRMC Solver` is chosen from one of our proposed algorithms, such as `Qprecon RGD` (Algorithm A.4 with the gradient setting QPRECON (18)). In Phase 1, f_α denotes the objective function of (10) with the parameter value α. In Phase 2, the regularization term is disabled, with the parameter α of (10) set to zero, and the underlying objective function reduces to f_Ω in (11).

5 Convergence analysis

In this section, we analyze the convergence properties of Algorithm 4.1 with step sizes selected by line minimization (24). We conduct the analysis as follows: First, we show that the objective function of (10) is Lipschitz continuously differentiable in the search space $\mathbb{R}^{m\times k} \times \mathbb{R}^{n\times k}$ with respect to the Euclidean geometry. Second, we show that the specially designed non-Euclidean gradient descent directions, defined as QRIGHTINV (17) and QPRECON (18), ensures sufficient decrease in the function value, provided that the step sizes are chosen properly depending on the local geometry at each iterate. At the same time, we show that the line minimization approach (24) is a valid method for finding such step sizes. Based on these results, we are able to show the convergence behavior of the proposed RGD algorithm based on a generic convergence result given by Boumal et al. [3]. We assume throughout this section that $\alpha > 0$ in (10).

It is shown in previous work [49] that the objective function of matrix factorization for matrix completion is Lipschitz continuously differentiable. This property is preserved for the objective function of (10) since the quadratic forms in the regularizer also satisfies this regularity property. In the following lemma, we show the Lipschitz-continuity of the gradient of f in the sublevel set with respect to a point $x^0 \in \mathbb{R}^{m\times k} \times \mathbb{R}^{n\times k}$:

$$\mathcal{S}^0 = \left\{ x \in \mathbb{R}^{m\times k} \times \mathbb{R}^{n\times k}, f(x) \le f(x^0) \right\}. \tag{25}$$

Note that the inner product $\langle , \rangle$ and the norm $\|\cdot\|$ used below are defined in (9) with respect to the Euclidean geometry on $\mathbb{R}^{m\times k} \times \mathbb{R}^{n\times k}$.

Lemma 5.1 (Lipschitz-continuous gradient)**.** *For a given point $x^0 \in \mathbb{R}^{m\times k} \times \mathbb{R}^{n\times k}$, there exists a Lipschitz constant $L_0 > 0$ such that the Euclidean gradient of f in $\mathbb{R}^{m\times k} \times \mathbb{R}^{n\times k}$ is L_0-Lipschitz continuous for any $x, y \in \mathbb{R}^{m\times k} \times \mathbb{R}^{n\times k}$ belonging to the sublevel set $\mathcal{S}^0$ (25),*

$$f(y) - f(x) \le \langle \nabla f(x), y - x \rangle + \frac{L_0}{2}\|y - x\|^2. \tag{26}$$

Proof. The objective function of (10) is coercive since $\alpha > 0$. Hence $\mathcal{S}^0$ is bounded. Let B be a closed ball that contains $\mathcal{S}^0$. Since f is C^∞, it follows that it is Lipschitz continuously differentiable in B. The claim follows by a classical argument (see for example [36, Lemma 1.2.3]). □

Based on Lemma 5.1, we get the following sufficient decrease property.

Lemma 5.2. *At any iterate $x^t = (G^t, H^t)$ produced by Algorithm 4.1 before termination, the following sufficient decrease property holds, provided that the step size s satisfies $0 < s < 2\Sigma_t/L_0$, for a strictly positive value $\Sigma_t > 0$,*

$$f(x^{t+1}) - f(x^t) \le -C_t(s)\|\mathrm{grad} f\left(x^t\right)\|^2, \tag{27}$$

where $C_t(s) = s(\Sigma_t - \frac{L_0 s}{2}) > 0$ and Σ_t is defined as follows:

$$\Sigma_t = \min\left(\frac{1}{\delta + \sigma^2_{\max}(G^t)}, \frac{1}{\delta + \sigma^2_{\max}(H^t)} \right), \tag{28}$$

under the gradient setting QRIGHTINV (17) *and*

$$\Sigma_t = \delta + \min\left(\sigma^2_{\min}(G^t), \sigma^2_{\min}(H^t)\right), \tag{29}$$

under the gradient setting QPRECON (18).

Proof. In Algorithm 4.1, at iteration $x^t \in \mathbb{R}^{m\times k} \times \mathbb{R}^{n\times k}$, the Riemannian gradient descent step is $\eta^t = -\mathrm{grad} f\left(x^t\right)$. Let $s > 0$ denote the step size for producing the next iterate: $x^{t+1} = x^t + s\eta^t$. In the gradient setting QPRECON where $\mathrm{grad} f\left(x\right)$ is defined by (18), the partial differentials are

$$\partial_G f(x) = \eta_G(H^T H + \delta I_k) \text{ and } \partial_H f(x) = \eta_H(G^T G + \delta I_k). \tag{30}$$

From Lemma 5.1, we have

$$\begin{aligned} f(x^{t+1}) - f(x^t) &\le \langle \nabla f(x^t), x^{t+1} - x^t \rangle + \frac{L_0}{2}\|x^{t+1} - x^t\|^2, && (31)\\ &\le -s\|\eta^t\|^2\left(\delta + \min\left(\sigma^2_{\min}(G^t), \sigma^2_{\min}(H^t)\right)\right) + \frac{L_0 s^2}{2}\|\eta^t\|^2, && (32)\\ &= -C_t(s)\|\mathrm{grad} f\left(x^t\right)\|^2, && (33) \end{aligned}$$

where $C_t(s) = s(\Sigma_t - \frac{L_0 s}{2})$ and $\Sigma_t = \delta + \min\left(\sigma^2_{\min}(G^t), \sigma^2_{\min}(H^t)\right)$. The inequality (32) is obtained by using (30) as follows,

$$\begin{aligned}\langle \nabla f(x^t), x^{t+1} - x^t \rangle &= -s \langle \nabla f(x^t), \text{grad} f\left(x^t\right) \rangle & (34)\\ &= -s\left(\text{Tr}\left(\eta_G^T \eta_G (H^T H + \delta I_k)\right) + \text{Tr}\left(\eta_H^T \eta_H (G^T G + \delta I_k)\right)\right) & (35)\\ &\leq -s\left(\delta \|\eta^t\|^2 + \sigma^2_{\min}(H)\|\eta_G\|_F^2 + \sigma^2_{\min}(G)\|\eta_H\|_F^2\right), & (36)\end{aligned}$$

where the superscript of the element matrices $(G, H) = x^t$ and $(\eta_G, \eta_H) = \eta^t$ are omitted for brevity.

Similarly, the same result applies to the gradient setting Qprecon (17), with the quantity Σ_t determined by (28). ☐

Next, we prove that Algorithm 4.1 with step sizes selected by line minimization (24) ensures sufficient decrease at each iteration.

Lemma 5.3. *The iterates produced by Algorithm 4.1, with step sizes selected by line minimization* (24)*, satisfy the following sufficient decrease property,*

$$f(x^{t+1}) - f(x^t) \leq -\left(\Sigma_t^2 / 2L_0\right) \|\textit{grad} f\left(x^t\right)\|^2. \tag{37}$$

Proof. In Algorithm 4.1, let $\eta = -\text{grad} f\left(x^t\right)$ denote the Riemannian gradient descent direction at iteration $x^t \in \mathbb{R}^{m\times k} \times \mathbb{R}^{n\times k}$. From Lemma 5.1 and Lemma 5.2, we have

$$f(x^t + s\eta) \leq f(x^t) - C_t(s)\|\text{grad} f\left(x^t\right)\|^2,$$

for $s \in [0, 2\Sigma_t/L_0]$, with Σ_t defined in (29) and (28). One the other hand, let $\bar{s}$ be the stepsize determined by (24), then by definition, the next iterate $x^{t+1} = x^t + \bar{s}\eta$ is the minimum of f along the direction η: $f(x^{t+1}) \leq f(x^t + s\eta)$, for all $s \geq 0$. Hence

$$\begin{aligned} f(x^{t+1}) &\leq \min_{s\in[0,2\Sigma_t/L_0]} f(x^t + s\eta) & (38)\\ &\leq \min_{s\in[0,2\Sigma_t/L_0]} f(x^t) - C_t(s)\|\text{grad} f\left(x^t\right)\|^2 & (39)\\ &= f(x^t) - \left(\Sigma_t^2/2L_0\right)\|\text{grad} f\left(x^t\right)\|^2. & (40)\end{aligned}$$

☐

In both Lemma 5.2 and Lemma 5.3, the sufficient decrease quantity depends on the local information Σ_t. The quantity Σ_t is useful only when it is a strictly positive number. We address this point in Proposition 5.4 for two gradient settings Qrightinv (17) and Qprecon (18).

Proposition 5.4. *Under the same settings as in Lemma 5.2 and 5.3, there exist positive numerical constants $\Sigma_* > 0$ such that the quantities Σ_t* (28) *and* (29) *are lower-bounded,*

$$\inf_{t\geq 0} \Sigma_t \geq \Sigma_*. \tag{41}$$

Proof. In the case of gradient setting Qrightinv (17),

$$\Sigma_t = \min\left(\frac{1}{\delta + \sigma^2_{\max}(G^t)}, \frac{1}{\delta + \sigma^2_{\max}(H^t)}\right).$$

First, we prove that there exists $D_0 > 0$ such that the norm of the iterate in $\mathbb{R}^{m\times k} \times \mathbb{R}^{n\times k}$ is bounded:

$$\|x^t\| \leq D_0 \tag{42}$$

for all $t \geq 0$. It suffices to note that the whole sequence $(x^t)_{t\geq 0}$ belongs to the sublevel set $\mathcal{S}^0$ (25) and that f is coercive. Second, for any $x = (G, H) \in \mathbb{R}^{m\times k} \times \mathbb{R}^{n\times k}$, the maximal singular values $\sigma_{\max}(G)$ and $\sigma_{\max}(H^t)$ are bounded by the norm as follows,

$$\|x^t\|^2 = \operatorname{Tr}\left(G^T G\right) + \operatorname{Tr}\left(H^T H\right) \geq \sigma^2_{\max}(G) + \sigma^2_{\max}(H). \tag{43}$$

The result (41) can be deduced by taking the numerical constant $\Sigma_* := 1/(\delta + D_0^2)$ and combining (42) and (43).

In the case of gradient setting QPRECON (18),

$$\Sigma_t = \delta + \min\left(\sigma^2_{\min}(G^t), \sigma^2_{\min}(H^t)\right) \geq \delta > 0,$$

and the result (41) can be ensured by $\Sigma_* := \delta$. □

We are now ready to conclude in a similar way as in [3, Theorem 2.5]. A minor difference, however, is that the norm in [3] is the Riemannian norm, whereas our search space is a vector space and we use the Euclidean norm (9b). It is possible to use the Riemannian norms induced by (13) and (14), but it suffices here to use the Euclidean norm in the development of our results.

Theorem 5.5. *Under the problem statement* (10)*, for a given initial point* x^0 *and the gradient settings* QRIGHTINV (17) *and* QPRECON (18)*, the sequence generated by Algorithm 4.1 with the stepsize* (24) *(Algorithm A.4) converges and the decay of the gradient norm satisfies*

$$\|\mathit{grad}f\left(x^N\right)\| \leq \sqrt{\frac{2L_0(f(x^0) - f^\star)}{\Sigma_* N}} \tag{44}$$

after N iterations, where $L_0 > 0$ is the Lipschitz constant mentioned in Lemma 5.1, the numerical constant $\Sigma_* > 0$ is given in Proposition 5.4 and $f^\star$ is a lower bound[2] *of the function value of* (10).

Proof. The convergence of the sequence $(x^t)_{t\geq 0}$ is a direct result of the sufficient decrease property (27) in Lemma 5.2 and the boundedness of the sequence of function values $(f(x^t))_{t\geq 0}$. See Theorem 2.5 of [3].

Let $N \geq 1$ denote the number of iterations needed for getting to an iterate x^N such that $\|\mathrm{grad}f\left(x^N\right)\| \leq \epsilon$, for a tolerance parameter $\epsilon > 0$.

Since our algorithm (Algorithm 4.1) does not terminate at $t \leq N-1$, the gradient norms $\|\mathrm{grad}f\left(x^t\right)\| > \epsilon$, for all $t \leq N-1$. By summing the right hand sides of (37) for $t = 0, \ldots, N-1$, we have

$$\begin{aligned} f(x^N) - f(x^0) &\leq -\sum_{t=0}^{N-1} (\Sigma_t^2/2L_0)\|\mathrm{grad}f\left(x^t\right)\|^2 && (45) \\ &\leq -(\epsilon^2/2L_0)\sum_{t=0}^{N-1} \Sigma_t^2 && (46) \\ &= -\left(\Sigma_*/2L_0\right)\epsilon^2 N && (47) \end{aligned}$$

Hence the number of iterations

$$N \leq \frac{2L_0(f(x^0) - f(x^N))}{\Sigma_*\epsilon^2} \leq \frac{2L_0(f(x^0) - f^\star)}{\Sigma_*\epsilon^2}.$$

[2]One can take $f^\star := 0$ since the objective function of (10) is nonnegative.

In other words, the iterate produced by the algorithm after N iterations satisfies

$$\|\mathrm{grad} f\left(x^N\right)\| \leq \sqrt{\frac{2L_0(f(x^0) - f^\star)}{\Sigma_* N}}.$$

□

6 Numerical Experiments

In this section, we evaluate the performance of the proposed algorithms for solving the graph-regularized matrix completion problem (10). The experimental tests are based on both synthetic and real-world datasets. The synthetic data are generated using a low-rank matrix model with graph information, and the graph Laplacian matrices underlying this graph-related data model are then used in the regularization term of (10). This synthetic experimental setting corresponds to an ideal situation where the graph Laplacian matrices in the regularization term are perfectly conform with the pairwise similarities between the matrix entries. In the tests on real-world data, a graph Laplacian matrix is constructed using basic graph proximity models based on pairwise distances between the rows (or columns) of an initial estimation of the data matrix.

On both synthetic and real-world data, we compare the time efficiency of our proposed methods with several baseline methods: Euclidean gradient descent on the product space $\mathbb{R}^{m\times k} \times \mathbb{R}^{n\times k}$, alternating minimization and a state-of-the-art method GRALS [42] (also an alternating minimization method). Note that by *time efficiency*, we mean the amount of time that an iterative method takes to arrive at an iterate of a certain recovery accuracy, and this mainly depends on the convergence behavior and the computational cost per-iteration of the method. We also compare the graph-regularized (GRMC) model (10) with two other matrix completion models in terms of matrix recovery errors: (i) unregularized matrix completion (MC) via the factorization model (11) and (ii) the maximum-margin matrix factorization (MMMF) model (3). The following list gives detailed description of the methods involved in the experiments.

