SIAM J. OPTIM.
Vol. 23, No. 2, pp. 1214–1236

LOW-RANK MATRIX COMPLETION BY RIEMANNIAN OPTIMIZATION*

BART VANDEREYCKEN†

Abstract. The matrix completion problem consists of finding or approximating a low-rank matrix based on a few samples of this matrix. We propose a new algorithm for matrix completion that minimizes the least-square distance on the sampling set over the Riemannian manifold of fixed-rank matrices. The algorithm is an adaptation of classical nonlinear conjugate gradients, developed within the framework of retraction-based optimization on manifolds. We describe all the necessary objects from differential geometry necessary to perform optimization over this low-rank matrix manifold, seen as a submanifold embedded in the space of matrices. In particular, we describe how metric projection can be used as retraction and how vector transport lets us obtain the conjugate search directions. Finally, we prove convergence of a regularized version of our algorithm under the assumption that the restricted isometry property holds for incoherent matrices throughout the iterations. The numerical experiments indicate that our approach scales very well for large-scale problems and compares favorably with the state-of-the-art, while outperforming most existing solvers.

Key words. matrix completion, low-rank matrices, optimization on manifolds, differential geometry, nonlinear conjugate gradients, Riemannian manifolds, Newton

AMS subject classifications. 15A83, 65K05, 53B21

DOI. 10.1137/110845768

1. Introduction.

Let $A \in \mathbb{R}^{m\times n}$ be an $m \times n$ matrix that is only known on a subset Ω of the complete set of entries $\{1,\dots,m\} \times \{1,\dots,n\}$. The low-rank matrix completion problem [16] consists of finding the matrix with lowest rank that agrees with A on Ω:

$$\begin{aligned} &\underset{X}{\text{minimize}} && \operatorname{rank}(X), \\ &\text{subject to} && X \in \mathbb{R}^{m\times n},\ \mathrm{P}_\Omega(X) = \mathrm{P}_\Omega(A), \end{aligned} \tag{1.1}$$

where

$$\mathrm{P}_\Omega \colon \mathbb{R}^{m\times n} \to \mathbb{R}^{m\times n},\ X_{i,j} \mapsto \begin{cases} X_{i,j} & \text{if } (i,j) \in \Omega, \\ 0 & \text{if } (i,j) \notin \Omega \end{cases} \tag{1.2}$$

denotes projection onto Ω. Without loss of generality, we assume $m \le n$.

Due to the presence of noise, it is advisable to relax the equality constraint in (1.1) to allow for misfit. Then, given a tolerance $\varepsilon \ge 0$, a more robust version of the rank minimization problem becomes

$$\begin{aligned} &\underset{X}{\text{minimize}} && \operatorname{rank}(X), \\ &\text{subject to} && X \in \mathbb{R}^{m\times n},\ \|\mathrm{P}_\Omega(X) - \mathrm{P}_\Omega(A)\|_{\mathrm{F}} \le \varepsilon, \end{aligned} \tag{1.3}$$

where $\|X\|_{\mathrm{F}}$ denotes the Frobenius norm of X.

*Received by the editors August 25, 2011; accepted for publication (in revised form) March 12, 2013; published electronically June 20, 2013.
http://www.siam.org/journals/siopt/23-2/84576.html

†Chair of Numerical Algorithms and HPC, MATHICSE, École Polytechnique Fédérale de Lausanne, CH-1015 Lausanne, Switzerland (bartv@math.princeton.edu).

Matrix completion has a number of interesting applications such as collaborative filtering, system identification, and global positioning, but unfortunately it is NP hard. Recently, there has been a considerable body of work devoted to the identification of large classes of matrices for which (1.1) has a unique solution that can be recovered in polynomial time. In [15], for example, the authors show that when Ω is sampled uniformly at random, the nuclear norm relaxation

$$\begin{aligned} &\underset{X}{\text{minimize}} && \|X\|_*, \\ &\text{subject to} && X \in \mathbb{R}^{m\times n},\ \mathrm{P}_\Omega(X) = \mathrm{P}_\Omega(A), \end{aligned} \tag{1.4}$$

can recover with high probability any matrix A of rank k that has so-called low incoherence, provided that the number of samples is large enough, $|\Omega| > Cnk\,\mathrm{polylog}(n)$. Related work has been done by [14, 16, 27, 28], which in particular also establish similar recovery results for the robust formulation (1.3).

Many of the potential applications for matrix completion involve very large data sets; the Netflix matrix, for example, has more than 10^8 entries [6]. It is therefore crucial to develop algorithms that can cope with such a large-scale setting, but, unfortunately, solving (1.4) by off-the-shelf methods for convex optimization scales very badly in the matrix dimension. This has spurred a considerable amount of algorithms that aim to solve the nuclear norm relaxation by specifically designed methods that try to exploit the low-rank structure of the solution; see, e.g., [13, 21, 35, 36, 37, 38, 50]. Other approaches include optimization on the Grassmann manifold [5, 18, 27]; atomic decompositions [33]; and nonlinear SOR [53].

1.1. The proposed method: Optimization on manifolds. We present a new method for low-rank matrix completion based on a direct optimization over the set of all fixed-rank matrices. By prescribing the rank of the global minimizer of (1.3), say k, the robust matrix completion problem is equivalent to

$$\begin{aligned} &\underset{X}{\text{minimize}} && f(X) := \tfrac{1}{2}\|\mathrm{P}_\Omega(X - A)\|_{\mathrm{F}}^2, \\ &\text{subject to} && X \in \mathcal{M}_k := \{X \in \mathbb{R}^{m\times n} : \mathrm{rank}(X) = k\}. \end{aligned} \tag{1.5}$$

It is well known that $\mathcal{M}_k$ is a smooth (C^∞) manifold; it can, for example, be identified as the smooth part of the determinantal variety of matrices of rank at most k [10, Proposition 1.1]. Since the objective function f is also smooth, problem (1.5) is a smooth optimization problem that can be solved by methods from Riemannian optimization as introduced, among others, by [2, 4, 20]. Simply put, Riemannian optimization is the generalization of standard unconstrained optimization, where the search space is $\mathbb{R}^n$, to optimization of a smooth objective function on a Riemannian manifold.

For solving the optimization problem (1.5), in principle any method from optimization on Riemannian manifolds could be used. In this paper, we use a generalization of classical nonlinear conjugate gradients (CGs) on Euclidean space to perform optimization on manifolds; see, e.g., [2, 20, 49]. The reason for this choice compared to, say, Newton's method was that CG performed best in our numerical experiments. The skeleton of the proposed method, LRGeomCG, is listed in Algorithm 1.

Algorithm 1 is derived using concepts from differential geometry, yet it closely resembles a typical nonlinear CG algorithm with Armijo line-search for unconstrained optimization. It is schematically visualized in Figure 1.1 for iteration number i and relies on the following crucial ingredients, which will be explained in more detail for (1.5) in section 2.

ALGORITHM 1. LRGEOMCG: GEOMETRIC CG FOR (1.5).

Require: initial iterate $X_1 \in \mathcal{M}_k$, tolerance $\tau > 0$, tangent vector $\eta_0 = 0$

1: **for** $i = 1, 2, \ldots$ **do**
2: Compute the gradient
$\xi_i := \operatorname{grad} f(X_i)$ # see Algorithm 2
3: Check convergence
if $\|\xi_i\| \leq \tau$, then break
4: Compute a conjugate direction by PR+
$\eta_i := -\xi_i + \beta_i \mathcal{T}_{X_{i-1} \to X_i}(\eta_{i-1})$ # see Algorithm 4
5: Compute an initial step t_i in closed-form from a straight-line search
$t_i = \arg\min_t f(X_i + t\,\eta_i)$ # see Algorithm 5
6: Perform Armijo backtracking to find the smallest integer $m \geq 0$ such that
$f(X_i) - f(R_{X_i}(0.5^m\, t_i\, \eta_i)) \geq -0.0001 \times 0.5^m\, t_i \langle \xi_i, \eta_i \rangle$
and obtain the new iterate
$X_{i+1} := R_{X_i}(0.5^m\, t_i\, \eta_i)$ # see Algorithm 6
7: **end for**

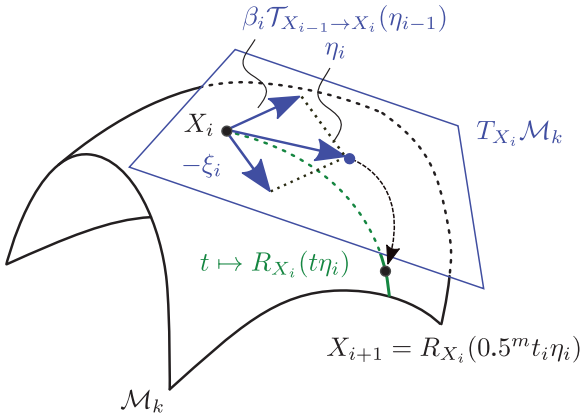

FIG. 1.1. *Visualization of Algorithm* 1: *nonlinear CG on a Riemannian manifold.*

1. The Riemannian gradient, denoted $\operatorname{grad} f(X_i)$, is a specific tangent vector ξ_i which corresponds to the direction of steepest ascent of $f(X_i)$ but is restricted to only directions in the tangent space $T_{X_i}\mathcal{M}_k$.
2. The search direction $\eta_i \in T_{X_i}\mathcal{M}_k$ is conjugate to the gradient and is computed by a variant of the classical Polak–Ribière updating rule in nonlinear CG. This requires taking a linear combination of the Riemannian gradient with the previous search direction η_{i-1}. Since η_{i-1} does not lie in $T_{X_i}\mathcal{M}_k$, it needs to be transported to $T_{X_i}\mathcal{M}_k$. This is done by a mapping $\mathcal{T}_{X_{i-1} \to X_i} : T_{X_{i-1}}\mathcal{M}_k \to T_{X_i}\mathcal{M}_k$, the so-called vector transport.
3. As a tangent vector only gives a direction but not the line search itself on the manifold, a smooth mapping $R_{X_i} : T_{X_i}\mathcal{M}_k \to \mathcal{M}_k$, called the *retraction*, is needed to map tangent vectors to the manifold. Using the conjugate direction η_i, a line search can then be performed along the curve $t \mapsto R_{X_i}(t\,\eta_i)$. Step 6 uses a standard backtracking procedure where we have chosen to fix the constants. More judiciously chosen constants can sometimes improve the line search, but our numerical experiments indicate that this is not necessary in the setting under consideration.

1.2. Relation to existing manifold-related methods. At the time of submitting the present work, a large number of other matrix completion solvers based on Riemannian optimization have been proposed in [8, 9, 19, 27, 28, 40, 41, 43, 48]. Like the current paper, all of these algorithms use the concept of retraction-based optimization on the manifold of fixed-rank matrices, but they differ in their specific choice of Riemannian manifold structure and metric. It remains a topic of further investigation to assess the performance of these different geometries with respect to each other and to nonmanifold based solvers.

A first attempt at such a comparison has been very recently done in [44], where the previous geometries were tested to complete one large matrix. The authors reach the conclusion that for gradient-based algorithms there exists a set of geometries—including ours—that perform remarkably more or less the same with respect to the total computational time, and that overall most geometries outperformed the state-of-the art. In addition, Newton-based algorithms performed well when high precision was required and now the algorithm based on our embedded submanifold geometry was clearly faster.

1.3. Outline of the paper. The plan of the paper is as follows. In the next section, the necessary concepts of differential geometry are explained to turn Algorithm 1 into a concrete method. The implementation of each step is explained in section 3. We also prove convergence of a slightly modified version of this method in section 4 under some assumptions which are reasonable for matrix completion. Numerical experiments and comparisons to the state-of-the art are carried out in section 5. The last section is devoted to conclusions.

2. Differential geometry for low-rank matrix manifolds. In this section, we explain the differential geometry concepts used in Algorithm 1 applied to our particular matrix completion problem (1.5).

2.1. The Riemannian manifold. Let

$$\mathcal{M}_k = \{X \in \mathbb{R}^{m\times n} \colon \operatorname{rank}(X) = k\}$$

denote the manifold of fixed-rank matrices. Using the SVD, one has the equivalent characterization

$$\mathcal{M}_k = \{U\Sigma V^T \colon U \in \mathrm{St}_k^m,\, V \in \mathrm{St}_k^n,\, \Sigma = \operatorname{diag}(\sigma_i),\, \sigma_1 \geq \cdots \geq \sigma_k > 0\}, \tag{2.1}$$

where St_k^m is the Stiefel manifold of $m \times k$ real, orthonormal matrices, and $\operatorname{diag}(\sigma_i)$ denotes a diagonal matrix with σ_i on the diagonal. Whenever we use the notation $U\Sigma V^T$ in the rest of the paper, we mean matrices that satisfy (2.1). Furthermore, the constants k, m, n are always used to denote dimensions of matrices, and, for simplicity, we assume $1 \leq k < m \leq n$.

The following proposition shows that $\mathcal{M}_k$ is indeed a smooth manifold. While the existence of such a smooth manifold structure, together with its tangent space, is well known (see, e.g., [25, 29] for applications of gradient flows on $\mathcal{M}_k$), more advanced concepts like retraction-based optimization on $\mathcal{M}_k$ have only very recently been investigated; see [47, 48]. In contrast, the case of optimization on *symmetric* fixed-rank matrices has been studied in more detail in [26, 39, 45, 51].

PROPOSITION 2.1. *The set $\mathcal{M}_k$ is a smooth submanifold of dimension $(m+n-k)k$ embedded in $\mathbb{R}^{m\times n}$. Its tangent space $T_X\mathcal{M}_k$ at $X = U\Sigma V^T \in \mathcal{M}_k$ is given by*

$$T_X\mathcal{M}_k = \left\{ \begin{bmatrix} U & U_\perp \end{bmatrix} \begin{bmatrix} \mathbb{R}^{k\times k} & \mathbb{R}^{k\times(n-k)} \\ \mathbb{R}^{(m-k)\times k} & 0_{(m-k)\times(n-k)} \end{bmatrix} \begin{bmatrix} V & V_\perp \end{bmatrix}^T \right\} \tag{2.2}$$

$$= \{UMV^T + U_pV^T + UV_p^T : M \in \mathbb{R}^{k\times k}, \\ U_p \in \mathbb{R}^{m\times k},\ U_p^TU = 0,\ V_p \in \mathbb{R}^{n\times k},\ V_p^TV = 0\}. \tag{2.3}$$

Proof. See [32, Example 8.14] for a proof that uses only elementary differential geometry based on local submersions. The tangent space is obtained by the first-order perturbation of the SVD and counting dimensions. ∎

The *tangent bundle* is defined as the disjoint union of all tangent spaces,

$$T\mathcal{M}_k := \bigcup_{X\in\mathcal{M}_k} \{X\} \times T_X\mathcal{M}_k = \{(X,\xi) \in \mathbb{R}^{m\times n} \times \mathbb{R}^{m\times n} : X \in \mathcal{M}_k,\ \xi \in T_X\mathcal{M}_k\}.$$

By restricting the Euclidean inner product on $\mathbb{R}^{m\times n}$,

$$\langle A, B\rangle = \operatorname{tr}(A^TB) \quad \text{with } A, B \in \mathbb{R}^{m\times n},$$

to the tangent bundle, we turn $\mathcal{M}_k$ into a Riemannian manifold with Riemannian metric

$$g_X(\xi,\eta) := \langle \xi, \eta\rangle = \operatorname{tr}(\xi^T\eta) \quad \text{with } X \in \mathcal{M}_k \text{ and } \xi,\eta \in T_X\mathcal{M}_k, \tag{2.4}$$

where the tangent vectors ξ, η are seen as matrices in $\mathbb{R}^{m\times n}$.