- `Qprecon RGD` (Algorithm A.4) corresponds to Riemannian gradient descent (Algorithm 4.1) using the gradient setting QPRECON (18); `Qprecon RCG` (Algorithm A.5) corresponds to Riemannian conjugate gradient descent (Algorithm 4.2) using the gradient setting QPRECON (18). Similarly, `Qrightinv RGD` and `Qrightinv RCG` correspond to Algorithm 4.1 and Algorithm 4.2 using the gradient setting QRIGHTINV (17). The step sizes in all these algorithms are selected via line minimization (24). Since we focus on the application of (10) in the low rank setting, where the rank parameter k is underestimated, we maintain this setting in all experiments. In relation to this setting, we set the parameter δ in the definition of QPRECON (18) and QRIGHTINV (17) to zero without any numerical issue. We show in Figure 10 (Appendix A.5) that the convergence behavior of the so-tested algorithms is almost the same as their counterparts in the theoretical setting (with a presumably small $\delta > 0$) analyzed in Section 5.

- For consistency, `Euclidean GD` and `Euclidean CG` (see Appendix A.7) stand for the gradient descent and nonlinear conjugate gradient descent algorithms respectively, in which the descent directions (such as the Euclidean gradient) are computed with respect to the Euclidean geometry on $\mathbb{R}^{m\times k} \times \mathbb{R}^{n\times k}$. The step sizes in all these algorithms are selected via line minimization (24).

- `AltMin`: An alternating minimization algorithm (Algorithm A.9), where each of the two graph-regularized least-squares subproblems is solved by using the linear CG method

Algorithm A.12. The method labeled as `AltMin1` is an instance of `AltMin` whose accuracy parameter $\epsilon = 10^{-14}$ and $n_{\mathrm{CG}}^{\max} = 500$. The method labeled as `AltMin2` is an instance of `AltMin` whose accuracy parameter $\epsilon = 10^{-6}$ and $n_{\mathrm{CG}}^{\max} = 500$. Note that the parameter $n_{\mathrm{CG}}^{\max}$ can also be set to even larger values: Since each of the two subproblems in the alternating minimization are initialized with the latest iterate (warm-started), the number of iterations required for each subproblem solver to obtain a solution with an accuracy ϵ usually does not exceed $n_{\mathrm{CG}}^{\max}$ preset here. Hence, the active parameter for controlling the stopping behavior of the subproblem solvers is ϵ in the experiments.

- `GRALS`: An alternating minimization algorithm implemented by Rao et al. [42] and is available on line[3]. In particular, `GRALS1` denotes the GRALS algorithm with the accuracy parameter $\epsilon = 10^{-10}$ and $n_{\mathrm{CG}}^{\max} = 500$. GRALS differs from `AltMin` in that the linear CG solver for the subproblems (79)–(80) has an additional stopping criterion[4] compared to `AltMin` (Algorithm A.9 and A.12), which could trigger early stop hence provide inexact solutions to the subproblems (79)–(80) under certain circumstances.[5]

All numerical experiments are performed on a workstation with 8-core Intel Core i7-4790 CPUs and 32GB of memory running Ubuntu 16.04 and MATLAB R2015a. The source code is available on `https://gitlab.com/shuyudong.x11/grmc`. To assess the approximation performance for the matrix completion task, we use the root mean-squared-error (RMSE). Suppose $M^\star \in \mathbb{R}^{m\times n}$ is known on $\Omega \subset [\![m]\!] \times [\![n]\!]$. Then the RMSE of $X \in \mathbb{R}^{m\times n}$ with respect to $M^\star$ is defined as

$$\mathrm{RMSE}\,(X;\Omega) = \sqrt{\sum_{(i,j)\in\Omega} (X_{ij} - M^\star_{ij})^2/|\Omega|}. \tag{48}$$

In particular, the training error and test error is measured by the following quantities,

$$\mathrm{RMSE\ (Train)} := \mathrm{RMSE}\,(X;\Omega_{\mathrm{train}})\,, \tag{49}$$

$$\mathrm{RMSE\ (Test)} := \mathrm{RMSE}\,(X;\Omega_{\mathrm{test}})\,, \tag{50}$$

where the index set of training entries Ω_{train} is Ω in (10) and Ω_{test} is the complementary of Ω_{train}.

6.1 Synthetic Data

We generate synthetic data with the following low-rank matrix model, which can be seen as a generalization of the model in [42, §5.1] . Let $\mathcal{G}^{\mathrm{r}} := (\mathcal{V}^{\mathrm{r}}, \mathcal{E}^{\mathrm{r}}, L^{\mathrm{r}})$ and $\mathcal{G}^{\mathrm{c}} := (\mathcal{V}^{\mathrm{c}}, \mathcal{E}^{\mathrm{c}}, L^{\mathrm{c}})$ be the graphs modeling the row-wise and column-wise similarities of $M^\star$ and let $(U^{\mathrm{r}}, \Lambda^{\mathrm{r}})$ (resp. $(U^{\mathrm{c}}, \Lambda^{\mathrm{c}})$) denote the pair of matrices containing the eigenvectors and associated eigenvalues of L^{r} (resp. L^{c}). Then a low-rank data matrix $X^\star$ is generated as follows,

$$Z^\star = F^\star Q^{\star T}, \tag{51a}$$

$$X^\star = A^{\mathrm{r}} Z^\star (A^{\mathrm{c}})^T, \tag{51b}$$

where $(F^\star, Q^\star) \in \mathbb{R}^{m\times r^\star} \times \mathbb{R}^{n\times r^\star}$ are composed of columns that are *i.i.d.* Gaussian vectors and the matrices $A^{\mathrm{r}} \in \mathbb{R}^{m\times m}$ and $A^{\mathrm{c}} \in \mathbb{R}^{n\times n}$ are defined below with respect to a function $g : \mathbb{R} \mapsto \mathbb{R}$ acting element-wisely on a diagonal matrix:

$$A^{\mathrm{r}} = U^{\mathrm{r}} g(\Lambda^{\mathrm{r}}), A^{\mathrm{c}} = U^{\mathrm{c}} g(\Lambda^{\mathrm{c}}). \tag{52}$$

[3]Link: `https://github.com/rofuyu/exp-grmf-nips15`.

[4]A stopping criterion that restricts the subproblem update to a region of radius depending on the norm of the partial gradients of f.

[5]This feature is one of reasons that make the convergence behavior of GRALS different from that of `AltMin`, and in several applications, faster than the latter.

More precisely, $g(\Lambda) = \mathrm{Diag}\left(g(\lambda_1, \cdots, g(\lambda_m)\right)$ for any diagonal matrix Λ. The function g in (52) enables one to control the way in which the graph information in L^{r} and L^{c} transforms the low-rank random Gaussian matrix $Z^\star$. In the literature of graph signal processing [45], the function g is referred to as a graph spectral filter, which is a graph analogue of filters in signal processing. In our experiments, we set the function g as

$$g(\lambda) = \begin{cases} \lambda^{-p} & \text{if } \lambda > 0, \\ 0 & \lambda = 0, \end{cases} \tag{53}$$

for $p \geq 1$. The spectral model (53) is a typical example of functions that are monotonically non-increasing over $\mathbb{R}_+^*$, which have the effect of low-pass filters [45] in the graph spectral domain [46]. Other examples include (i) the Tikhonov filter (*e.g.* [1]) $g_\gamma(\lambda) = 1/\sqrt{1+\gamma\lambda}$, and (ii) the diffusion operator [12, 13, 56] $g_\tau(\lambda) = e^{-\tau\lambda}$.

Remark 6.1. In order for model (51) to cover the graph-agnostic setting as a special case, we define by convention that $A^{\mathrm{r}} = I_m$ when $L^{\mathrm{r}} = \mathbf{0}$. The same convention applies to A^{c}.

The model (51) is of particular interest for experiments on synthetic data because it models a wide range of real data matrices whose entries present pairwise similarities: Due to the non-increasing nature of the function g on $(0, \infty)$ in (52), the transformations in (51b) with the matrices A^{r} and A^{c} return a data matrix $X^\star$ such that its graph-based semi-norm (8) is reduced compared to that before the transformation (see Appendix B.1 for detailed explanation). Hence the variations of the entries of $X^\star$ in (51b) with respect to the graph structure, measured by (2), are reduced compared to $Z^\star$ in (51a). This translates to the observation that the entries of $X^\star$ present pairwise similarities that agree with the graph $(\mathcal{V}^{\mathrm{r}}, L^{\mathrm{r}})$ and/or $(\mathcal{V}^{\mathrm{c}}, L^{\mathrm{c}})$, unlike the structureless entries in $Z^\star$ (51a). Figure 1 shows the difference between $Z^\star$ and $X^\star := A^{\mathrm{r}} Z^\star$ regarding this property.

6.1.1 Matrix completion from noiseless observations

In this subsection, the ground-truth data matrix $M^\star$ is generated by (51) for $r^\star \ll \min(m, n)$ and is partially observed without any noise. The index of the revealed entries are *i.i.d.* sampled according to the Bernoulli model

$$(i, j) \in \Omega \text{ with probability } \rho, \text{ for any } (i, j) \in [\![m]\!] \times [\![n]\!]. \tag{54}$$

For simplicity, we let $L^{\mathrm{c}} = 0$ such that $A^{\mathrm{c}} = I_n$ (see Remark 6.1). Hence the graph information is incorporated in $M^\star$ row-wisely by A^{r} with respect to (52). For this purpose, a graph Laplacian matrix L^{r} is generated with the prototypical graph model `Community` using the GSPbox [40]. The function g in this model is (53) with $p = 2$.

For the matrix completion model (10), we set the rank parameter by $k := \mathrm{rank}(M^\star)$. Note that in this case, $M^\star$ belongs to $\mathcal{M}_{\leq k}$, and any point $(G^*, H^*) \in \mathbb{R}^{m \times k} \times \mathbb{R}^{n \times k}$ such that $G^* H^{*T} = M^\star$ exactly recovers the hidden matrix $M^\star$. Hereafter, we refer to the search of such a point (G^*, H^*) as exact recovery of the data matrix. In the literature of matrix completion, exact recovery of a low-rank matrix $M^\star$ by a factorization model such as (11) is possible under conditions on the extent of incoherence [6] of the singular subspaces of $M^\star$ and the observation model Ω. Specifically, several sample complexity lower-bounds for $\rho \approx |\Omega|/mn$ are proved with both regularized (*e.g.* [49, 16]) and unregularized (implicitly regularized) [29] matrix factorization models.

In the experiments of this subsection, we carry out tests for recovering the hidden matrix $M^\star$ with our proposed two-phase (2-phase GRMC) regularization scheme (Algorithm 4.3). The following algorithms are chosen and tested as the solver in Phases 1 and 2 (line 3

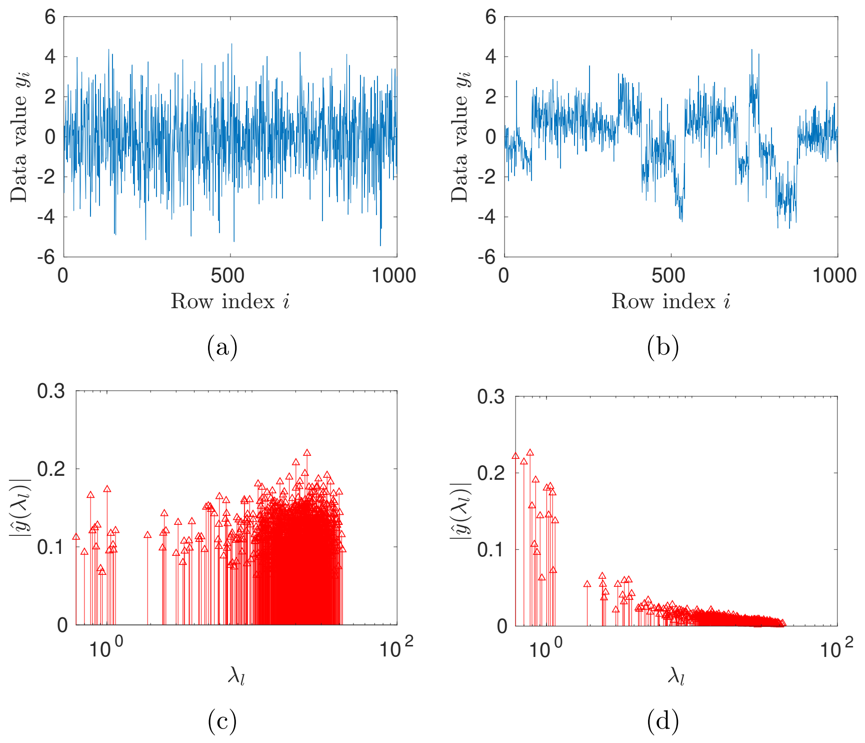
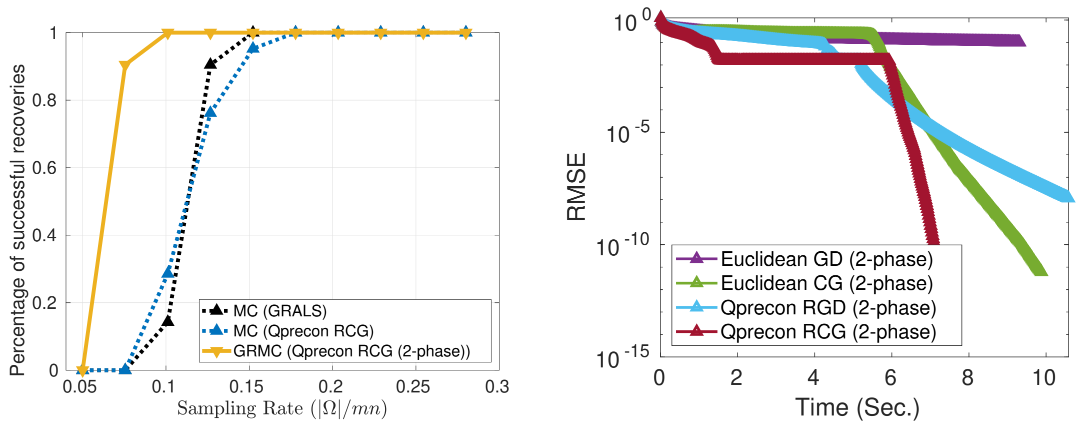


Figure 1: A data matrix Z from the Gaussian random model (51a) and $X = A^r Z$ from the graph-based model (51b). The Laplacian matrix L^{r} involved in (51b) is generated with the prototypical graph model `Community` using GSPbox [40]. Comparison in the data entries and in the graph spectral domain. Top: A randomly chosen column (a): $y = Z_{(j)}$ and (b): $y = X_{(j)}$. Bottom: Average amplitude of graph Fourier coefficients (c): $\tilde{\mathbb{E}}|\widehat{Z_{(j)}}(\lambda_l)|$ and (d): $\tilde{\mathbb{E}}|\widehat{X_{(j)}}(\lambda_l)|$ from low (small eigenvalue $\lambda_l(L^{\mathrm{r}})$) to high graph vertex frequencies.

and line 4 of Algorithm 4.3): `Qprecon RGD`, `Qprecon RCG` and baseline Euclidean geometry-based algorithms (`Euclidean GD` and `Euclidean CG`). The (one-phase) unregularized matrix completion (MC) model (11), which corresponds to the special setting of (10) for $(\alpha, \gamma^{\mathrm{r}}, \gamma^{\mathrm{c}}) = \mathbf{0}$, is also tested.