Once the metric is fixed, the notion of the gradient of an objective function can be introduced. For a Riemannian manifold, the Riemannian gradient of a smooth function $f : \mathcal{M}_k \to \mathbb{R}$ at $X \in \mathcal{M}_k$ is defined as the unique tangent vector $\operatorname{grad} f(X)$ in $T_X\mathcal{M}_k$ such that

$$\langle \operatorname{grad} f(X), \xi \rangle = \mathrm{D}\, f(X)[\xi] \quad \text{for all } \xi \in T_X\mathcal{M}_k,$$

where we denoted directional derivatives by $\mathrm{D}\, f$. Since $\mathcal{M}_k$ is embedded in $\mathbb{R}^{m\times n}$, the Riemannian gradient is given as the orthogonal projection onto the tangent space of the gradient of f seen as a function on $\mathbb{R}^{m\times n}$; see, e.g., [2, equation (3.37)]. Defining $\mathrm{P}_U := UU^T$ and $\mathrm{P}_U^\perp := I - \mathrm{P}_U$ for any $U \in \mathrm{St}_k^m$, we denote the orthogonal projection onto the tangent space at X as

$$\mathrm{P}_{T_X\mathcal{M}_k} : \mathbb{R}^{m\times n} \to T_X\mathcal{M}_k,\ Z \mapsto \mathrm{P}_U\, Z\, \mathrm{P}_V + \mathrm{P}_U^\perp\, Z\, \mathrm{P}_V + \mathrm{P}_U\, Z\, \mathrm{P}_V^\perp\,. \tag{2.5}$$

Then, using $\mathrm{P}_\Omega(X - A)$ as the (Euclidean) gradient of $f(X) = \|\,\mathrm{P}_\Omega(X - A)\|_\mathrm{F}^2/2$, we obtain

$$\operatorname{grad} f(X) := \mathrm{P}_{T_X\mathcal{M}_k}(\mathrm{P}_\Omega(X - A)). \tag{2.6}$$

2.2. Metric projection as retraction. As explained in section 1.1, we need a so-called retraction mapping to go back from an element in the tangent space to the manifold. Retractions are essentially first-order approximations of the exponential map of the manifold; see, e.g., [3, Definition 1]. In our setting, we have chosen metric projection as a retraction:

$$R_X : \mathcal{U}_X \to \mathcal{M}_k,\ \xi \mapsto \mathrm{P}_{\mathcal{M}_k}(X + \xi) := \operatorname*{arg\,min}_{Z\in\mathcal{M}_k} \|X + \xi - Z\|_\mathrm{F}, \tag{2.7}$$

where $\mathcal{U}_X \subset T_X\mathcal{M}_k$ is a suitable neighborhood around zero and $\mathrm{P}_{\mathcal{M}_k}$ is the orthogonal projection onto $\mathcal{M}_k$. Based on [34, Lemma 2.1], it can be easily shown that R_X indeed satisfies the conditions of a retraction, which allows us to invoke the convergence proofs of [2].

Mapping R_X can be computed in closed-form by the SVD: Let $X \in \mathcal{M}_k$ be given, and then for sufficiently small $\xi \in \mathcal{U}_X \subset T_X\mathcal{M}_k$, we have

$$R_X(\xi) = \mathrm{P}_{\mathcal{M}_k}(X+\xi) = \sum_{i=1}^{k} \sigma_i u_i v_i^T, \tag{2.8}$$

where σ_i, u_i, v_i are the (ordered) singular values and vectors of the SVD of $X+\xi$. It is obvious that when $\sigma_k = 0$, there is no best rank-k solution for (2.7), and additionally, when $\sigma_{k-1} = \sigma_k$ the minimization in (2.7) does not have a unique solution. In fact, since the manifold is nonconvex, metric projections can never be well defined on the whole tangent bundle. Fortunately a retraction has to be defined only locally because this is sufficient to establish convergence of the Riemannian algorithms.

2.3. Riemannian Newton on $\mathcal{M}_k$. Although we do not exploit second-order information in Algorithm 1, Newton or its variants may be preferable for some applications. In [44], for example, it was shown that the second-order Riemannian trust-region algorithm of [1] performs very well in combination with our embedded submanifold geometry. We will therefore give the following theorem with an explicit expression of the Riemannian Hessian of f. Its proof is a straightforward but technical analogue of that in [51] for symmetric fixed-rank matrices. It can be found in Appendix A of the extended technical report [52] of this paper.

PROPOSITION 2.2. *For any $X = U\Sigma V^T \in \mathcal{M}_k$ satisfying (2.1) and $\xi \in T_X\mathcal{M}_k$ satisfying (2.3), the Riemannian Hessian of f at X in the direction of ξ satisfies*

$$\begin{aligned}\operatorname{Hess} f(X)[\xi] = {} & \mathrm{P}_U\,\mathrm{P}_\Omega(\xi)\,\mathrm{P}_V + \mathrm{P}_U^{\perp}\left[\mathrm{P}_\Omega(\xi) + \mathrm{P}_\Omega(X-A)V_p\Sigma^{-1}V^T\right]\mathrm{P}_V \\ & + \mathrm{P}_U\left(\mathrm{P}_\Omega(\xi) + U\Sigma^{-1}U_p^T\,\mathrm{P}_\Omega(X-A)\right)\mathrm{P}_V^{\perp}.\end{aligned}$$

2.4. Vector transport. Vector transport was introduced in [2] and [46] as a means to transport tangent vectors from one tangent space to another. In a similar way as retractions are approximations of the exponential mapping, vector transport is the first-order approximation of parallel transport, another important concept in differential geometry. See Figure 2.1 for an illustration of the retraction $\mathcal{T}_{X\to Y} : T_X\mathcal{M}_k \to T_Y\mathcal{M}_k$, and we refer to [46, Definition 2] for an exact definition.

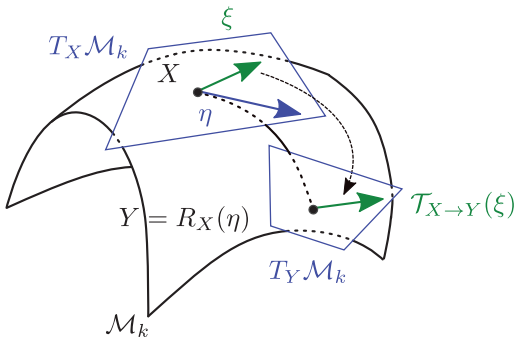

FIG. 2.1. *Vector transport $\mathcal{T}_{X\to Y}$ on a Riemannian manifold.*

Since $\mathcal{M}_k$ is an embedded submanifold of $\mathbb{R}^{m\times n}$, orthogonally projecting the translated tangent vector in $\mathbb{R}^{m\times n}$ onto the new tangent space constitutes a vector transport; see [2, section 8.1.2]. In other words, we define

$$\mathcal{T}_{X\to Y} : T_X\mathcal{M}_k \to T_Y\mathcal{M}_k,\ \xi \mapsto \mathrm{P}_{T_Y\mathcal{M}_k}(\xi) \tag{2.9}$$

with $\mathrm{P}_{T_Y\mathcal{M}_k}$ as defined in (2.5).

As explained in section 1.1, the conjugate search direction η_i in Algorithm 1 is computed as a linear combination of the gradient and the previous direction:

$$\eta_i = -\operatorname{grad} f(X_i) + \beta_i\, \mathcal{T}_{X_{i-1}\to X_i}(\eta_{i-1}),$$

where we transported η_{i-1}. For β_i, we have chosen the geometrical variant of Polak–Ribière (PR+), as introduced in [2, Chapter 8.2]. Again using vector transport, this becomes

$$\beta_i = \frac{\langle \operatorname{grad} f(X_i),\ \operatorname{grad} f(X_i) - \mathcal{T}_{X_{i-1}\to X_i}(\operatorname{grad} f(X_{i-1}))\rangle}{\langle \operatorname{grad} f(X_{i-1}), \operatorname{grad} f(X_{i-1})\rangle}. \tag{2.10}$$

In order to prove convergence, we also enforce that the search direction η_i is sufficiently gradient-related in the sense that its angle with the gradient is never too small; see, e.g., [7, Chapter 1.2].

3. Implementation details. This section is devoted to the implementation of Algorithm 1. For most operations, we will also provide a flop count for the low-rank regime, i.e., $k \ll m \le n$.

Low-rank matrices. Since every element $X \in \mathcal{M}_k$ is a rank k matrix, we store it as the result of a compact SVD:

$$X = U\Sigma V^T$$

with orthonormal matrices $U \in \mathrm{St}_k^m$ and $V \in \mathrm{St}_k^n$ and a diagonal matrix $\Sigma \in \mathbb{R}^{k\times k}$ with decreasing positive entries on the diagonal. For the computation of the objective function and the gradient, we also precompute the sparse matrix $X_\Omega := \mathrm{P}_\Omega(X)$. As explained in the next paragraph, computing X_Ω costs $(m + 2|\Omega|)k$ flops.

Projection operator P_Ω. During the course of Algorithm 1, we require the application of P_Ω, as defined in (1.2), to certain low-rank matrices. By exploiting the low-rank form, this can be done efficiently as follows: Let Z be a rank-k_Z matrix with factorization $Z := Y_1Y_2^T$. (k_Z is possibly different from k, the rank of the matrices in $\mathcal{M}_k$.) Define $Z_\Omega := \mathrm{P}_\Omega(Z)$. Then element (i,j) of Z_Ω is given by

$$(Z_\Omega)_{i,j} = \begin{cases} \sum_{l=1}^{k_Z} Y_1(i,l)Y_2(j,l) & \text{if } (i,j)\in\Omega, \\ 0 & \text{if } (i,j)\notin\Omega. \end{cases} \tag{3.1}$$

The cost for computing Z_Ω is $2|\Omega|k_Z$ flops.

Tangent vectors. A tangent vector $\eta \in T_X\mathcal{M}_k$ at $X = U\Sigma V^T \in \mathcal{M}_k$ will be represented as

$$\eta = UMV^T + U_pV^T + UV_p^T, \tag{3.2}$$

where $M \in \mathbb{R}^{k\times k}$, $U_p \in \mathbb{R}^{m\times k}$ with $U^TU_p = 0$, and $V_p \in \mathbb{R}^{n\times k}$ with $V^TV_p = 0$. After vectorizing this representation, the inner product $\langle \eta, \nu\rangle$ can be computed in $2(m+n)k + 2k^2$ flops.

Since X is available as an SVD, the orthogonal projection onto the tangent space $T_X\mathcal{M}_k$ becomes

$$\mathrm{P}_{T_X\mathcal{M}_k}(Z) := \mathrm{P}_U\, Z\, \mathrm{P}_V + (Z - \mathrm{P}_U\, Z)\, \mathrm{P}_V + \mathrm{P}_U(Z^T - \mathrm{P}_V\, Z^T)^T,$$

where $\mathrm{P}_U := UU^T$ and $\mathrm{P}_V := VV^T$. If Z can be efficiently applied to a vector, evaluating and storing the result of $\mathrm{P}_{T_X\mathcal{M}_k}(Z)$ as a tangent vector in the format (3.2) can also be performed efficiently.

Riemannian gradient. From (2.6), the Riemannian gradient at $X = U\Sigma V^T$ is the orthogonal projection of $\mathrm{P}_\Omega(X - A)$ onto the tangent space at X. Since the sparse matrix $A_\Omega = \mathrm{P}_\Omega(A)$ is given and $X_\Omega = \mathrm{P}_\Omega(X)$ was already precomputed, the gradient can be computed by Algorithm 2. The total cost is $2(n + 2m)k^2 + 4|\Omega|k$ flops.

ALGORITHM 2. CALCULATE GRADIENT $\operatorname{grad} f(X)$.

Require: matrix $X = U\Sigma V^T \in \mathcal{M}_k$, sparse matrix $R = X_\Omega - A_\Omega \in \mathbb{R}^{m\times n}$
Ensure: $\operatorname{grad} f(X) = UMV^T + U_pV^T + UV_p^T \in T_X\mathcal{M}_k$

1: $R_u \leftarrow R^TU, \quad R_v \leftarrow RV$ # $4|\Omega|k$ flops
2: $M \leftarrow U^TR_v$ # $2mk^2$ flops
3: $U_p \leftarrow R_v - UM, \quad V_p \leftarrow R_u - VM^T$ # $2(m+n)k^2$ flops

Vector transport. Let $\nu = UMV^T + U_pV^T + UV_p^T$ be a tangent vector at $X = U\Sigma V^T \in \mathcal{M}_k$. By (2.9), the vector

$$\nu_+ := \mathcal{T}_{X\to X_+}(\nu) = \mathrm{P}_{T_{X_+}\mathcal{M}_k}(\nu)$$

is the transport of ν to the tangent space at some $X_+ = U_+\Sigma_+V_+^T$. It is computed by Algorithm 3 with a total cost of approximately $14(m + n)k^2 + 10k^3$ flops.

ALGORITHM 3. CALCULATE VECTOR TRANSPORT $\mathcal{T}_{X\to X_+}(\nu)$.

Require: matrices $X = U\Sigma V^T \in \mathcal{M}_k$ and $X_+ = U_+\Sigma_+V_+^T \in \mathcal{M}_k$,
tangent vector $\nu = UMV^T + U_pV^T + UV_p^T$
Ensure: $\mathcal{T}_{X\to X_+}(\nu) = U_+M_+V_+^T + U_{p_+}V_+^T + U_+V_{p_+}^T \in T_{X_+}\mathcal{M}_k$

1: $A_v \leftarrow V^TV_+, \quad A_u \leftarrow U^TU_+$ # $2(m+n)k^2$
2: $B_v \leftarrow V_p^TV_+, \quad B_u \leftarrow U_p^TU_+$ # $2(m+n)k^2$
3: $M_+^{(1)} \leftarrow A_u^TMA_v, \quad U_+^{(1)} \leftarrow U(MA_v), \quad V_+^{(1)} \leftarrow V(M^TA_u)$ # $6k^3 + 2(m+n)k^2$
4: $M_+^{(2)} \leftarrow B_u^TA_v, \quad U_+^{(2)} \leftarrow U_pA_v, \quad V_+^{(2)} \leftarrow VB_u$ # $2k^3 + 2(m+n)k^2$
5: $M_+^{(3)} \leftarrow A_u^TB_v, \quad U_+^{(3)} \leftarrow UB_v, \quad V_+^{(3)} \leftarrow V_pA_u$ # $2k^3 + 2(m+n)k^2$
6: $M_+ \leftarrow M_+^{(1)} + M_+^{(2)} + M_+^{(3)}$ # $2k^2$
7: $U_{p_+} \leftarrow U_+^{(1)} + U_+^{(2)} + U_+^{(3)}, \quad U_{p_+} \leftarrow U_{p_+} - U_+(U_+^TU_{p_+})$ # $4mk^2$
8: $V_{p_+} \leftarrow V_+^{(1)} + V_+^{(2)} + V_+^{(3)}, \quad V_{p_+} \leftarrow V_{p_+} - V_+(V_+^TV_{p_+})$ # $4nk^2$

Nonlinear CG. The geometric variant of Polak–Ribière is implemented as Algorithm 4 costing about $28(m + n)k^2 + 20k^3$ flops. In order to improve robustness, we use the PR+ variant and restart when the conjugate direction is almost orthogonal to the gradient.