First, we compare empirically the sample complexities of (i) the unregularized matrix completion model (MC) and (ii) the graph-regularized matrix completion model (10) through the 2-phase GRMC scheme described above. Under the experimental settings described in the beginning of this Section, for $m = 500$, $n = 600$ and $r^\star = 12$, we carry out repeated tests at various sampling rates $|\Omega|/mn$ ranging from 5% to 28%. At each sampling rate, we compute the percentage of successful recoveries among $N_{\text{tests}} = 20$ repeated tests. Each test is counted as successful if the RMSE on test entries (49) of the solution is smaller than 10^{-12}. In particular, at each sampling rate, the parameters $(\alpha, \gamma_{\mathrm{r}})$ in the GRMC model (10) (for Phase 1 of Algorithm 4.3) are selected among a collection of randomly generated values, see descriptions of the fixed-parameter scheme in Section 4.2 for details. As shown in the 2-phase GRMC scheme (Algorithm 4.3), the whole algorithm is stopped by either the accuracy parameter ϵ (see Algorithms 4.1, 4.2), when the iterate becomes an ϵ-stationary point or by the iteration budget parameter S, which is tuned to a sufficiently large value for both successful and unsuccessful recovery scenarios. Experimental results are shown in Figure 2(a): These results show empirically that the 2-phase GRMC method has a lower sample complexity than unregularized matrix factorization.

Second, we compare the time efficiency of the proposed algorithms with their counterparts



(a) (b)

Figure 2: Matrix completion from noiseless observations. $M^\star$ is generated with non-trivial graph information with the model (53) and is partially observed without any noise. The rank parameter $k := \operatorname{rank}(M^\star)$. (a): Percentage of successful recoveries under various sampling rates. The solutions are given by the two different matrix completion models (MC and GRMC): matrix size $m = 500$, $n = 600$, rank $r^\star = 12$. The sampling rate $|\Omega|/mn$ ranges from 5.0% to 28.0%. (b): Results per iteration by 2-phase GRMC (Algorithm 4.3) at sampling rate $|\Omega|/mn = 10.0\%$: matrix size $m = 800$, $n = 900$, rank $r^\star = 12$.

under the Euclidean geometry. Figure 2(b) shows results per iteration under a sampling rate that is sufficiently large for 2-phase GRMC.

In particular, when the sampling rate ρ is sufficiently large, it is possible to exactly recover the hidden matrix $M^\star$ without any regularization (see Figure 2(a) at sampling rates larger than 15%). Therefore, we test our algorithms for this special case, with the problem parameter α set to zero in (10). In this special regime, we compare the time efficiency of our proposed methods with the several other methods in two different settings for the initialization point $x^0 \in \mathbb{R}^{m\times k} \times \mathbb{R}^{n\times k}$: In the first test, each method is initialized at a point $x^0 = (G_0, H_0)$ given by (19). In this case, the two factors G_0 and H_0 are *balanced*, in the sense that their matrix norms are equal. In the second test, we test the same methods with an unbalanced initial point

$$y^0 = (\lambda G_0, H_0/\lambda), \tag{55}$$

for $\lambda = 5$. The comparative results are given in Figure 3.

From the results in Figure 2 and Figure 3, we have the following observations:

- Our algorithms (`Qprecon RGD, RCG`) are faster than their Euclidean geometry-based counterparts (`Euclidean GD, CG`) in every experimental setting.
- Our algorithms are faster than the baseline alternating minimization methods `AltMin1, AltMin2`.
- `Qprecon RCG` is faster than `GRALS1` and this comparison becomes much more evident when the initialization point is unbalanced than when it is balanced. Similarly, `Qprecon RGD` is as fast as `GRALS1` in the balanced initialization setting and much faster than the latter in the unbalanced case.
- In relation to the remark above, the baseline methods `Euclidean GD`, `Euclidean CG` and `GRALS1` are significantly slower when the initialization point is unbalanced.

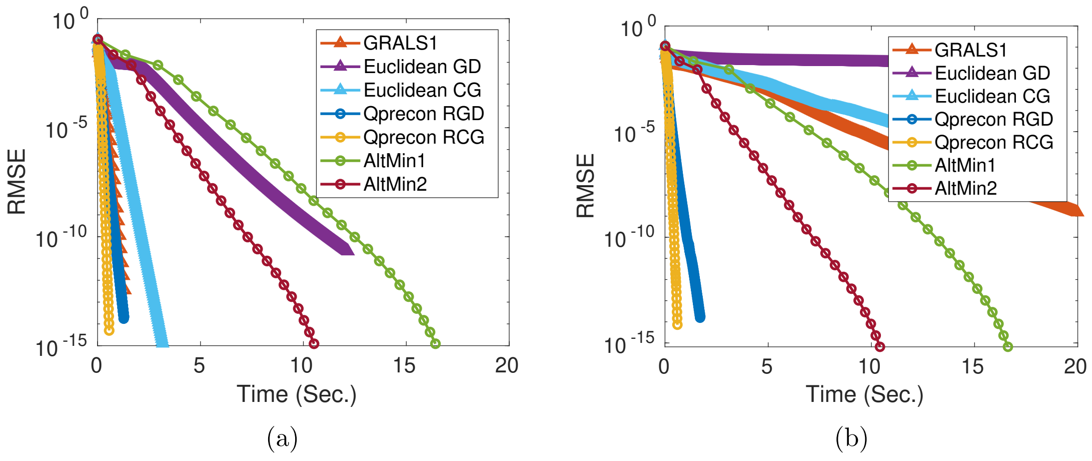


Figure 3: Results per iteration. Experimental settings: $m = 1000, n = 900, r = r^\star = 10, |\Omega|/mn = 20.0\%$. $M^\star$ is generated with the model (53) and is partially observed without any noise. (a): Each method is initialized at x^0 by (19). (b): Each method is initialized at y^0 by (55).

6.1.2 Matrix completion from noisy observations

In this subsection, we assume that the partially observed data matrix $M^\star$ is composed of noisy observations from a low-rank matrix $X^\star$,

$$M^\star = X^\star + E, \tag{56}$$

where $E_{ij} \overset{\text{i.i.d.}}{\sim} \mathcal{N}(0, \sigma_N^2)$ for all $(i,j) \in [\![m]\!] \times [\![n]\!]$ and $X^\star \in \mathbb{R}^{m\times n}$ is generated using the model (51) with rank $r^\star \ll \min(m,n)$. For simplicity, we let $L^{\mathrm{c}} = 0$ and only incorporate row-wise similarities in $X^\star$ through $L^{\mathrm{r}} \in \mathbb{R}^{m\times m}$ with respect to (52). For this purpose, a graph Laplacian matrix L^{r} is generated with the prototypical graph model `Community` using the GSPbox [40]. The function g in this model is (53) with $p = 2$. Figure 4 shows the singular values of $M^\star$ generated with (56), where the true low-rank matrix $X^\star$ is generated with (51), and the noise level of E is determined by a signal-to-ratio parameter SNR $= 20$.

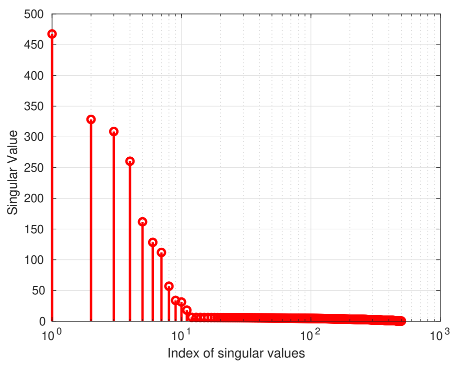


Figure 4: Singular values of $M^\star$ generated with (51) and (56), where the noise level is set by a signal-to-ratio parameter SNR $= 20$.

For the matrix completion problem (10), the rank parameter k is set to be smaller than rank$(X^\star)$ and its values will be specified later. We test our algorithms, the baseline algorithms and the state-of-the-art algorithm GRALS [42] for solving the problem (10) with fixed parameter values $\alpha \geq 0$ and $\gamma_{\mathrm{r}} \geq 0$.

First, we compare the recovery quality of the solutions to the three types of matrix completion models, that is, the unregularized matrix completion (MC), the graph-regularized matrix completion (GRMC) model (10) and the maximum-margin matrix completion (MMMF) in (3). For the unregularized problem model (MC), the parameter setting is $\alpha = 0$. For the MMMF model ($\alpha > 0$ and $\gamma_{\mathrm{r}} = 0$), the parameter setting is selected among a collection of N_{HP} randomly generated values in a reasonable range, $(\alpha^i)_{i=1,\dots,N_{\mathrm{HP}}}$. For the GRMC model ($\alpha > 0$ and $\gamma_{\mathrm{r}} > 0$), the parameter setting is selected among a collection of N_{HP} randomly generated values in a reasonable range, $(\alpha^i, \gamma_{\mathrm{r}}^i)_{i=1,\dots,N_{\mathrm{HP}}}$. The criterion for the parameter selection procedure is the recovery score of the solution in terms of the RMSE on test entries (49). At each sampling rate and once the parameters are selected for MMMF and GRMC, the recovery score of each of the three matrix completion models, using a given algorithm (ours as well as the baseline methods) corresponds to the average score after N_{tests} training instances based on $(M^\star, \Omega_s)_{s=1,\dots,N_{\mathrm{tests}}}$, where the observation sets Ω_s are generated according to (54) with the given (fixed) sampling rate. All methods tested are stopped by either the accuracy parameter ϵ (see Algorithms 4.1, 4.2), when the iterate becomes an ϵ-stationary point or by an iteration budget parameter, which is set to a sufficiently large value for all methods. Figure 5 shows the recovery scores of each of the three problem models under different sampling rates. From Figure 5, we can see that at various sampling rates, the GRMC model (10) provide solutions with superior recovery qualities than the other two graph-agnostic models. Naturally enough, the improvement on recovery qualities via GRMC is significant at small sampling rates.

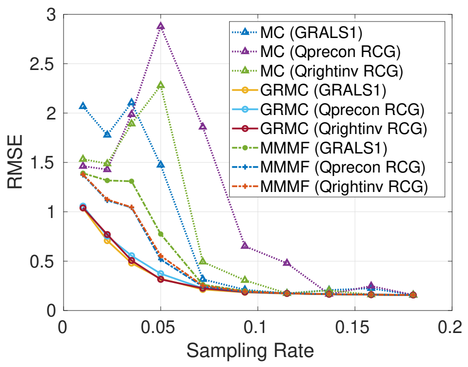
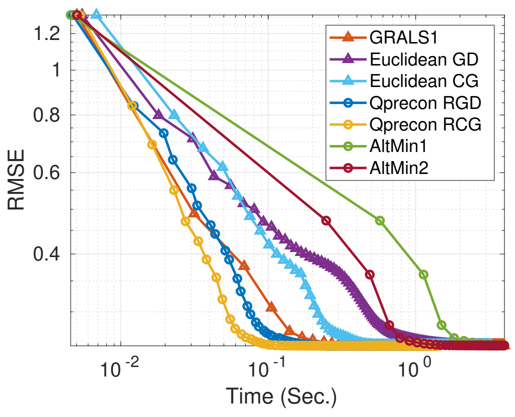


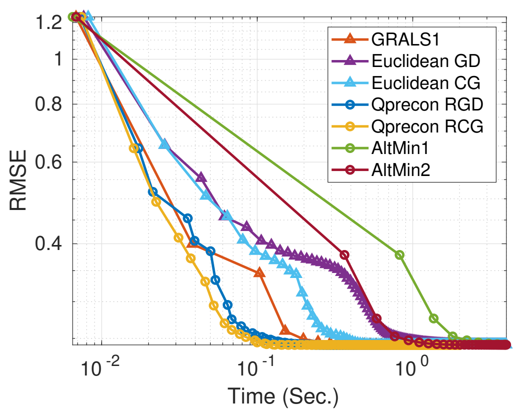
Figure 5: Matrix completion from noisy observations. $M^\star$ is generated with non-trivial graph information with the model (53) and is partially observed with additive noise (SNR = 20). The rank parameter $k = 10$. Recovery score of solutions to MC, MMMF and GRMC at various sampling rates. Matrix size: $m = 500$, $n = 600$, rank $r^\star = 12$. The sampling rate $|\Omega|/mn$ ranges from 1.0% to 18.0%.

Second, we compare the time efficiency of the proposed algorithms with the baseline methods. The methods are tested in two slightly different experimental settings. Based on the data generation method described in the beginning of this subsection, the data matrix in each of these two experiments is rescaled with respect to a given constant value: We set the constant scalar such that $\mathbb{E}\left[|M^\star_{ij}|\right] = 1$ in the first setting and $\mathbb{E}\left[|M^\star_{ij}|\right] = 10^{-3}$ in the second one. Figure 6 shows the time efficiency of the tested methods in these two experiments in terms of the RMSE score per iteration. Note that the tests for producing Figure 5 are conducted under the data setting $\mathbb{E}\left[|M^\star_{ij}|\right] = 1$, without loss of generality.

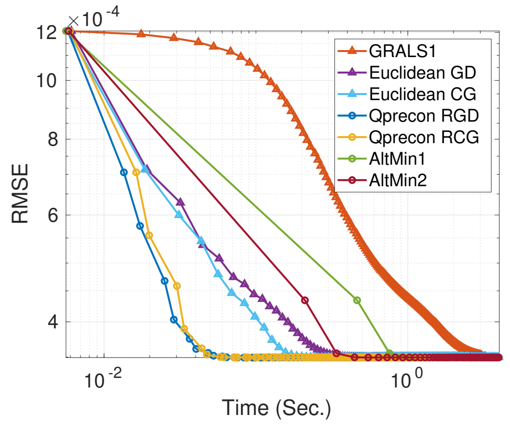
In particular, for the second data setting, with $\mathbb{E}\left[|M^\star_{ij}|\right] = 10^{-3}$, we show in Figure 7 the recovery qualities of the iterates under the unregularized (MC) and the graph-regularized



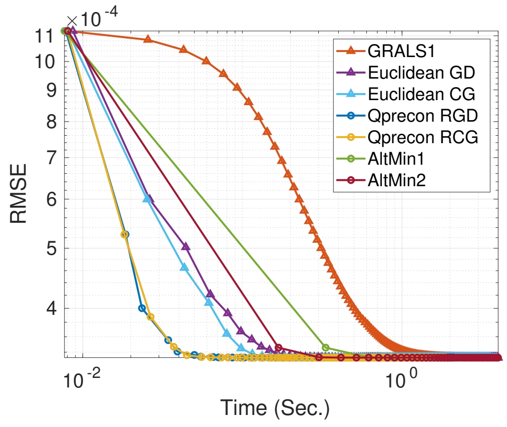
(a)



(b)



(c)



(d)

Figure 6: Results per iteration on sythetic data. The data matrix $M^\star$ is generated via (51) and (56). Matrix size: $m = 1000$,$n = 900$, rank $r^\star = 12$. The rank parameter $k = 8$. Subplots (a-b): The data matrix $M^\star$ is rescaled by a scalar constant such that $\mathbb{E}\left[|M^\star_{ij}|\right] = 1$; (a): the sampling rate $|\Omega|/mn = 11.5\%$, (b): $|\Omega|/mn = 18.0\%$. Subplots (c-d): The data matrix $M^\star$ is rescaled by a scalar constant such that $\mathbb{E}\left[|M^\star_{ij}|\right] = 10^{-3}$; (c): the sampling rate $|\Omega|/mn = 11.5\%$, (d): $|\Omega|/mn = 18.0\%$.

(GRMC) models, for a relatively low sampling rate. Given the graph L^{r} underlying the synthetic data model (51), the GRMC model corresponds to one randomly generated set of parameters ($\alpha > 0, \gamma_{\mathrm{r}} > 0$), where α is randomly generated in the range $(10^{-6}, 10^{-3})$ and γ_{r} randomly generated in the range $(10^{-2}, 5)$. We can see that for all the tested methods, the recovery qualities of the iterates under the GRMC model outperforms those of the unregularized matrix completion model.

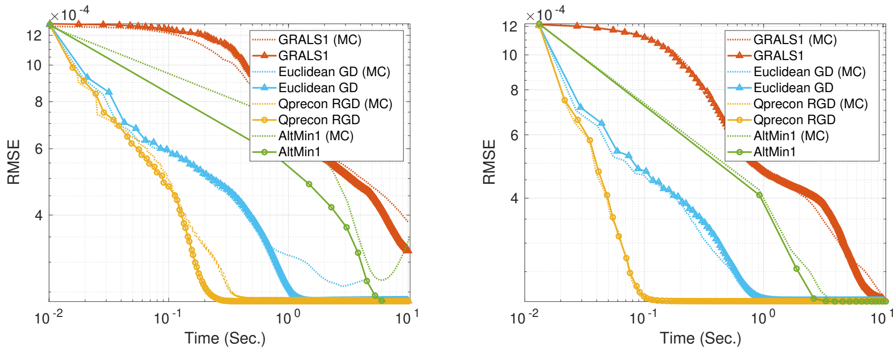


Figure 7: Results per iteration on synthetic data. The data matrix $M^\star$ is rescaled by a scalar constant such that $\mathbb{E}\left[|M^\star_{ij}|\right] = 10^{-3}$. Matrix size: $m = 1000$, $n = 900$, rank $r^\star = 12$. The rank parameter $k = 8$. The sampling rate $|\Omega|/mn$ is 3.5%.

From the results in Figure 6 and Figure 7, we have the following observations about the time efficiency for recovering the hidden data matrix:

- Our algorithms (`Qprecon RGD, RCG`) are faster than their counterparts under the Euclidean geometry (`Euclidean GD, CG`).
- Our algorithms are faster than the baseline alternating minimization methods `AltMin1, AltMin2`.
- `Qprecon RCG` is either faster than `GRALS1` or as fast as the latter in various settings.
- In relation to the previous remark. The time efficiency of GRALS changes significantly when there is a simple change in the scale of the data matrix, as shown in Figure 6, since it has an additional stopping criterion that restricts the search of the solution to the least-squares subproblem (79) (resp. (80)) to a region of radius $\|\partial_G f(G^{t-1}, H^{t-1})\|$ (resp. $\|\partial_H f(G^t, H^t)\|$). This restricted-region criterion depends however, both on properties of the data matrix (such as the scale of the data) and the iterate.

6.2 Real Data

In this subsection, we conduct experiments on real-world datasets. An essential difference between these experiments and experiments on synthetic data is that there is no reference graph associated with the data matrices in a real-world application. Since the data matrix in a real-world application often present pairwise similarities between its entries, we build the graphs L^{r} and L^{c} based on the given data before the graph-regularized matrix completion task. Subsequently, we conduct tests with the graph-regularized and graph-agnostic matrix completion models using all the methods involved, and compare their time efficiency. The real-world data used for these tests are from the PeMS Traffic occupancy and MovieLens datasets.

6.2.1 Methodology for graph construction

In the existing work on matrix completion using graph-based regularization, there are two main approaches to constructing the graph Laplacian matrices: (i) build the graph Laplacian matrix from the data $M^\star \in \mathbb{R}^{m\times n}$ itself by using a certain graph node proximity model (*e.g.* [26]), and (ii) build a similarity graph L^{r} (and/or L^{c}) from side information [42, 55], that is, information related to the entities of the row (and/or column) indices of $M^\star$.

In our experiments, we adopt the first approach. Note that in [26], the computation of the graph proximity parameters is based on pairwise distances using only the revealed entries in $M^\star$. In contrast, we compute the graph proximity parameters based on a low-rank approximation of the partially revealed matrix. More precisely, we propose to use a rank-r approximation of M_0 as the features for constructing the graph. Let (U_0, S_0, V_0) denote the r-SVD of the zero-filled matrix $M_0 := P_\Omega(M^\star) \in \mathbb{R}^{m\times n}$ and let $\widetilde{M_0} := U_0 S_0 V_0^T$. Next, the computation of the graph edge weight parameters based on the given matrix $M := \widetilde{M_0}$ can be realized by using various node proximity methods such as K-Nearest Neighbors (K-NN) and ε-graph models [9, 2, 21, 10], which boils down to computing a certain distance matrix between the rows (resp. columns) of M. Let $Z^{\mathrm{r}}(M) \in \mathbb{R}^{m\times m}$ denote the row-wise distance matrix of M defined as follows,

$$Z_{ij}(M) = d\left(M_{i,:}, M_{j,:}\right), \text{ for } i, j \in [\![m]\!], \tag{57}$$

where $d : \mathbb{R}^n \times \mathbb{R}^n \mapsto \mathbb{R}_+$ is a distance on the n-dimensional vector space. Subsequently, we build a Gaussian ε-graph by computing the node proximity weights as follows

$$[W_\varepsilon(M)]_{ij} = \exp\left(-Z_{ij}(M)/\varepsilon^2\right), \text{ for } i, j \in [\![m]\!], \tag{58}$$

where $\varepsilon \in \mathbb{R}$ is a hyperparameter of the graph model. Furthermore, a sparse graph adjacency matrix is more preferable than a dense in a computational point of view, as the per-iteration cost for computing the gradient (as well as the function value) in (20) depends partly on $\mathrm{nnz}(L^{\mathrm{r}})$ and $\mathrm{nnz}(L^{\mathrm{c}})$, hence the sparsity of the row-wise (resp. column-wise) graphs. For simplicity, we sparsify the graph adjacency matrix defined in (58) by the following thresholding operation

$$[W_{\varepsilon,\sigma}(M)]_{ij} = \mathbf{1}_{\geq\sigma}\left(\exp\left(-Z_{ij}(M)/\varepsilon^2\right)\right), \text{ for } i, j \in [\![m]\!], \tag{59}$$

where $\mathbf{1}_{\geq\sigma}$ is the hard threshold function $\mathbf{1}_{\geq\sigma}(z) = \begin{cases} z & \text{if } z \geq \sigma \\ 0 & \text{otherwise.} \end{cases}$

In the graph model (59), the parameter ε is tuned according to the variance of $(Z_{ij})_{i,j=1,..,m}$ and σ is chosen according to a preset sparsity level $\mathfrak{s} \ll 1$ for the edge set associated with $W_{\epsilon,\sigma}$ such that $|\mathcal{E}(W_{\epsilon,\sigma})|/m^2 \leq \mathfrak{s}$.

6.2.2 The Traffic Data

The PeMS *Traffic* occupancy data[6] is a matrix with dimensions $963 \times 10\,560$ containing traffic occupancy rates (between 0 and 1) recorded across time by $m = 963$ sensors placed along different car lanes of the San Francisco Bay area freeways. The recordings are sampled every 10 minutes covering a period of 15 months. The column index set corresponds to the time domain and the row index set corresponds to geographical points (sensors), which are referred to as the spatial domain. Unlike the case with data from social networks or any other kind with useful meta-data, there is no straightforward way to find any side information for the *Traffic* dataset that may help constructing a spatial-domain graph. Hence we construct a sparse row-wise similarity graph with the Gaussian ε-graph model (59).

[6]https://archive.ics.uci.edu/ml/datasets/PEMS-SF

Based on the same methodology for parameter selection (for the GRMC and MMMF models) as described in Section 6.1.2, we compare the matrix recovery qualities of GRMC and the other two graph-agnostic matrix completion models. The results are shown in Table 1.

	SR=1%			SR=5%			SR=20%		
Method	MC	MMMF	GRMC	MC	MMMF	GRMC	MC	MMMF	GRMC
GRALS1	0.0453	0.0351	0.0343	0.0778	0.0291	0.0272	0.0371	0.0246	0.0232
AltMin1	0.1218	0.0356	0.0344	0.0455	0.0332	0.0280	0.0317	0.0249	0.0244
AltMin2	0.1217	0.0352	0.0343	0.0455	0.0308	0.0275	0.0317	0.0247	0.0238
Euclidean GD	0.0455	0.0352	0.0343	0.0399	0.0299	0.0276	0.0261	0.0253	0.0241
Euclidean CG	0.0472	0.0350	0.0343	0.1443	0.0289	0.0272	0.0360	0.0246	0.0232
Qprecon RGD	0.0553	0.0351	0.0344	0.0477	0.0293	0.0273	0.0297	0.0246	0.0232
Qprecon RCG	0.0514	0.0350	0.0343	0.1875	0.0291	0.0271	0.0570	0.0246	0.0232
Qrightinv RGD	0.0454	0.0452	0.0349	0.0329	0.0326	0.0326	0.0300	0.0299	0.0300
Qrightinv RCG	0.0454	0.0452	0.0343	0.0812	0.0295	0.0273	0.0261	0.0254	0.0241

Table 1: Recovery scores (RMSE on test entries) of solutions to the three types of matrix completion models: the unregularized matrix completion (MC), maximum-margin matrix factorization (MMMF) and graph-regularized matrix completion (GRMC).

Figure 8 shows the time efficiency of the methods tested in terms of the RMSE score per iteration.

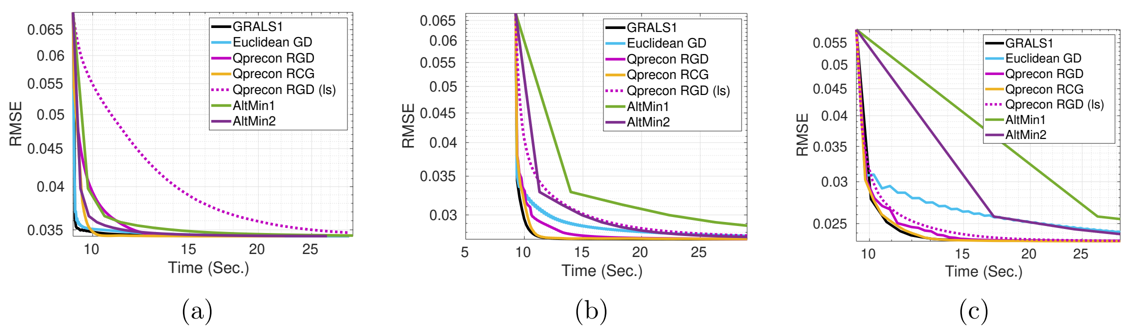


Figure 8: Results per iteration on the *Traffic* dataset ($m = 963$, $n = 10560$). The rank chosen is for the model (10) is $k = 18$. (a): the sampling rate $|\Omega|/mn = 1.0\%$, (b): $|\Omega|/mn = 5.0\%$, (c): $|\Omega|/mn = 20.0\%$. In particular, the label `(ls)` of the dashed line refers to the method using backtracking linesearch, see Algorithm A.7.

6.2.3 MovieLens dataset

The *MovieLens 100K*[7] dataset [19] consists of 10 000 ratings (1 to 5) from 943 users on 1 682 movies. Each user has rated at least 20 movies. The data was collected through the MovieLens web site (`movielens.umn.edu`) during the seven-month period from September 19th, 1997 through April 22nd, 1998. This data has been cleaned up—users who had less than 20 ratings or did not have complete demographic information were removed from this data set. For the graph-regularized matrix completion model, we construct a sparse row-wise similarity graph (on the set of users) with the Gaussian ε-graph model (59).

Based on the same methodology for parameter selection (for the GRMC and MMMF models) as described in Section 6.1.2, we compare the matrix recovery qualities of GRMC

[7]https://grouplens.org/datasets/movielens/100k/

and the other two graph-agnostic matrix completion models. The results are shown in Table 2. The RMSE scores of the GRMC model, returned by the methods tested using the selected parameter setting, are around 0.957, which is close to the RMSE score of 0.945 given by the graph-regularized method in [42] and is better than the scores of all other methods reported in [42]. Note that (i) the rank value chosen in the present experiment is the same as that in [42] and (ii) the graph Laplacian matrix used by Rao et al. [42] comes from side information, while the graph Laplacian matrix in the present experiment is constructed with the sparse ε-graph model (59), and (iii) in our experiment, the training set is 80% of the data entries in the ML100k dataset, while Rao et al. [42] used 90% of the available data. To achieve even better recovery scores under the GRMC framework, one needs to refine the construction of the graph Laplacian matrix either with models that are more adapted to the features of the data matrix or using more sensible user/movie-related information.

Methods	MC	MMMF	GRMC
GRALS1	2.076	0.984	0.957
GRALS2	1.203	0.983	0.957
Euclidean CG	1.411	0.986	0.957
AltMin1	4.069	0.984	0.956
AltMin2	4.018	0.984	0.956
Qprecon RGD	1.083	0.986	0.957
Qprecon RCG	1.917	0.984	0.959

Table 2: Matrix completion score (RMSE on test entries) of solutions to the three types of problem models: unregularized matrix completion (MC), Maximum-margin matrix factorization and Graph-regularized matrix completion (GRMC).

We also compare the time efficiency of the methods tested in terms of the RMSE score per iteration. Results are shown in Figure 9.

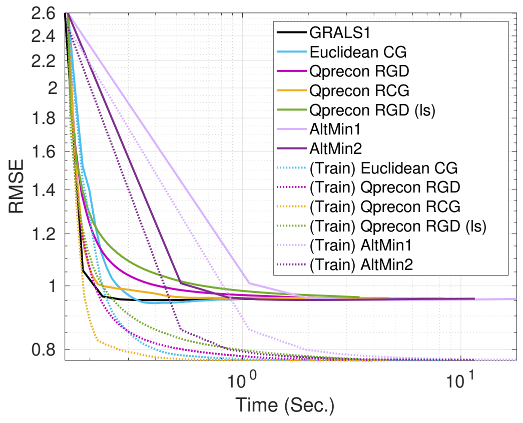


Figure 9: Results per iteration on the *MovieLens100k* dataset ($m = 943$, $n = 1682$). Rank parameter $k = 10$. The number of revealed entries is 80% of the 100k available ratings and the effective sampling rate $|\Omega|/mn \approx 5.05\%$. In particular, the label `(ls)` of the green line refers to the method using backtracking linesearch, see Algorithm A.7.

6.2.4 Discussion on real-data experiments

From both Table 1 and Table 2, we observe that the matrix recovery quality of solutions to the GRMC model (10) is superior to those of the other two graph-agnostic matrix completion models. From Figure 8 and Figure 9, we have the following observations:

- Our algorithms (`Qprecon RGD, RCG`) are faster than `Euclidean GD` and the baseline alternating minimization methods `AltMin1, AltMin2`.
- The time efficiency of `Qprecon RCG` and the state-of-the-art method `GRALS1` are similar on the two real datasets tested. Observe that both `Qprecon RCG` and `GRALS1` are considerably faster than the two `AltMin` methods, though `GRALS1` and `AltMin` are based on the same alternating minimization strategy. This can be due to the programming language (C++ for `GRALS1` and MATLAB for `AltMin`) and to GRALS's above-mentioned additional stopping criterion for the subproblem solver.
- The stepsize by line minimization (24) yields faster convergence behavior than back-tracking line search (with respect to the Armijo rule, starting from an arbitrary guess $s_0 = 1$ for the initial stepsize).