Initial guess for line search. Although Algorithm 1 uses line search, a good initial guess can greatly enhance performance. We observed in our numerical experiments that an exact minimization on the tangent space alone (so, neglecting the retraction),

$$\min_t f(X + t\,\eta) = \frac{1}{2}\min_t \|\,\mathrm{P}_\Omega(X) + t\,\mathrm{P}_\Omega(\eta) - \mathrm{P}_\Omega(A)\|_{\mathrm{F}}^2, \tag{3.3}$$

ALGORITHM 4. COMPUTE THE CONJUGATE DIRECTION BY PR+.

Require: previous iterate X_{i-1}, previous gradient ξ_{i-1}, previous direction η_{i-1}
current iterate X_i, current gradient ξ_i
Ensure: conjugate direction $\eta_i \in T_{X_i}\mathcal{M}_k$

1: Transport previous gradient and direction to current tangent space:
$\overline{\xi}_i \leftarrow \mathcal{T}_{X_{i-1}\to X_i}(\xi_{i-1})$ # apply Algorithm 3
$\overline{\eta}_i \leftarrow \mathcal{T}_{X_{i-1}\to X_i}(\eta_{i-1})$ # apply Algorithm 3
2: Compute conjugate direction:
$\delta_i \leftarrow \xi_i - \overline{\xi}_i$
$\beta \leftarrow \max(0, \langle \delta_i, \xi_i\rangle / \langle \xi_{i-1}, \xi_{i-1}\rangle)$
$\eta_i \leftarrow -\xi_i + \beta\,\overline{\eta}_i$
3: Compute angle between conjugate direction and gradient:
$\alpha \leftarrow \langle \eta_i, \xi_i\rangle / \sqrt{\langle \eta_i, \eta_i\rangle\ \langle \xi_i, \xi_i\rangle}$
4: Reset to gradient if desired:
if $\alpha \le 0.1$, then $\eta_i \leftarrow \xi_i$

performed extremely well as an initial guess in the sense that backtracking is almost never necessary.

Equation (3.3) is a one-dimensional least-square fit for t on Ω. The closed-form solution of the minimizer t_* satisfies

$$t_* = \langle\, \mathrm{P}_\Omega(\eta), \mathrm{P}_\Omega(A - X)\,\rangle \,/\, \langle\, \mathrm{P}_\Omega(\eta), \mathrm{P}_\Omega(\eta)\,\rangle$$

and is unique when $\eta \neq 0$. As long as Algorithm 1 has not converged, η will always be nonzero since it is the direction of search. The solution to (3.3) is performed by Algorithm 5. In the actual implementation, the sparse matrices N and B are stored as the nonzero entries on Ω. Hence, the total cost is about $2mk^2 + 4|\Omega|(k+1)$ flops.

ALGORITHM 5. COMPUTE THE INITIAL GUESS FOR LINE SEARCH $t_* = \arg\min_t f(X + t\,\eta)$.

Require: iterate $X = U\Sigma V^T$ and projection X_Ω, tangent vector $\eta = UMV^T + U_pV^T + UV_p^T$, sparse matrix $R = A_\Omega - X_\Omega \in \mathbb{R}^{m\times n}$
Ensure: step length t_*

1: $N \leftarrow \mathrm{P}_\Omega(\begin{bmatrix} UM + U_p & U \end{bmatrix} \begin{bmatrix} V & V_p \end{bmatrix}^T)$ # $2mk^2 + 4|\Omega|k$ flops
2: $t_* \leftarrow \mathrm{tr}(N^TR)/\,\mathrm{tr}(N^TN)$ # $4|\Omega|$ flops

Retraction. As shown in (2.8), the retraction (2.7) can be directly computed by the SVD of $X + \xi$. A full SVD of $X + \xi$ is, however, prohibitively expensive since it costs $O(n^3)$ flops. Fortunately, the matrix to retract has the particular form

$$X + \xi = \begin{bmatrix} U & U_p \end{bmatrix} \begin{bmatrix} \Sigma + M & I_k \\ I_k & 0 \end{bmatrix} \begin{bmatrix} V & V_p \end{bmatrix}^T$$

with $X = U\Sigma V^T \in \mathcal{M}_k$ and $\xi = UMV^T + U_pV^T + UV_p^T \in T_X\mathcal{M}_k$. Algorithm 6 now performs a compact QR and an SVD of a small $2k$-by-$2k$ matrix to reduce the flop count to $14(m+n)k^2 + C_{\mathrm{SVD}}k^3$ when $k \ll \min(m,n)$.

Observe that the listing of Algorithm 6 uses MATLAB notation to denote common matrix operations. In addition, the flop count of computing the SVD in step 3 is given as $C_{\mathrm{SVD}}k^3$ since it is difficult to estimate beforehand in general. In practice, the constant C_{SVD} is modest, say, less than 200. Furthermore, in step 4, we have added $\varepsilon_{\mathrm{mach}}$ to Σ_s so that in the unlucky event of zero singular values, the retracted matrix is perturbed to a rank-k matrix in $\mathcal{M}_k$.

ALGORITHM 6. COMPUTE THE RETRACTION BY METRIC PROJECTION.

Require: iterate $X = U\Sigma V^T$, tangent vector $\xi = UMV^T + U_pV^T + UV_p^T$

Ensure: retraction $R_X(\xi) = \mathrm{P}_{\mathcal{M}_k}(X + \xi) = U_+\Sigma_+V_+^T$

1: $(Q_u, R_u) \leftarrow \mathrm{qr}(U_p, 0), \quad (Q_v, R_v) \leftarrow \mathrm{qr}(V_p, 0)$ # $10(m+n)k^2$ flops

2: $S \leftarrow \begin{bmatrix} \Sigma + M & R_v^T \\ R_u & 0 \end{bmatrix}$

3: $(U_s, \Sigma_s, V_s) \leftarrow \mathrm{svd}(S)$ # $C_{\mathrm{SVD}}k^3$ flops

4: $\Sigma_+ \leftarrow \Sigma_s(1:k, 1:k) + \varepsilon_{\mathrm{mach}}$

5: $U_+ \leftarrow \begin{bmatrix} U & Q_u \end{bmatrix} U_s(:, 1:k), \quad V_+ \leftarrow \begin{bmatrix} V & Q_v \end{bmatrix} V_s(:, 1:k)$ # $4(m+n)k^2$ flops

Computational cost. Summing all the flop counts for Algorithm 1, we arrive at a cost per iteration of approximately

$$(48m + 44n)k^2 + 10|\Omega|k + \mathrm{nb}_{\mathrm{Armijo}}\,(8(m+n)k^2 + 2|\Omega|k),$$

where $\mathrm{nb}_{\mathrm{Armijo}}$ denotes the average number of Armijo backtracking steps. It is remarkable that in all experiments below we have observed that $\mathrm{nb}_{\mathrm{Armijo}} = 0$, that is, backtracking was never needed.

For our typical problems in section 5, the size of the sampling set satisfies $|\Omega| = \mathrm{OS}\,(m + n - k)k$ with $\mathrm{OS} > 2$ the oversampling factor (OS). When $m = n$ and $\mathrm{nb}_{\mathrm{Armijo}} = 0$, this brings the total flops per iteration to

$$t_{\mathrm{theoretical}} = (92 + 20\,\mathrm{OS})nk^2. \tag{3.4}$$

Comparing this theoretical flop count with experimental results, we observe that sparse matrix vector multiplications and applying P_Ω require significantly more time than predicted by (3.4). This can be explained due to the lack of data locality for these sparse matrix operations, whereas the majority of the remaining time is spent by dense linear algebra, rich in BLAS3. After some experimentation, we estimated that for our MATLAB environment these sparse operations are penalized by a factor of about $C_{\mathrm{sparse}} \simeq 5$. For this reason, we normalize the theoretical estimate (3.4) to obtain the following practical estimate:

$$t_{\mathrm{practical}}^{\mathrm{LRGeomCG}} = (92 + C_{\mathrm{sparse}}\,20\,\mathrm{OS})nk^2, \quad C_{\mathrm{sparse}} \simeq 5. \tag{3.5}$$

For a sensible value of $\mathrm{OS} = 3$, we can expect that about 75% of the computational time will be spent on operations with sparse matrices.