7 Conclusion

In this paper, we focused on a graph-regularized matrix factorization problem for matrix completion. We proposed efficient algorithms for the underlying optimization problem on the product space $\mathbb{R}^{m\times k} \times \mathbb{R}^{n\times k}$. Our proposed gradient descent and conjugate gradient methods on $\mathbb{R}^{m\times k} \times \mathbb{R}^{n\times k}$ are based on specially designed nonlinear metrics in this search space. Moreover, we focused on a stepsize selection method by exact line minimization, which results in a superior time efficiency compared to the approach using back-tracking line search. We provided rigorous theoretical analysis of the convergence property of the proposed Riemannian gradient descent algorithm.

We have conducted extensive experiments on synthetic data: we observed that our approach achieves significant speedup compared to several baseline methods, including a state-of-the-art method (GRALS) using alternating minimization, on various experimental settings. Moreover, we have shown via several tests that the proposed methods are much less influenced by changes in the initialization point or the scale of the data matrix. We also investigated the matrix recovery qualities of various matrix completion models, including the graph-regularized model, under various sampling rates and found that the graph-based regularization does provide improvement for the matrix recovery quality compared to graph-agnostic matrix completion models, especially for relatively low sampling rates. In our experiments on real-world data, we found that our methods produce solutions to the graph-regularized matrix completion model in comparable or less time than the baseline and the state-of-the-art methods.

Appendix A Algorithms

A.1 Computation details of Algorithms 4.1–4.2 with line minimization

Computing the Riemannian gradient. Detailed steps and their respective computational costs for computing the Riemannian gradient are given in Algorithm A.1. In the case of computing Qprecon (18): For the matrix inversion-related computations in the form of AB^{-1}, with $A := \partial_G f(x) \in \mathbb{R}^{m\times k}$ and $B := G^T G \in \mathbb{R}^{k\times k}$, a typical approach[8] is to first take (once) a Cholesky decomposition of B, whose cost is $C_{\text{chol}} k^3$, and then compute the

[8]as is used by many matrix computation environments such as Matlab's built-in syntax `A/B` for AB^{-1}, with B positive definite

Algorithm A.1 Computation of the Riemannian gradient

```
Input: $x = (G, H) \in \mathbb{R}^{m\times k} \times \mathbb{R}^{n\times k}, P_\Omega(M^\star) \in \mathbb{R}^{m\times n}, \Omega, \Theta^{\mathrm{r}} \in \mathbb{R}^{m\times m}, \Theta^{\mathrm{c}} \in \mathbb{R}^{n\times n}$, and the parameter $\alpha$.
Output: Riemannian gradient $\xi = (\xi_G, \xi_H) \in T_x(\mathbb{R}^{m\times k} \times \mathbb{R}^{n\times k})$.
1: Compute the residual $S = P_\Omega(GH^T - M^\star)$.    # $(2k+1)|\Omega|$ flops
2: Compute
       $\partial_G f(x) = SH + \alpha\Theta^{\mathrm{r}}G, \quad \partial_H f(x) = S^T G + \alpha\Theta^{\mathrm{c}}H.$
                                  # $4(|\Omega| + \mathrm{nnz}(\Theta^{\mathrm{r}}) + \mathrm{nnz}(\Theta^{\mathrm{c}}))k$ flops
3: Compute
       $\xi = (\partial_G f(x)(G^T G), \partial_H f(x)(H^T H))$ w.r.t. (17) and (20)
                                  # $4(m+n)k^2$ flops
   or compute
       $\xi = (\partial_G f(x)(H^T H)^{-1}, \partial_H f(x)(G^T G)^{-1})$ w.r.t. (18) and (20)
                                  # $4(m+n)k^2 + 2C_{\mathrm{chol}}k^3$ flops, see (60)
```

forward-and-backward substitution to get each of the m rows of AB^{-1}, which costs $2mk^2$. In brief, the flop counts of this line consists of

- Computing $G^T G$ and $H^T H$: $2(m+n)k^2$ flops,
- Computing the Cholesky decomposition of $(G^T G)$ and $(H^T H)$: $2C_{\mathrm{chol}}k^3$, where $C_{\mathrm{chol}} = 1/3$.
- Forward-and-backward substitutions: $2(m+n)k^2$,

which sum to

$$4(m+n)k^2 + 2C_{\mathrm{chol}}k^3. \tag{60}$$

The dominant term in (60) is $4(m+n)k^2$ when $k \ll \min(m,n)$, which is the case for low-rank matrix approximation problems with a small rank parameter k and large data matrices.

The total number of flops needed for Algorithm A.1 is either of the following

$$(6k+1)|\Omega| + 4\mathrm{nnz}(\Theta)\,k + 4(m+n)k^2, \tag{61a}$$
$$(6k+1)|\Omega| + 4\mathrm{nnz}(\Theta)\,k + 4(m+n)k^2 + 2C_{\mathrm{chol}}k^3, \tag{61b}$$

where (61a) is for computing Qrightinv (17) and (61b) is for computing Qprecon (18). The dominant cost in Algorithm A.1 is for the computations in lines 1 and 2, which is

$$\mathcal{O}\left((|\Omega| + \mathrm{nnz}(\Theta))k\right),$$

where we use the term $\mathrm{nnz}(\Theta) := \mathrm{nnz}(\Theta^{\mathrm{r}}) + \mathrm{nnz}(\Theta^{\mathrm{c}})$ to denote the sum on the right-hand side, for simplicity. Indeed, when $k \ll \min(m,n)$ and the sampling rate ρ is of the order of 10%, the terms $(m+n)k \le (m+n)k^2 \ll |\Omega|k = \rho mnk$.

Computing the cost function. For the algorithms with the line search procedure (Algorithm A.6), the evaluation of the cost function is needed.

Hence the cost for evaluating once the objective function (10) is

$$\mathrm{FLOPS}_{\mathrm{fobj}} = (2k+3)|\Omega| + 2\mathrm{nnz}(\Theta)\,k + 2(m+n)k. \tag{62}$$

To see the order of magnitude of the total cost: the dominant costs of Algorithm A.2 are $\mathcal{O}((|\Omega| + \mathrm{nnz}(\Theta))k)$. Hence the total cost of Algorithm A.2 is at the order of

$$\mathcal{O}\left((|\Omega| + \mathrm{nnz}(\Theta))k\right).$$

Algorithm A.2 Computation of the cost function (10)

```
Input: $x = (G, H) \in \mathbb{R}^{m\times k} \times \mathbb{R}^{n\times k}, P_\Omega(M^\star) \in \mathbb{R}^{m\times n}, \Omega, \Theta^{\mathrm{r}} \in \mathbb{R}^{m\times m}, \Theta^{\mathrm{c}} \in \mathbb{R}^{n\times n}$, and the parameter $\alpha$.
Output: Function value $f(x)$ in (10).
1: Compute the residual $S = P_\Omega\left(GH^T - M^\star\right)$.    # $(2k+1)|\Omega|$ flops
2: Compute $f_\Omega(x) := \frac{1}{2}\|S\|_F^2$,    # $2|\Omega|$ flops
3: Compute $\mathrm{Reg}(x) := \mathrm{Tr}\left(G^T\Theta^{\mathrm{r}}G\right) + \mathrm{Tr}\left(H^T\Theta^{\mathrm{c}}H\right)$,
                                   # $2\mathrm{nnz}(\Theta)\,k + 2(m+n)k$ flops
4: Return $f(x) = f_\Omega(x) + \frac{\alpha}{2}\mathrm{Reg}(x)$.
```

Computing the conjugate gradient direction. The following schemes for computing the CG step parameter β_t in the Riemannian optimization setting (21) are adapted from nonlinear conjugate gradient schemes in the classical Euclidean setting, , such as

$$\begin{aligned}
\text{Polak-Ribiere [41] (PR) } \beta &= \max\left(0, \frac{g_{x^t}\left(\xi^t - \xi^{t-1}, \xi^t\right)}{g_{x^t}\left(\xi^{t-1}, \xi^{t-1}\right)}\right), && \text{(63a)}\\
\text{Hestenes-Stiefel [22] (HS+) } \beta &= \max\left(0, \frac{g_{x^t}\left(\xi^t - \xi^{t-1}, \xi^t\right)}{g_{x^t}\left(\xi^t - \xi^{t-1}, \eta^{t-1}\right)}\right), && \text{(63b)}\\
\text{Fletcher-Reeves [15] (FR) } \beta &= \frac{g_{x^t}\left(\xi^t, \xi^t\right)}{g_{x^t}\left(\xi^{t-1}, \xi^{t-1}\right)}. && \text{(63c)}
\end{aligned}$$

A survey on nonlinear conjugate gradient can be found in [17]. Implementation of these schemes (63) can be found in the Riemannian optimization toolbox Manopt [4]. In our experiments, we choose the modified Hestenes-Stiefel (HS+) rule. The flop counts for the HS+ rule is $5(m+n)k$.

Algorithm A.3 Computation of the conjugate gradient direction

```
Input: iterates $x^{t-1}, x^t \in \mathbb{R}^{m\times k} \times \mathbb{R}^{n\times k}$, gradients $\xi^t \in T_x\left(\mathbb{R}^{m\times k} \times \mathbb{R}^{n\times k}\right)$, $\xi^{t-1} \in T_{x^{t-1}}\left(\mathbb{R}^{m\times k} \times \mathbb{R}^{n\times k}\right)$, previous CG direction $\eta^{t-1} \in T_{x^{t-1}}\left(\mathbb{R}^{m\times k} \times \mathbb{R}^{n\times k}\right)$.
Output: CG direction $\eta^t \in T_x\left(\mathbb{R}^{m\times k} \times \mathbb{R}^{n\times k}\right)$.
1: Compute the CG step parameter $\beta_t$ with one of the schemes in (63) and then the direction
       $$\eta^t = -\xi^t + \beta_t\eta^{t-1}.$$
                                   # $7(m+n)k$ flops
2: Compute the angle between the CG direction and the gradient:
       $$\theta = \langle\eta^t, \xi^t\rangle / \|\eta^t\|\|\xi^t\|.$$
                                   # $6(m+n)k$ flops
   Reset to gradient if desired: $\eta^t = \xi^t$ if $\theta < 0.1$
```

From Algorithm A.3, the total flop counts for computing once the Riemannian CG direction, given two consecutive Riemannian gradients, is

$$13(m+n)k. \tag{64}$$

A.2 Flop counts for `Qprecon/Qrightinv RGD (linemin-lsFree)` (Algorithm A.4)

Algorithm A.4 is an instance of Algorithm 4.1 with a step size given by line minimization (24).

Flop counts for computing the step size via line minimization (24). This corresponds to the computations for $c_1, .., c_4$ in (69a)–(69d), which sums to

$$(6k + 11)|\Omega| + 2\text{nnz}(\Theta)\, k + 4(m + n)k.$$

Details are in Appendix A.4.

Note that since the step size s_t is obtained by (24), which guarantees a sufficient decrease, there will be no need for any additional line search steps.

Algorithm A.4 Riemannian Gradient Descent (RGD (LINEMIN-LSFREE))

```
Input: $f : \mathbb{R}^{m\times k} \times \mathbb{R}^{n\times k} \mapsto \mathbb{R}$, an initial point $x^0 \in \mathbb{R}^{m\times k} \times \mathbb{R}^{n\times k}$ and $\epsilon > 0$.
Output: $x^t$.
1: $t \leftarrow 0$.
2: Compute gradient: $\text{grad}f\left(x^t\right)$                          # see Algorithm A.1.
3: while $\|\text{grad}f\left(x^t\right)\| > \epsilon$ do
4:     Find step size $s_t$ via (24)                                       # see (70)
5:     Update: $x^{t+1} = x^t - s_t \text{grad}f\left(x^t\right)$.        # $(m + n)k$ flops
6:     $t \leftarrow t + 1$.
7:     Compute $\text{grad}f\left(x^t\right)$                              # see (61)
8:     Compute $\|\text{grad}f\left(x^t\right)\|$                          # $2(m + n)k$ flops
9: end while
```

The total flop counts for one iteration (line 3–9) is

$$\text{FLOPS}(t) \equiv 12(k + 1)|\Omega| + 6\text{nnz}(\Theta)\, k + (m + n)(4k^2 + 7k), \tag{65}$$

for any iteration $t \geq 0$.

A.3 Flop counts for `Qprecon/Qrightinv RCG (linemin-lsFree)`

Algorithm A.5 is an instance of Algorithm 4.2 with a step size given by line minimization (24).

Algorithm A.5 Riemannian Conjugate Gradient (RCG (LINEMIN-LSFREE))

```
Input: Function $f : \mathcal{M} \mapsto \mathbb{R}$, a retraction $\|$ on $\mathcal{M}$, an initial point $x^0 \in \mathcal{M}, \eta^0 = 0$, and
    $\epsilon > 0$.
Output: $x^t$.
1: $t \leftarrow 0$.
2: Compute gradient: $\text{grad}f\left(x^0\right), \eta^0 = \text{grad}f\left(x^0\right)$.
3: while $\|\text{grad}f\left(x^t\right)\| > \epsilon$ do
4:     Find step size $s_t$ (along $\eta^t$) via (24)                      # see (70)
5:     Update: $x^{t+1} = x^t - s_t\eta^t$.                                 # $(m + n)k$ flops
6:     $t \leftarrow t + 1$.
7:     Compute $\text{grad}f\left(x^t\right)$                              # see (61)
8:     Compute CG direction $\eta^t$ based on $\eta^{t-1}, \text{grad}f(x)^t, \text{grad}f(x)^{t-1}$ via Algorithm A.3
                                                                            # see (64)
9:     Compute $\|\text{grad}f\left(x^t\right)\|$                          # $2(m + n)k$ flops
10: end while
```

`RCG linemin-lsFree` (Algorithm A.5) needs to compute the nonlinear CG direction via Algorithm A.3, and its flop counts is larger than (65) by exactly that in (64). In sum, it is

$$\text{FLOPS}(t) \equiv 12(k + 1)|\Omega| + 6\text{nnz}(\Theta)\, k + 4(m + n)(k^2 + 5k), \tag{66}$$

for any iteration $t \geq 0$.

A.4 Stepsize computation via line minimization

Computing the stepsize (24) consists in minimizing

$$f(G + s\eta_G, H + s\eta_H) - f(G, H)$$

for $s \geq 0$.

We have $f(G + s\eta_G, H + s\eta_H) - f(G, H) = A + B$ such that

$$A = \frac{1}{2}\|P_\Omega\left(s(G\eta_H^T + \eta_G H^T) + s^2\eta_G\eta_H^T\right)\|_F^2 + \\ \left\langle P_\Omega(GH^T - M), P_\Omega(s(G\eta_H^T + \eta_G H^T) + s^2\eta_G\eta_H^T)\right\rangle \quad (67)$$

and

$$B = \frac{1}{2}\mathrm{Tr}\Big[\left(sG^T L_r\eta_G + s\eta_G^T L_r G + s^2\eta_G^T L_r\eta_G\right) + \\ \left(sH^T L_c\eta_H + s\eta_H^T L_c H + s^2\eta_H^T L_c\eta_H\right)\Big]. \quad (68)$$

These two equations lead to the following quartic polynomial form $A + B = \sum_{j=1}^4 c_j s^j$, where

$$\begin{aligned}
c_1 &= \left\langle P_\Omega(GH^T - M), P_\Omega(G\eta_H^T + \eta_G H^T)\right\rangle + \mathrm{Tr}\left(\eta_G^T L_r G + \eta_H^T L_c H\right), && \text{(69a)}\\
c_2 &= \frac{1}{2}\|P_\Omega(G\eta_H^T + \eta_G H^T)\|_F^2 + \left\langle P_\Omega(GH^T - M), P_\Omega(\eta_G\eta_H^T)\right\rangle + \\
&\quad \frac{1}{2}\mathrm{Tr}\left(\eta_G^T L_r\eta_G + \eta_H^T L_c\eta_H\right), && \text{(69b)}\\
c_3 &= P_\Omega(G\eta_H^T + \eta_G H^T), P_\Omega(\eta_G\eta_H^T), && \text{(69c)}\\
c_4 &= \frac{1}{2}\|P_\Omega(\eta_G\eta_H^T)\|_F^2. && \text{(69d)}
\end{aligned}$$

The solution to s^* is selected from the real positive roots of the derivative of this quartic function, which is the polynomial of degree 3 as follows, $(A + B)'(s) = \sum_{j=1}^4 c_j s^{j-1}$, whose roots can be computed easily.

Computational costs. In Algorithms 4.1–4.2, whenever the line minimization (24) is called, it always follows the computation of a Riemannian gradient, during which we have stored the following intermediate matrices

- $S = P_\Omega(GH^T - M^\star) \in \mathbb{R}^{m\times n}$.
- $\Theta^{\mathrm{r}}G \in \mathbb{R}^{m\times k}, \Theta^{\mathrm{c}}H \in \mathbb{R}^{n\times k}$.

Hence in the following list of flop counts, the computations related to the items above need not be counted:

- For c_1 in (69a): $(4k + 3)|\Omega| + 2(m + n)k$ flops. Information stored:[9] $P_\Omega(G\eta_H^T)$ and $P_\Omega(\eta_G H^T)$.
- For c_2 in (69b): $(2k + 4)|\Omega| + 2\mathrm{nnz}(\Theta)\, k + 2(m + n)k$ flops. Information stored: $P_\Omega(\eta_G\eta_H^T)$.

[9]The information is stored only inside the current iteration.

- For c_3 in (69c): $2|\Omega|$ flops.
- For c_4 in (69d): $2|\Omega|$ flops.

These sum up to

$$(6k+11)|\Omega| + 2\mathrm{nnz}\,(\Theta)\,k + 4(m+n)k. \tag{70}$$

A.5 The constant parameter δ in the definition of gradients

In all the experiments in Section 6, we chose to use the gradients defined in (17) or (18) with a parameter $\delta = 0$. In this setting, the underlying metric (13) is not guaranteed to be positive definite and the metric (18) is not always well defined at any iterate $x \in \mathbb{R}^{m\times k} \times \mathbb{R}^{n\times k}$. The convergence analysis does not cover the case where $\delta = 0$ in (17)–(18). Nevertheless, we note that the convergence behavior of our proposed algorithms so far tested agrees with the theoretical results presented in Section 5. In fact, in all our experimental settings, the problem parameters $(\alpha, \gamma_{\mathrm{r}}, \gamma_{\mathrm{c}})$ and the largest rank value k are chosen properly, especially that the rank parameter k is set to an underestimated value. Therefore, we did not observe any singularity in all the results presented in Section 6.

Figure 10 shows that the iterations information of our proposed RGD algorithms using the gradients defined with a strictly positive δ (for δ set to 10^{-4}) and the gradients defined with $\delta = 0$ are almost the same. The experimental setting for this illustration is the same as in Section 6.1.2.

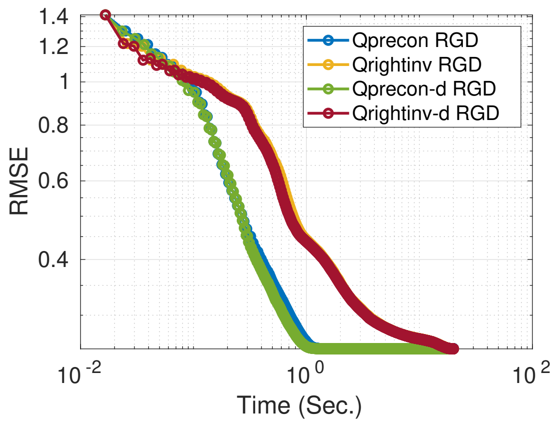


Figure 10: Results per iteration on sythetic data. The data matrix $M^\star$ is generated via (51) and (56): Matrix size $m = 1000$, $n = 900$, rank $r^\star = 12$. The rank parameter $k = 8$. The sampling rate $|\Omega|/mn = 5\%$. The results returned by the RGD algorithm using gradients defined by the two metrics with $\delta = 10^{-4}$ (labeled with "-d") and $\delta = 0$ respectively. The convergence behaviors of the algorithms with $\delta = 0$ and with a small $\delta > 0$ are almost same.

A.6 Computation details of Algorithms 4.1–4.2 with Armijo line-search

The line search procedure by backtracking with respect to the Armijo rule (23) is given in Algorithm A.6.

Algorithm A.6 Armijo line search

```
Input: $f : \mathcal{M} \mapsto \mathbb{R}$, a descent direction a retraction $\mathcal{R}$ on $\mathcal{M}$, $x^t \in \mathcal{M}$, initial stepsize $s_t^0 > 0$
    and $\sigma$, $\beta \in ]0, 1[$.
Output: $s$.
1: Initialize: $s = s_t^0$.
2: while $f(x^t) - f(\mathcal{R}_{x^t}(s\eta^t)) < \sigma s \langle -\mathrm{grad} f\left(x^t\right), \eta^t \rangle$ do
3:     $s \leftarrow \beta s$.
4: end while
```

Flop counts for `Qprecon/Qrightinv RGD (Armijo line search)` (Algorithm A.7). This corresponds to using the backtracking procedure with the Armijo rule at each iteration. We set the initial guess for each backtracking procedure by a constant $s_t^0 := 1$. Algorithm A.7 is an instance of Algorithm 4.1 with this step size selection method. Computing once the

Algorithm A.7 Riemannian Gradient Descent (RGD (lsArmijo))

```
Input: $f : \mathbb{R}^{m\times k} \times \mathbb{R}^{n\times k} \mapsto \mathbb{R}$, an initial point $x^0 \in \mathbb{R}^{m\times k} \times \mathbb{R}^{n\times k}$ and $\epsilon > 0$.
Output: $x^t$.
1: $t \leftarrow 0$.
2: Compute $(f(x^t), \mathrm{grad} f\left(x^t\right))$
3: while $\|\mathrm{grad} f\left(x^t\right)\| > \epsilon$ do
4:     Find step size $s_t$ via Algorithm A.6, for $s_t^0 = 1$.        # see (72)
5:     Update: $x^{t+1} = x^t - s_t \mathrm{grad} f\left(x^t\right)$.        # $(m+n)k$ flops
6:     $t \leftarrow t + 1$.
7:     Compute $(f(x^t), \mathrm{grad} f\left(x^t\right))$        # see (71)
8:     Compute $\|\mathrm{grad} f\left(x^t\right)\|$        # $2(m+n)k$ flops
9: end while
```

function value and the Riemannian gradient at the same time costs in total

$$\begin{aligned} & \mathrm{FLOPS}_{\mathrm{fobj}} + \mathrm{FLOPS}_{\mathrm{gradf}} - \left[\mathrm{FLOPS}(P_\Omega(GH^T)) + \mathrm{FLOPS}(\Theta^{\mathrm{r}}G, \Theta^{\mathrm{c}}H)\right] \\ = \ & (6k+4)|\Omega| + 4\mathrm{nnz}\left(\Theta\right)k + 4(m+n)k^2 + 2(m+n)k. \end{aligned} \tag{71}$$

Let n_t^{LS} denote the number of backtracking steps, then the flops required by Algorithm A.6 is

$$n_t^{\mathrm{LS}} \mathrm{FLOPS}_{\mathrm{fobj}}, \tag{72}$$

where $\mathrm{FLOPS}_{\mathrm{fobj}}$ is defined in (62). Hence, the total flop counts for one iteration (Algorithm A.7, line 3–9) is

$$(6k+4)|\Omega| + 4\mathrm{nnz}\left(\Theta\right)k + 4(m+n)k^2 + 2(m+n)k + n_t^{\mathrm{LS}} \mathrm{FLOPS}_{\mathrm{fobj}} \tag{73}$$

for an iteration $t \geq 0$.

Flop counts for `Qprecon/Qrightinv RCG (Armijo line search)`. Compared to `RGD lsArmijo` (Algorithm A.7), `RCG lsArmijo` (Algorithm 4.2, with stepsize chosen via the Armijo line-search) needs to compute the nonlinear CG direction via Algorithm A.3, and its flop counts is larger than (73) by exactly that in (64). In sum, it is

$$(6k+4)|\Omega| + 4\mathrm{nnz}\left(\Theta\right)k + 12(m+n)k^2 + 17(m+n)k + n_t^{\mathrm{LS}} \mathrm{FLOPS}_{\mathrm{fobj}}. \tag{74}$$

for an iteration $t \geq 0$.

A.7 Algorithms using the Euclidean gradient

The cost for computing the Euclidean gradient can be listed as follows. The total number

Algorithm A.8 Computation of the Euclidean gradient

```
Input: $x = (G, H) \in \mathbb{R}^{m\times k} \times \mathbb{R}^{n\times k}, P_\Omega(M^\star) \in \mathbb{R}^{m\times n}, \Omega, \Theta^{\mathrm{r}} \in \mathbb{R}^{m\times m}, \Theta^{\mathrm{c}} \in \mathbb{R}^{n\times n}$, and the parameter $\alpha$.
Output: Euclidean gradient $\nabla f(x) = (\partial_G f(x), \partial_H f(x)) \in T_x\left(\mathbb{R}^{m\times k} \times \mathbb{R}^{n\times k}\right)$.
1: Compute the residual $S = P_\Omega\left(GH^T - M^\star\right)$.        # $(2k+1)|\Omega|$ flops
2: Compute
       $\partial_G f(x) = SH + \alpha\Theta^{\mathrm{r}} G, \quad \partial_H f(x) = S^T G + \alpha\Theta^{\mathrm{c}} H.$
                                   # $4(|\Omega| + \mathrm{nnz}(\Theta^{\mathrm{r}}) + \mathrm{nnz}(\Theta^{\mathrm{c}}))k$ flops
```

of flops needed for Algorithm A.8 is the following

$$(6k+1)|\Omega| + 4\mathrm{nnz}(\Theta)\, k. \tag{75}$$

Flop counts for `Euclidean GD (linemin-lsFree)`. The total flop counts for one iteration of `Euclidean GD (linemin-lsFree)` is smaller than (65) by exactly $4(m+n)k^2$, hence

$$\mathrm{FLOPS}(t) \equiv 12(k+1)|\Omega| + 6\mathrm{nnz}(\Theta)\, k + 7(m+n)k, \tag{76}$$

for any iteration $t \geq 0$.

Flop counts for `Euclidean CG (linemin-lsFree)`. Computing the Euclidean CG step, using the same rule for computing the CG directions, requires the same cost as by (64), hence it equals

$$13(m+n)k. \tag{77}$$

As a consequence, the total flop counts for one iteration of `Euclidean CG (linemin-lsFree)` is larger than (76) by exactly that of (77)

$$\mathrm{FLOPS}(t) \equiv 12(k+1)|\Omega| + 6\mathrm{nnz}(\Theta)\, k + 20(m+n)k, \tag{78}$$

for any iteration $t \geq 0$.

A.8 Computation details in GRALS [42]

This subsection contains a description of the algorithm proposed by Rao et al. [42] and a detailed list of flop counts for the standard CG steps involved in this algorithm.

GRALS consists of two alternating least squares procedures as shown in Algorithm A.9.

The quadratic forms of the least-squares systems (79)–(80) have the following structures. The subproblem (79) is a least-squares problem whose objective $f(G) := f(G, H^t)$ can be rewritten as a quadratic form of the vectorization of G^T via the identification $f(G) = \tilde{f}_1(\mathrm{vec}(G^T))$,

$$\min_s \tilde{f}_1(s) = \frac{1}{2} s^T A^{(1)} s - \mathrm{vec}(H^{tT} P_\Omega(M^\star)^T)^T s, \tag{81}$$

where $A^{(1)} \in \mathbb{R}^{km\times km}$ has the following structure,

$$A^{(1)} = \bar{B}^{(1)} + \alpha\Theta^{\mathrm{r}} \otimes I_k, \tag{82}$$

Algorithm A.9 Alternating minimization

```
Input: Data (known on $\Omega$) $P_\Omega(M^\star) \in \mathbb{R}^{m\times n}, \Omega \subset [\![m]\!] \times [\![n]\!]$. Objective function $f$ in (10).
    Iteration budget $N$. Accuracy parameter $\epsilon \geq 0$ for inner iterations.
Output: $G^N, H^N$.
1: Initialize: $G^0 \in \mathbb{R}^{m\times k}, H^0 \in \mathbb{R}^{n\times k}$.
2: for $t = 1, .., N$ do
3:
       $$G^t = \arg\min_G f(G, H^{t-1}). \quad (79)$$
                                              # See Algorithm A.12
4:
       $$H^t = \arg\min_H f(G^t, H). \quad (80)$$
                                              # See Algorithm A.12
5: end for
```

where $\bar{B}^{(1)} \in \mathbb{R}^{km\times km}$ is block diagonal with m diagonal blocks $(B_i^{(1)})_{i=1,..,m}$ of size $k \times k$ such that

$$B_i^{(1)} = \sum_{j\in\Omega_i} h_j h_j^T, \tag{83}$$

for $i \in [\![m]\!]$. Here the index sets $\Omega_i := \{j \in [\![n]\!] : (i,j) \in \Omega\}$ and

$$h_j = [H_{j1}, .., H_{jk}]^T \in \mathbb{R}^k \tag{84}$$

is the transpose of the j-th row of H.

Similarly, the subproblem (80) is a least-squares problem whose objective $f(H) := f(G^t, H)$ can be rewritten as a quadratic form of the vectorization of H^T via the identification $f(H) = \tilde{f}_1(\mathrm{vec}(H^T))$,

$$\min_s \tilde{f}_2(s) = \frac{1}{2} s^T A^{(2)} s - \mathrm{vec}(P_\Omega(M^\star)^T G)^T s, \tag{85}$$

where $A^{(2)}$ has the following structure,

$$A^{(2)} = \bar{B}^{(2)} + \alpha\Theta^{\mathrm{c}} \otimes I_k, \tag{86}$$

where $\bar{B}^{(2)} \in \mathbb{R}^{kn\times kn}$ is block diagonal with n diagonal blocks $(B_j^{(2)})_{j=1,..,n}$ of size $k \times k$ such that

$$B_j^{(2)} = \sum_{i\in\Omega_j} g_i g_i^T, \tag{87}$$

for $j \in [\![n]\!]$. Here $\Omega_j := \{i \in [\![m]\!] : (i,j) \in \Omega\}$ and

$$g_i = [G_{i1}, .., G_{ik}]^T \in \mathbb{R}^k \tag{88}$$

is the transpose of the i-th row of G.