Comparison to existing methods. The vast majority of specialized solvers for matrix completion require computing the dominant singular vectors of a sparse matrix in each step of the algorithm. This is, for example, the case for approaches using soft- and hard-thresholding, like in [8, 13, 21, 35, 36, 37, 38, 43, 50]. Typically, PROPACK from [31] is used for computing such a truncated SVD. The use of sparse matrix-vector products and the potential convergence issues frequently make this the computationally most expensive part of all these algorithms.

On the other hand, our algorithm first projects any sparse matrix onto the tangent space and then computes the dominant singular vectors for a small $2k$-by-$2k$ matrix. Apart from the application of P_Ω and a *few* sparse matrix-vector products, the rest of the algorithm consists solely of dense linear algebra operations. This is more robust and significantly faster than computing the SVD of a sparse matrix in each step. To the best of our knowledge, LMAFit [53] and other manifold-related algorithms

like [5, 18, 27, 28, 39] are the only competitive solvers where mostly dense linear algebra together with a few sparse matrix-vector products are sufficient.

Due to the difficulty of estimating practical flop counts for algorithms based on computing a sparse SVD, we will only compare the computational complexity of our method with that of LMAFit. An estimate similar to the one of above reveals that LMAFit has a computational cost of

$$t^{\text{LMAFit}}_{\text{practical}} = (18 + C_{\text{sparse}}\, 12\,\text{OS})nk^2, \quad C_{\text{sparse}} \simeq 5, \tag{3.6}$$

where the sparse manipulations involve $2k$ sparse matrix vector multiplications and one application of P_Ω to a rank k matrix. Comparing this estimate to (3.5), LMAFit should be about two times faster per iteration when $\text{OS} = 3$. As we will see later (Tables 5.1 and 5.2) this is almost exactly what we observe experimentally. Since LRGeomCG is more costly per iteration, it will have to converge much faster in order to compete with LMAFit. This is studied in detail in section 5.

4. Convergence for a modified cost function. In this section, we show the convergence of Algorithm 1 applied to a *modified* cost function.

4.1. Reconstruction on the tangent space. As the first step, we apply the general convergence theory for Riemannian optimization from [2].

PROPOSITION 4.1. *Let $\{X_i\}$ be an infinite sequence of iterates generated by Algorithm* 1. *Then, every accumulation point X_* of $\{X_i\}$ satisfies $\mathrm{P}_{T_{X_*}\mathcal{M}_k}\mathrm{P}_\Omega(X_*) = \mathrm{P}_{T_{X_*}\mathcal{M}_k}\mathrm{P}_\Omega(A)$.*

Proof. Since R_X is a smooth retraction, the Armijo-type line search together with the gradient-related search directions, allows us to conclude that any limit point of Algorithm 1 is a critical point by Theorem 4.3.1 in [2]. These critical points X_* are determined by $\operatorname{grad} f(X) = 0$. By (2.6) this gives $\mathrm{P}_{T_{X_*}\mathcal{M}_k}\mathrm{P}_\Omega(X_* - A) = 0$. ∎

The next step is establishing that there exist limit points. However, since the closure of $\mathcal{M}_k$ are the matrices of rank *bounded* by k, a sequence in $\mathcal{M}_k$ may have a limit point that is a matrix of rank lower than k, which is not in $\mathcal{M}_k$. In addition, the objective function $f(X)$ is not coercive because P_Ω has a nontrivial kernel.

These two observations, motivate our choice to prove convergence when Algorithm 1 is applied not to f but a regularized version, namely,

$$g\colon \mathcal{M} \to \mathbb{R},\ X \mapsto f(X) + \mu^2(\|X^\dagger\|_{\mathrm{F}}^2 + \|X\|_{\mathrm{F}}^2), \quad \mu > 0.$$

In g, the term $\|X^\dagger\|_{\mathrm{F}}^2$ will act as barrier to lower-rank matrices, while $\|X\|_{\mathrm{F}}^2$ makes it coercive. We will show below that as a consequence, all iterates $\{X_i\}$ stay in a compact subset of $\mathcal{M}_k$. In addition, $\operatorname{grad} g(X)$ can be computed very cheaply after modifying step 2 in Algorithm 2: Since every $X \in \mathcal{M}_k$ is of rank k, the pseudoinverse is smooth on $\mathcal{M}_k$ and hence $g(X)$ is also smooth. Using [23, Theorem 4.3], one can then show that the gradient of $X \mapsto \|X^\dagger\|_{\mathrm{F}}^2 + \|X\|_{\mathrm{F}}^2$ equals $2U(\Sigma - \Sigma^{-3})V^T$ for $X = U\Sigma V^T$.

Before continuing, we hasten to point out that although optimizing g using Algorithm 1 can be done at virtually no extra cost, there is actually no need for g in a practical algorithm for matrix completion; see Remark 4.3. As such our convergence result is artificial, but it suits our need without formally proving convergence of this more practical scheme, which would be quite technical.

PROPOSITION 4.2. *Let $\{X_i\}$ be an infinite sequence of iterates generated by Algorithm* 1 *but with the objective function $g(X) = f(X) + \mu^2(\|X^\dagger\|_F^2 + \|X\|_F^2)$ for some $0 < \mu < 1$. Then, $\lim_{i\to\infty}\mathrm{P}_{T_{X_i}\mathcal{M}_k}\mathrm{P}_\Omega(X_i - A) = \lim_{i\to\infty} 2\mu^2(X_i^\dagger + X_i)$.*

Proof. We will first show that the iterates stay in a closed and bounded subset of $\mathcal{M}_k$. Let $L = \{X \in \mathcal{M}_k : g(X) \leq g(X_0)\}$ be the level set at X_0. By construction of the line search, all elements of $\{X_i\}$ stay inside L and we get

$$\frac{1}{2}\|\operatorname{P}_\Omega(X_i - A)\|_{\mathrm{F}}^2 + \mu^2\|X_i^\dagger\|_{\mathrm{F}}^2 + \mu^2\|X_i\|_{\mathrm{F}}^2 \leq C_0^2, \quad i > 0,$$

where $C_0^2 := g(X_0)$. This implies

$$\mu^2\|X_i\|_{\mathrm{F}}^2 \leq C_0^2 - \frac{1}{2}\|\operatorname{P}_\Omega(X_i - A)\|_{\mathrm{F}}^2 - \mu^2\|X_i^\dagger\|_{\mathrm{F}}^2 \leq C_0^2,$$

and we obtain an upper bound on the largest singular value:

$$\sigma_1(X_i) \leq \|X_i\|_{\mathrm{F}} \leq C_0/\mu =: C^\sigma, \quad i > 0.$$

Similarly, we get a lower bound on the smallest singular value:

$$\mu^2\|X_i^\dagger\|_{\mathrm{F}}^2 = \sum_{j=1}^{k} \frac{\mu^2}{\sigma_j^2(X_i)} \leq C_0^2 - \frac{1}{2}\|\operatorname{P}_\Omega(X_i - A)\|_{\mathrm{F}}^2 - \mu^2\|X_i\|_{\mathrm{F}}^2 \leq C_0^2, \quad i > 0,$$

which implies that

$$\sigma_k(X_i) \geq \mu/C_0 =: C_\sigma, \quad i > 0.$$

It is clear that all elements of $\{X_i\}$ stay inside the set

$$B = \{X \in \mathcal{M}_k : \sigma_1(X) \leq C^\sigma,\ \sigma_k(X) \geq C_\sigma\}.$$

This set is closed and bounded, hence compact.