To solve the two subproblems in the least-squares formulations (81) and (85), Rao et al. [42] applies the linear CG algorithm. Algorithm A.12 is a standard CG descent procedure with $A \in \mathbb{R}^{q\times q}$, $b \in \mathbb{R}^q$ as inputs and x^0 as the initial point. Note that GRALS [42] uses a warm-start scheme: x^0 corresponds to latest iterate G^{t-1} (resp. H^{t-1}) for the t-th step (79) (resp. (80)).

For the quadratic forms defined in (81) and (85), the Hessian-vector multiplication (in lines 13) are computed via Algorithm A.10 and Algorithm A.11 respectively.

Algorithm A.10 Hessian-vector multiplication $A^{(1)}s$ [42]

```
Input: Data (known on $\Omega$) $P_\Omega(M^\star) \in \mathbb{R}^{m\times n}, \Omega \subset [\![m]\!] \times [\![n]\!]$. Quadratic form $A^{(1)}$ in (82).
    Vector $s := \mathrm{vec}(G^T) \in \mathbb{R}^{k\times m}$. Laplacian-based matrix $\bar{\Theta}$.
Output: $A^{(1)}s$.
1: for $i = 1,..,m$ do
2:     Get $g_i := [G_{i1}..,G_{ik}]^T$ from $s$ (vectorization of $G^T$).
3:     Compute $\tilde{g}_i = \sum_{j\in\Omega_i} h_j(h_j^T g_i)$.        # See (83).
4: end for
5: Get $G$ from the vectorization $s = \mathrm{vec}\left(G^T\right)$ and compute $\tilde{G} = \bar{\Theta}G$.
6: Return: $\mathrm{vec}([\tilde{g}_1,..,\tilde{g}_m]) + \mathrm{vec}(\tilde{G}^T)$.
```

Algorithm A.11 Hessian-vector multiplication $A^{(2)}s$ [42]

```
Input: Data (known on $\Omega$) $P_\Omega(M^\star) \in \mathbb{R}^{m\times n}, \Omega \subset [\![m]\!] \times [\![n]\!]$. Quadratic form $A^{(2)}$ in (86).
    Vector $s := \mathrm{vec}(H^T) \in \mathbb{R}^{k\times n}$. Laplacian-based matrix $\bar{\Theta}$.
Output: $A^{(2)}s$.
1: for $j = 1,..,n$ do
2:     Get $h_j := [H_{j1}..,H_{jk}]^T$ from $s$ (vectorization of $H^T$).
3:     Compute $\tilde{h}_j = \sum_{i\in\Omega_j} g_i(g_i^T h_j)$.        # See (87).
4: end for
5: Get $H$ from the vectorization $s = \mathrm{vec}\left(H^T\right)$ and compute $\tilde{H} = \bar{\Theta}H$.
6: Return: $\mathrm{vec}([\tilde{h}_1,..,\tilde{h}_n]) + \mathrm{vec}(\tilde{H}^T)$.
```

Algorithm A.12 CG Algorithm for solving (80) (resp. (79))

```
Input: $A \in \mathbb{R}^{q\times q}$, for $q = nk$ (resp. $mk$), initial point $x^0 \in \mathbb{R}^q$. Accuracy parameter $\epsilon$,
    iteration budget $n_{\mathrm{CG}}$.
Output: $x^\star \in \mathbb{R}^q$, $n_{\mathrm{CG}}^\star$
1: Compute: $b = \mathrm{vec}(P_\Omega(M^\star)^T G) \in \mathbb{R}^q$ (resp. $b = \mathrm{vec}(P_\Omega(M^\star)H)$).
                                                        # $2|\Omega|k$ flops
2: $r_0 = b - Ax^0$.
3: for $k = 0,..,n_{\mathrm{CG}}$ do
4:     Compute: $\|r_k\|$.                                 # $2nk$ (resp. $2mk$) flops
5:     if $\|r_k\| \le \epsilon\|r_0\|$ then
6:         Break;
7:     end if
8:     if $k = 0$ then
9:         $p_1 = r_0$.
10:    else
11:        $p_{k+1} = r_k + \frac{\|r_k\|^2}{\|r_{k-1}\|^2}p_k$.          # $2nk$ (resp. $2mk$) flops
12:    end if
13:    Compute: $v_{k+1} = Ap_{k+1}$.                      # $2(|\Omega| + \mathrm{nnz}\,(\Theta))k$ flops
14:    Compute: $\beta = \frac{\|r_k\|^2}{p_{k+1}^T v_{k+1}}$.                  # $2nk$ (resp. $2mk$) flops
15:    Compute: $x_{k+1} = x_k + \beta p_{k+1}$, $r_{k+1} = r_k - \beta v_{k+1}$.   # $4nk$ (resp. $4mk$) flops
16: end for
17: Return $x^\star = x^k, n_{\mathrm{CG}}^\star = k$.
```

Flop counts for Algorithm A.9 (GRALS). Now the method GRALS in Section 6 corresponds to Algorithm A.9–A.12 with $\epsilon = 0$ and the CG iteration budget $n_{\mathrm{CG}} = 20$. To count the number of flops needed for each iteration t in GRALS, we need to

- count the flops needed to compute the input $b = \mathrm{vec}(P_\Omega(M^\star)^T G) : 2|\Omega|k$. See line 1 of Algorithm A.12.
- count the flops needed for each CG iteration : Computing the Hessian-vector multiplication Ap_{k+1}, where $A = M_\Omega$ defined in (82) and (86). See line 13 of Algorithm A.12.

Hence the number of flops required by Algorithm A.12 is

$$2(n^\star_{\mathrm{CG}} + 1)|\Omega|k + 2n^\star_{\mathrm{CG}}\mathrm{nnz}\,(\Theta)\,k + 10n^\star_{\mathrm{CG}}nk$$
$$(\text{ resp. } 2(n^\star_{\mathrm{CG}} + 1)|\Omega|k + 2n^\star_{\mathrm{CG}}\mathrm{nnz}\,(\Theta)\,k + 10n^\star_{\mathrm{CG}}mk).$$

During the t-th iteration in GRALS, let n_t^H (resp. n_t^G) denote the number of CG iterations (*i.e.* $n^\star_{\mathrm{CG}}$ returned by this algorithm) required by Algorithm A.12 for solving the subproblem (80) (resp. (79)) at iteration t. Then the number of flops required by GRALS to complete the t-th iteration, from (G^t, H^t) to (G^{t+1}, H^{t+1}) as in Algorithm A.9, is

$$\mathrm{FLOPS}(t) = 2(n_t^G + n_t^H + 2)|\Omega|k + 2(n_t^G + n_t^H)\mathrm{nnz}\,(\Theta)\,k + 10(n_t^G m + n_t^H n)k, \tag{89}$$

for iteration $t \geq 0$.

Figure 11 shows the results per iteration, where the x-axis is represented either by the wall time recorded at each iteration or the cumulative cost (in Flops) required by the main computational steps in each of the algorithms at each iteration.

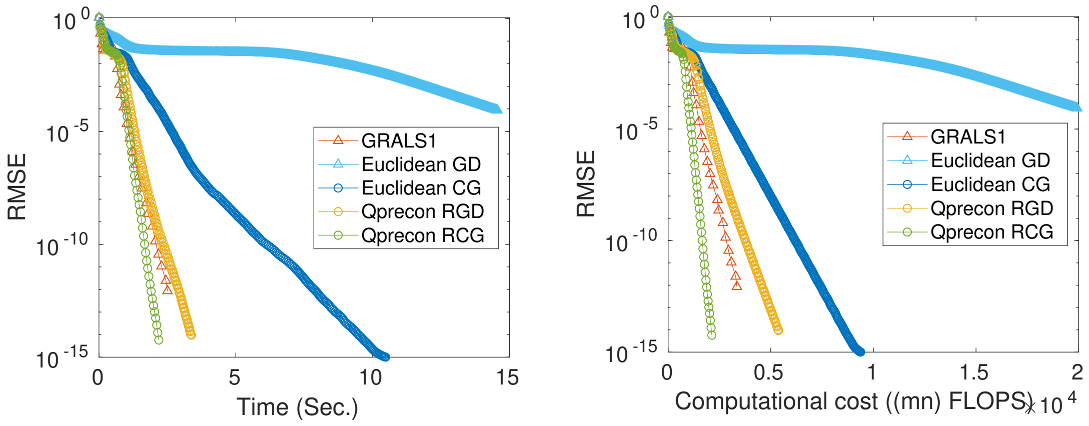


Figure 11: Results per iteration. Left: the x-axis is wall time at each iteration. Right: the x-axis is the cumulative computational cost at each iteration. Experimental settings: $m = 1000$, $n = 900$, $k = r^\star = 10$, $|\Omega|/mn = 20.0\%$. $M^\star$ is generated with the model (53) and is partially observed without any noise.

A.8.1 Our implementation (AltMin)

In addition to GRALS [42], we implement the alternating minimization method (Algorithm A.9) with a linear CG solver (for (79)–(80)) that is controlled exactly by the two following parameters,

- n_{CG}: the iteration budget for each of the two least-squares subproblems.
- ϵ: tolerance parameter to control the accuracy of the solutions to each of the two subproblems.

The most costly computation in AltMin/GRALS is the computation of $A^{(1)}\mathrm{vec}\left(G^T\right)$ and $A^{(2)}\mathrm{vec}\left(H^T\right)$. For these computations, the time-efficiency of different implementations for computing $B^{(1)}\mathrm{vec}\left(G^T\right)$ and $B^{(2)}\mathrm{vec}\left(H^T\right)$, in (83) and (87), can be significantly different, since finding the reveled entries $\Omega_i := \{j, (i,j) \in \Omega\}$ from Ω is costly. Algorithm A.13 avoids finding the indices in Ω_i for each $i \in [\![m]\!]$ by using the following incremental procedure. The notations therein are adapted to the computation of $B^{(1)}\mathrm{vec}\left(G^T\right)$. Note that for the computation of $B^{(2)}\mathrm{vec}\left(H^T\right)$, this algorithm applies by swapping the roles of G and H.

Algorithm A.13 Hessian-vector multiplication $B^{(1)}s$

```
Input: $\Omega \subset [\![m]\!] \times [\![n]\!]$. $H \in \mathbb{R}^{n\times k}$, vector $s := \mathrm{vec}(G^T) \in \mathbb{R}^{km}$.
Output: $B^{(1)}s$, for $B^{(1)}$ in (83).
1: Initialize the $k$-dimensional vectors: $y_i = \mathbf{0}$ for $i = 1, .., m$.
2: for $l = 1, .., |\Omega|$ do
3:     Get $(i_l, j_l)$: the $l$-th pair of $\Omega$.
4:     Get $h_{j_l}$ from $H$.                                              # See (84).
5:     Get $g_{i_l} = [G_{i_l 1}, .., G_{i_l k}]^T$, which is $(s_{i_l k-k+1}, .., s_{i_l k})$.   # See (88).
6:     Compute $y_{i_l} = y_{i_l} + h_{j_l}(h_{j_l}^T g_{i_l})$.
7: end for
8: Return: $\mathrm{vec}\left([y_1, .., y_m]\right) \in \mathbb{R}^{km}$.
```

A.9 The Hessian of the objective function of (10)

The second-order directional derivative of f at $x = (G, H) \in \mathbb{R}^{m\times k} \times \mathbb{R}^{n\times k}$ along a direction $\xi = (\xi_G, \xi_H) \in T_x\left(\mathbb{R}^{m\times k} \times \mathbb{R}^{n\times k}\right)$ is defined as

$$\nabla^2 f(x)[\xi] := \frac{d}{dt}\nabla f(x + t\xi)|_{t=0}. \tag{90}$$

The gradient vector field has the following expression,

$$\nabla f(x) = \left(SH + \alpha\Theta^{\mathrm{r}} G, S^T G + \alpha\Theta^{\mathrm{c}} H\right), \tag{91}$$

where $S := P_\Omega(GH^T - M)$.

To simplify notations, we calculate the two matrix components separately,

$$\begin{aligned}\frac{d}{dt}\partial_G f\left(x + t\xi\right)|_{t=0} &= \lim_{t\to 0}\frac{1}{t}\Big[P_\Omega((G + t\xi_G)(H + t\xi_H)^T - M)(H + t\xi_H) \qquad (92)\\ &\qquad - P_\Omega(GH^T - M)H + t\alpha\Theta^{\mathrm{r}}\xi_G\Big]\\ &= P_\Omega(G\xi_H^T + \xi_G H^T)H + S\xi_H + \alpha\Theta^{\mathrm{r}}\xi_G. \qquad (93)\end{aligned}$$

$$\begin{aligned}\frac{d}{dt}\partial_H f\left(x+t\xi\right)|_{t=0} &= \lim_{t\to 0}\frac{1}{t}\Big[P_\Omega((G+t\xi_G)(H+t\xi_H)^T-M)^T(G+t\xi_G) \quad (94)\\ &\quad -P_\Omega(GH^T-M)^TG+t\alpha\Theta^{\mathrm{c}}\xi_H\Big]\\ &= P_\Omega(G\xi_H^T+\xi_GH^T)^TG+S^T\xi_G+\alpha\Theta^{\mathrm{c}}\xi_H. \quad (95)\end{aligned}$$

Hence we have

$$\nabla^2 f(x)[\xi]=\begin{pmatrix}P_\Omega(G\xi_H^T+\xi_GH^T)H+S\xi_H+\alpha\Theta^{\mathrm{r}}\xi_G\\ P_\Omega(G\xi_H^T+\xi_GH^T)^TG+S^T\xi_G+\alpha\Theta^{\mathrm{c}}\xi_H\end{pmatrix}. \quad (96)$$

Appendix B The graph-based regularization

B.1 A low-rank matrix model with graph information

The model (51) is of particular interest because it models a wide range of real data matrices whose entries present pairwise similarities. Figure 1 shows differences between Z (51a) and $A^{\mathrm{r}}Z$ (51b) in both the data entries and in the graph spectral domain. By using the concept of graph Fourier transforms (*e.g.* [46]), we illustrate how $(g, A^{\mathrm{r}}, A^{\mathrm{c}})$ in (52) transforms a Gaussian random matrix Z into a matrix with more apparent pairwise similarities on the given graphs.