Next, suppose that $\lim_{i\to\infty} \|\operatorname{grad} g(X_i)\|_{\mathrm{F}} \neq 0$. Then there is a subsequence in $\{X_i\}_{i\in\mathcal{K}}$ such that $\|\operatorname{grad} g(X)\|_{\mathrm{F}} \geq \varepsilon > 0$ for all $i \in \mathcal{K}$. Since $X_i \in B$, this subsequence $\{X_i\}_{i\in\mathcal{K}}$ has a limit point X_* in B. By continuity of $\operatorname{grad} g$, this implies that $\|\operatorname{grad} g(X_*)\|_{\mathrm{F}} \geq \varepsilon$ which contradicts Theorem 4.3.1 in [2] that every accumulation point is a critical point of g. Hence, $\lim_{i\to\infty} \|\operatorname{grad} g(X_i)\|_{\mathrm{F}} = 0$. ∎

Remark 4.3. Proposition 4.2 is theoretical and μ^2 can be chosen arbitrarily small, for example, as small as $\varepsilon_{\mathrm{mach}} \simeq 10^{-16}$. This means that as long as $\sigma_1(X_i) = O(1)$ and $\sigma_k(X_i) \gg \varepsilon_{\mathrm{mach}}$, the regularization terms in g are negligible and one might as well have optimized the original objective function f, that is, g with $\mu = 0$. This is what we observed in all numerical experiments when $\operatorname{rank}(A) \geq k$. In case $\operatorname{rank}(A) < k$, it is obvious that now $\sigma_k(X_i) \to 0$ as $i \to \infty$. Theoretically one can still optimize g, thereby forcing the optimization problem to stay on $\mathcal{M}_k$, but in practice it makes more sense to transfer the original optimization problem to the manifold of rank $k-1$ matrices. This can be accomplished by monitoring $\sigma_k(X_i)$ throughout the iteration. Since these rank drops do not occur in our experiments, proving convergence of such a method is beyond the scope of this paper.

A similar argument appears in the analysis of other methods for rank-constrained optimization too. For example, the proof of convergence in [12] for SDPLR [11] uses a regularization by $\mu \det(\Sigma)$ with Σ containing the nonzero singular values of X; yet in practice, ones optimizes without this regularization using the observation that μ can be made arbitrarily small, and in practice one always observes convergence when $\mu = 0$. Analogous modifications also appear in [8, 9, 28].

4.2. Reconstruction on the whole space. Based on Proposition 4.2, we have that for a suitable choice of μ, any limit point satisfies

$$\| \mathrm{P}_{T_{X_*}\mathcal{M}_k} \mathrm{P}_\Omega(X_* - A)\|_\mathrm{F} \leq \varepsilon, \quad \varepsilon > 0.$$

In other words, we will have approximately fitted X to the data A on the image of $\mathrm{P}_{T_{X_*}\mathcal{M}_k} \mathrm{P}_\Omega$, and, hopefully, this is sufficient to have $X_* = A$ on the whole space $\mathbb{R}^{m\times n}$. Without further assumptions on X_* and A, however, it is not reasonable to expect that X_* equals A since P_Ω usually has a very large null space.

Let us split the error $E := X - A$ into the three quantities,

$$E_1 = \mathrm{P}_{T_X\mathcal{M}_k} \mathrm{P}_\Omega(X - A), \quad E_2 = \mathrm{P}_{N_X\mathcal{M}_k} \mathrm{P}_\Omega(X - A), \quad \text{and} \quad E_3 = \mathrm{P}_\Omega^\perp(X - A),$$

where $N_X\mathcal{M}_k$ is the normal space of X, i.e., the subspace of all matrices $Z \in \mathbb{R}^{m\times n}$ orthogonal to $T_X\mathcal{M}_k$.

Upon convergence of Algorithm 1, $\|E_1\|_\mathrm{F} \to \varepsilon$, but how big can E_2 be? In general, E_2 can be arbitrarily large since it is the Lagrange multiplier of the fixed-rank constraint of X. In fact, when the rank of X is smaller than the exact rank of A, E_2 typically never vanishes, even though E_1 can converge to zero. On the other hand, when the ranks of X and A are the same, one typically observes that $\|E_1\|_\mathrm{F} \to 0$ implies $\|E_1\|_\mathrm{F} \to 0$. This can also be easily checked during the course of the algorithm, since $\mathrm{P}_\Omega(X - A) = E_1 + E_2$ is computed explicitly for the computation of the gradient in Algorithm 2.

Suppose now that $\|E_1\|_\mathrm{F} \leq \varepsilon$ and $\|E_2\|_\mathrm{F} \leq \tau$, which means that X is in good agreement with A on Ω. The hope is that X and A agree on the complement of Ω too. This is of course not always the case, but it is exactly the crucial assumption of low-rank matrix completion that the observations on Ω are sufficient to complete A. To quantify this assumption, one can make use of standard theory in matrix completion literature; see, e.g., [15, 16, 27]. Since it is not the purpose of this paper to analyze the convergence properties of Algorithm 1 for various, rather theoretical random models, and we do not rely on this theory in the numerical experiments later on, we omit this discussion.

5. Numerical experiments. The implementation of LRGeomCG, i.e., Algorithm 1, was done in MATLAB R2012a on a desktop computer with a 3.10 GHz CPU and 8 GB of memory. All reported times are wall-clock time that include the setup phase of the solvers (if needed) but exclude setting up the (random) problem.

As is mostly done in the literature, we focus only on square matrices. Note however that the performance of most matrix completion solvers behaves very differently for highly nonsquare matrices, as observed in [8]. In our numerical experiments below we only compare with LMAFit of [53]. Extensive comparison with the other approaches from [13, 21, 35, 36, 37, 38, 50] revealed that LMAFit is overall the fastest method for square matrices. We refer to [42] for these results.

The implementation of LMAFit was the one provided by the authors.[1] All options were kept the same, except rank adaptivity was turned off. In sections 5.4 and 5.5 we also changed the stopping criteria as explained there. In order to have a fair comparison between LMAFit and LRGeomCG, the operation P_Ω was performed by the same MATLAB function `part_XY.m` in both solvers. In addition, the initial guesses were taken as random rank k matrices (more on this below).

[1] The version used was downloaded August 14, 2012 from http://lmafit.blogs.rice.edu/.

In all experiments, we will complete random low-rank matrices which are generated as proposed in [13]. First, construct two random matrices $A_L, A_R \in \mathbb{R}^{n\times k}$ with i.i.d. standard Gaussian entries; then, assemble $A := A_L A_R^T$; finally, the set of observations Ω is sampled uniformly at random among all sets of cardinality $|\Omega|$. The resulting observed matrix to complete is now $A_\Omega := \mathrm{P}_\Omega(A)$. Standard random matrix theory asserts that $\|A\|_\mathrm{F} \simeq n\sqrt{k}$ and $\|A_\Omega\|_\mathrm{F} \simeq \sqrt{|\Omega|k}$. In the later experiments, the test matrices are different and their construction will be explained there.

When we report on the relative error of an approximation X, we mean the error

$$\text{relative error} = \|X - A\|_\mathrm{F}/\|A\|_\mathrm{F} \tag{5.1}$$

computed on all the entries of A. On the other hand, the algorithms will compute a relative residual on Ω only:

$$\text{relative residual} = \|\,\mathrm{P}_\Omega(X - A)\|_\mathrm{F}/\|\,\mathrm{P}_\Omega(A)\|_\mathrm{F}. \tag{5.2}$$

Unless stated otherwise, we set as tolerance for the solvers a relative residual of 10^{-12}. Such a high precision is necessary to study the asymptotic convergence rate of the solvers, but where appropriate we will draw conclusions for moderate precisions too.