By definition (*e.g.* [46]), the graph Fourier transform of a vector $f\in\mathbb{R}^m$ with respect to the graph Laplacian $L^{\mathrm{r}}=U\Lambda U^T$, is

$$\widehat{f}(\lambda_l)=(Ue_l)^Tf, \forall l\in[\![m]\!].$$

Now we compare the smoothness of the Gaussian low-rank model $Z=FQ^T$ in (51a) and the graph-based model $X=A^{\mathrm{r}}Z$ (51b) with respect to the row-wise similarity graph L^{r}: For any $j=1,..,k$, the graph Fourier coefficients of the j-th column of the Gaussian random matrix $F\in\mathbb{R}^{m\times k}$ and transformed matrix $G=A^{\mathrm{r}}F$ are

$$\begin{aligned}\widehat{F_{(j)}}(\lambda_l) &= (U^Te_l)^TF_{(j)},\\ \widehat{G_{(j)}}(\lambda_l) &= (U^Te_l)^TA^{\mathrm{r}}F_{(j)}=\sqrt{g(\lambda_l)}e_l^TF_{(j)}, \forall l\in[\![m]\!],\end{aligned}$$

where $F_{(j)}\overset{\text{i.i.d.}}{\sim}\mathcal{N}(0,\sigma_F^2I_m)$. From basic calculations, the amplitudes of their graph Fourier coefficients satisfy

$$\begin{aligned}\mathbb{E}\left[|\widehat{F_{(j)}}(\lambda_l)|^2\right] &= \sum_{i=1}^m U_{(i,l)}^2\mathbb{E}\left[F_{(i,j)}^2\right]=\sigma_F^2\sum_{i=1}^m U_{(i,l)}^2=\sigma_F^2, \quad (97)\\ \mathbb{E}\left[|\widehat{G_{(j)}}(\lambda_l)|^2\right] &= g(\lambda_l)\mathbb{E}\left[\|e_l^TF_{(j)}\|_2^2\right]=\sigma_F^2 g(\lambda_l) \quad (98)\end{aligned}$$

Therefore, when g is decreasing on $(0,+\infty)$ as defined in (53), the sequence $(g(\lambda_l))_l$ is decreasing for increasing values of $(\lambda_l)_{l=2,..,m}$. Note that a small eigenvalue λ corresponds eigenfunctions on the graph with small variations. This means the energy of $G_{(j)}$ in the graph Fourier domain, determined by $(|\widehat{G_{(j)}}(\lambda_l)|)_{1\le l\le m}$, is mostly concentrated on the "low graph-vertex frequencies".

The overall variations of the matrix factor G is related to $\operatorname{Tr}\left(G^TL^{\mathrm{r}}G\right)$ in the regularizer of our main problem (10) as follows,

$$\frac{1}{k}\operatorname{Tr}\left(G^TL^{\mathrm{r}}G\right) = \frac{1}{k}\sum_{j=1}^k\sum_{l=1}^m\lambda_l^2|\widehat{G_{(j)}}(\lambda_l)|^2=\sum_{l=1}^m\lambda_l^2\tilde{\mathbb{E}}|\widehat{G_{(1)}}(\lambda_l)|^2. \quad (99)$$

The weighted-sum expression (99) dictates that $\operatorname{Tr}\left(G^T L^{\mathrm{r}} G\right)$ is small when the amplitudes $(\|\widehat{G_{(1)}}(\lambda_l)\|)_l$ are concentrated on low-frequencies, such as in (98). The same property applies to the factor H with respect to L^{c}. This reflects that the graph-based regularizer

$$S_L(x) = \operatorname{Tr}\left(G^T L^{\mathrm{r}} G\right) + \operatorname{Tr}\left(H^T L^{\mathrm{c}} H\right),$$

quantifies the smoothness of of the entries of (G, H) on the row and column index sets with respect to the row-wise and column-wise similarity graphs ($\mathcal{G}^{\mathrm{r}}$ and $\mathcal{G}^{\mathrm{c}}$), as explained in (2).

References

[1] M. Belkin, I. Matveeva, and P. Niyogi. Tikhonov regularization and semi-supervised learning on large graphs. In *2004 IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 3, pages iii–1000. IEEE, 2004.

[2] M. Belkin and P. Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural computation*, 15(6):1373–1396, 2003.

[3] N. Boumal, P. A. Absil, and C. Cartis. Global rates of convergence for nonconvex optimization on manifolds. *IMA Journal of Numerical Analysis*, 39(1):1–33, 2019.

[4] N. Boumal, B. Mishra, P.-A. Absil, and R. Sepulchre. Manopt, a Matlab toolbox for optimization on manifolds. *Journal of Machine Learning Research*, 15:1455–1459, 2014.

[5] E. Candès and B. Recht. Exact low-rank matrix completion via convex optimization. *2008 46th Annual Allerton Conference on Communication, Control, and Computing*, (m):1–49, 2008.

[6] E. J. Candès and B. Recht. Exact Matrix Completion via Convex Optimization. *Foundations of Computational Mathematics*, 9(6):717–772, 2009.

[7] E. J. Candès and T. Tao. The power of convex relaxation: Near-optimal matrix completion. *IEEE Transactions on Information Theory*, 56(5):2053–2080, 2010.

[8] V. Chandrasekaran, B. Recht, P. A. Parrilo, and A. S. Willsky. The Convex Geometry of Linear Inverse Problems. *Foundations of Computational Mathematics*, 12(6):805–849, 2012.

[9] B. Chazelle. An improved algorithm for the fixed-radius neighbor problem. *Information Processing Letters*, 16(4):193–198, 1983.

[10] J. Chen, H. A. Gov, and Y. Saad. Fast Approximate kNN Graph Construction for High Dimensional Data via Recursive Lanczos Bisection Haw-ren Fang. *Journal of Machine Learning Research*, 10, 2009.

[11] F. R. K. Chung. *Spectral Graph Theory.* American Mathematical Society, 1997.

[12] R. R. Coifman, S. Lafon, A. B. Lee, M. Maggioni, B. Nadler, F. Warner, and S. W. Zucker. Geometric diffusions as a tool for harmonic analysis and structure definition of data: Diffusion maps. *Proceedings of the national academy of sciences*, 102(21):7426–7431, 2005.

[13] R. R. Coifman and M. Maggioni. Diffusion wavelets. *Applied and Computational Harmonic Analysis*, 21(1):53–94, 2006.

[14] S. Dong, P.-A. Absil, and K. A. Gallivan. Preconditioned Conjugate Gradient Algorithms for Graph Regularized Matrix Completion. In *European Symposium on Artificial Neural Networks (ESANN)*, pages 239–244, 2019.

[15] R. Fletcher and C. M. Reeves. Function minimization by conjugate gradients. *The Computer Journal*, 7(2):149–154, 1964.

[16] R. Ge, C. Jin, and Y. Zheng. No spurious local minima in nonconvex low rank problems: A unified geometric analysis. *34th International Conference on Machine Learning, ICML 2017*, 3:1990–2028, 2017.

[17] W. W. Hager and H. Zhang. A Survey of Nonlinear Conjugate Gradient Methods. *Pacific journal of Optimization*, 2(1):35–58, 2006.

[18] M. Hardt. Understanding alternating minimization for matrix completion. *Proceedings - Annual IEEE Symposium on Foundations of Computer Science, FOCS*, pages 651–660, 2014.

[19] F. M. Harper and J. A. Konstan. The MovieLens Datasets: History and Context. *ACM Transactions on Interactive Intelligent Systems (TiiS) 5, 4, Article 19 (December 2015), 19 pages.*, 2015.

[20] T. Hastie, R. Mazumder, J. D. Lee, and R. Zadeh. Matrix completion and low-rank SVD via fast alternating least squares. *The Journal of Machine Learning Research*, 16(1):3367–3402, 2015.

[21] X. He and P. Niyogi. Locality preserving projections. In *Advances in neural information processing systems*, pages 153–160, 2004.

[22] M. R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *Journal of research of the National Bureau of Standards*, 49:409–436, 1952.

[23] P. Jain and I. S. Dhillon. Provable Inductive Matrix Completion. Technical report, 2013.

[24] P. Jain, P. Netrapalli, and S. Sanghavi. Low-rank matrix completion using alternating minimization. *Proceedings of the 45th annual ACM symposium on Symposium on theory of computing - STOC '13*, page 665, 2013.

[25] G. James, D. Witten, T. Hastie, and R. Tibshirani. *An introduction to statistical learning*, volume 112. Springer, 2013.

[26] V. Kalofolias, X. Bresson, M. Bronstein, and P. Vandergheynst. Matrix Completion on Graphs. In *NIPS2014 - Robustness in High Dimension*, 2014.

[27] R. H. Keshavan, A. Montanari, and S. Oh. Matrix completion from noisy entries. *Journal of Machine Learning Research*, 11(Jul):2057–2078, 2010.

[28] R. H. Keshavan and S. Oh. OptSpace : A Gradient Descent Algorithm on the Grassman Manifold for Matrix Completion. 2009.

[29] C. Ma, K. Wang, Y. Chi, and Y. Chen. Implicit regularization in nonconvex statistical estimation: Gradient descent converges linearly for phase retrieval and matrix completion. In *35th International Conference on Machine Learning, ICML 2018*, volume 8, pages 5264–5331, 2018.

[30] J. Mairal, F. Bach, J. Ponce, and G. Sapiro. Online learning for matrix factorization and sparse coding. *Journal of Machine Learning Research*, 11(Jan):19–60, 2010.

[31] R. Mazumder, T. Hastie, H. Edu, R. Tibshirani, T. Edu, and T. Jaakkola. Spectral Regularization Algorithms for Learning Large Incomplete Matrices. *Journal of Machine Learning Research*, 11:2287–2322, 2010.

[32] G. Meyer, S. Bonnabel, and R. Sepulchre. Linear regression under fixed-rank constraints: a Riemannian approach. In *Proceedings of the 28th international conference on machine learning*, 2011.

[33] B. Mishra, K. A. Apuroop, and R. Sepulchre. A Riemannian geometry for low-rank matrix completion. *arXiv preprint arXiv:1211.1550*, 2012.

[34] B. Mishra, G. Meyer, F. Bach, and R. Sepulchre. Low-rank optimization with trace norm penalty. *SIAM Journal on Optimization, Society for Industrial and Applied Mathematics*, 23(4):2124–2149, 2013.

[35] B. Mishra, G. Meyer, S. Bonnabel, and R. Sepulchre. Fixed-rank matrix factorizations and Riemannian low-rank optimization. *Computational Statistics*, 29(3-4):591–621, 2014.

[36] Y. Nesterov. *Introductory Lectures on Convex Optimization*, volume 87. Springer Publishing Company, Incorporated, 1 edition, 2004.

[37] T. Ngo and Y. Saad. Scaled gradients on Grassmann manifolds for matrix completion. In *Advances in Neural Information Processing Systems*, pages 1412–1420, 2012.

[38] L. T. Nguyen, J. Kim, and B. Shim. Low-Rank Matrix Completion: A Contemporary Survey. *IEEE Access*, 7:94215–94237, jul 2019.

[39] B. A. Olshausen and D. J. Field. Sparse coding with an overcomplete basis set: A strategy employed by V1? *Vision Research*, 37(23):3311–3325, 1997.

[40] N. Perraudin, J. Paratte, D. Shuman, L. Martin, V. Kalofolias, P. Vandergheynst, and D. K. Hammond. GSPBOX: A toolbox for signal processing on graphs. *ArXiv e-prints*, aug 2014.

[41] E. Polak and G. Ribiere. Note sur la convergence de méthodes de directions conjuguées. *Rev. Francaise Informat Recherche Opertionelle*, pages 35–43, 1969.

[42] N. Rao, H.-F. Yu, P. Ravikumar, and I. S. Dhillon. Collaborative Filtering with Graph Information: Consistency and Scalable Methods. In *Advances in Neural Information Processing Systems 28*, pages 2107–2115. 2015.

[43] B. Recht, M. Fazel, and P. A. Parrilo. Guaranteed minimum-rank solutions of linear matrix equations via nuclear norm minimization. *SIAM Review*, 52(3):471–501, 2010.

[44] J. D. M. Rennie and N. Srebro. Fast maximum margin matrix factorization for collaborative prediction. In *Proceedings of the 22nd international conference on Machine learning - ICML '05*, pages 713–719, New York, New York, USA, 2005. ACM Press.

[45] D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst. The emerging field of signal processing on graphs. *IEEE Signal Processing Magazine*, 30(3):83–98, 2013.

[46] D. I. Shuman, B. Ricaud, and P. Vandergheynst. Vertex-frequency analysis on graphs. *Applied and Computational Harmonic Analysis*, 40(2):260–291, mar 2016.

[47] D. Spielman. *Spectral Graph Theory.* Combinatorial Scientific Computing. Chapman and Hall/CRC Press, 2012.

[48] N. Srebro, J. D. Rennie, and T. S. Jaakkola. Maximum-margin matrix factorization. *Advances in Neural Information Processing Systems*, 17:1329–1336, 2005.

[49] R. Sun and Z. Q. Luo. Guaranteed Matrix Completion via Non-Convex Factorization. *IEEE Transactions on Information Theory*, 62(11):6535–6579, 2016.

[50] A. Uschmajew and B. Vandereycken. Geometric methods on low-rank matrix and tensor manifolds. Technical report.

[51] B. Vandereycken. Low-Rank Matrix Completion by Riemannian Optimization. *SIAM Journal on Optimization*, 23(2):1214–1236, 2013.

[52] Y.-X. Wang and Y.-J. Zhang. Nonnegative matrix factorization: A comprehensive review. *IEEE Transactions on Knowledge and Data Engineering*, 25(6):1336–1353, 2012.

[53] M. Xu, R. Jin, and Z.-H. Zhou. Speedup matrix completion with side information: Application to multi-label learning. In *Advances in neural information processing systems*, pages 2301–2309, 2013.

[54] Y. Xu and W. Yin. A Block Coordinate Descent Method for Regularized Multiconvex Optimization with Applications to Nonnegative Tensor Factorization and Completion. *SIAM Journal on Imaging Sciences*, 6(3):1758–1789, 2013.

[55] H.-F. Yu, N. Rao, and I. S. Dhillon. Temporal Regularized Matrix Factorization for High-dimensional Time Series Prediction. In *Advances in Neural Information Processing Systems 29*, pages 847–855, 2016.

[56] F. Zhang and E. R. Hancock. Graph spectral image smoothing using the heat kernel. *Pattern Recognition*, 41(11):3328–3342, 2008.

[57] H. Zhang and S. Sra. First-order methods for geodesically convex optimization. In *Conference on Learning Theory*, pages 1617–1638, 2016.

[58] X. Zhang, S. Du, and Q. Gu. Fast and Sample Efficient Inductive Matrix Completion via Multi-Phase Procrustes Flow. In J. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 5756–5765, Stockholmsmässan, Stockholm Sweden, 2018. PMLR.

[59] Z. Zhao, L. Zhang, X. He, and W. Ng. Expert Finding for Question Answering via Graph Regularized Matrix Completion. *IEEE Transactions on Knowledge and Data Engineering*, 27(4):993–1004, apr 2015.

[60] G. Zhou, W. Huang, K. A. Gallivan, P. Van Dooren, and P. A. Absil. A Riemannian rank-adaptive method for low-rank optimization. *Neurocomputing*, 192:72–80, jun 2016.

[61] T. Zhou, H. Shan, A. Banerjee, and G. Sapiro. Kernelized probabilistic matrix factorization: Exploiting graphs and side information. In *Proceedings of the 2012 SIAM international Conference on Data mining*, pages 403–414. SIAM, 2012.