As initial guess X_1 for Algorithm 1, we construct a random rank-k matrix following the same procedure as above for M. The OS for a rank-k matrix is defined as the ratio of the number of samples to the degrees of freedom in a nonsymmetric matrix of rank k,

$$\mathrm{OS} = |\Omega|/(k(2n - k)). \tag{5.3}$$

Obviously one needs at least $\mathrm{OS} \geq 1$ to uniquely complete any rank-k matrix. In the experiments below, we observe that regardless of the method, $\mathrm{OS} > 2$ is needed to reliably recover an incoherent matrix of rank k after uniform sampling. In addition, each row and each column needs to be sampled at least once. By the coupon collector's problem, this requires that $|\Omega| > Cn\log(n)$, but in all experiments below, Ω is always sufficiently large such that each row and column is sampled at least once. Hence, we will only focus on the quantity OS.

5.1. Influence of size and rank for fixed oversampling. As the first test, we complete random matrices A of exact rank k with LRGeomCG and LMAFit and we explicitly exploit the knowledge of the rank in the algorithms. The purpose of this test is to investigate the dependence of the algorithms on the size and the rank of the matrices. In section 5.3, we will study the influence of OS, but for now we fix the oversampling to $\mathrm{OS} = 3$. In Table 5.1, we report on the mean run time and the number of iterations taking over 10 random instances. The run time and the iteration count of all these instances are visualized in Figures 5.1 and 5.2, respectively.

Overall, LRGeomCG needs less iterations than LMAFit for all problems. In Figure 5.1, we see that this is because LRGeomCG converges faster asymptotically although the convergence of LRGeomCG exhibits a slower transient behavior in the beginning. This phase is, however, only limited to the first 20 iterations. Since the cost per iteration for LMAFit is cheaper than for LRGeomCG (see the estimates (3.5) and (3.6)), this transient phase leads to a trade-off between runtime and precision. This is clearly visible in the timings of Figure 5.2: When sufficiently high accuracy is requested, LRGeomCG is always faster, whereas for low precision, LMAFit is faster.

Let us now check in more detail the dependence on the size n. It is clear from the left sides of Table 5.1 and Figure 5.1 that the iteration counts of LMAFit and

TABLE 5.1
The mean of the computational results for Figures 5.1–5.2 for solving with a tolerance of 10^{-12}.

Fixed rank $k = 40$					Fixed size $n = 8000$				
n	LMAFit		LRGeomCG			LMAFit		LRGeomCG	
	Time(s.)	#its.	Time(s.)	#its.	k	Time(s.)	#its.	Time(s.)	#its.
1000	3.11	133	2.74	54.5	10	17.6	518	8.81	121
2000	9.13	175	6.43	61.3	20	23.9	297	14.6	86.5
4000	21.2	191	14.8	66.7	30	40.9	232	25.8	76.1
8000	53.3	211	36.5	71.7	40	52.7	211	35.8	71.7
16000	150	222	99.1	75.4	50	76.1	194	51.2	67.7
32000	383	233	254	79.1	60	96.6	186	67.0	66.1

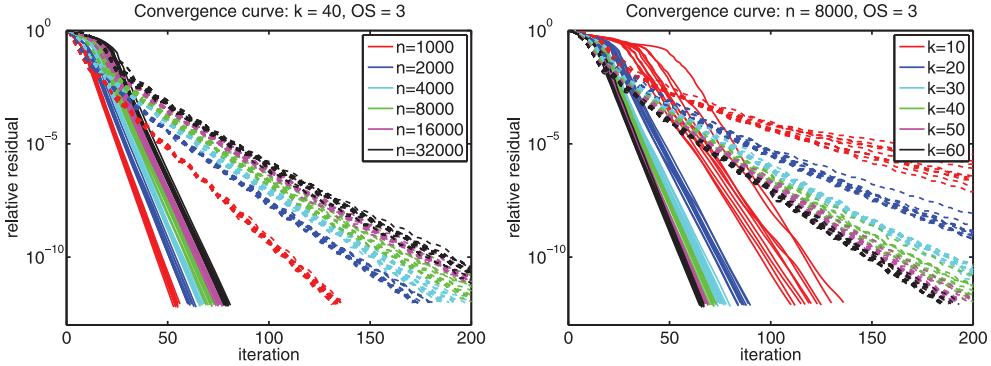

FIG. 5.1. *Convergence curves for LMAFit (dashed line) and LRGeomCG (full line) for fixed oversampling of* $OS = 3$. *Left: variable size* n *and fixed rank* $k = 40$; *right: variable ranks* k *and fixed size* $n = 8000$.

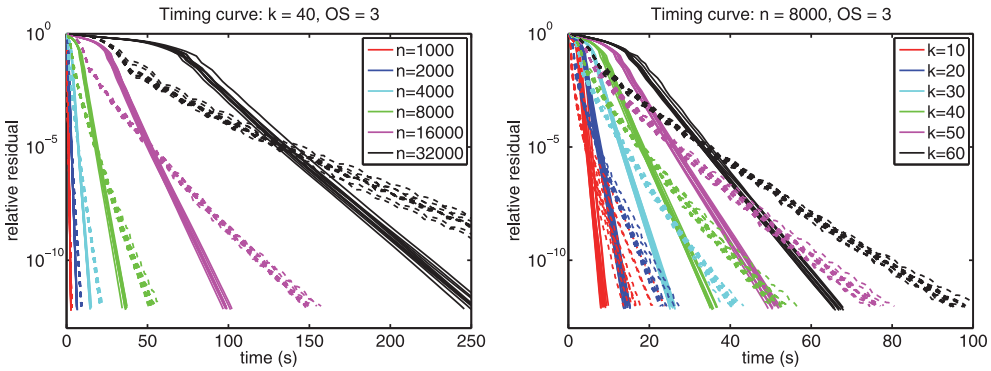

FIG. 5.2. *Timing curves for LMAFit (dashed line) and LRGeomCG (full line) for fixed oversampling of* $OS = 3$. *Left: variable size* n *and fixed rank* $k = 40$; *right: variable ranks* k *and fixed size* $n = 8000$.

LRGeomCG grow with n but seem to stagnate as $n \to \infty$. In addition, LMAFit needs about three times more iterations than LRGeomCG. Hence, even though each iteration of LRGeomCG is about two times more expensive than one iteration of LMAFit, LRGeomCG will eventually always be faster than LMAFit for sufficiently

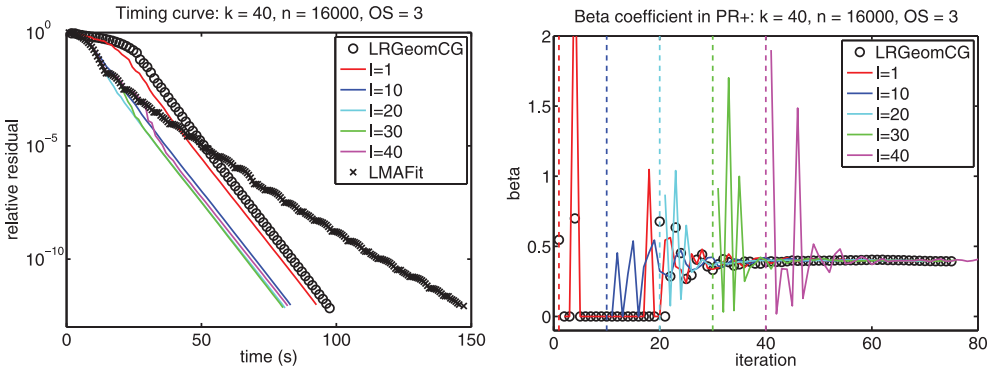

FIG. 5.3. *Timing curve and β coefficient of the hybrid strategy for different values of I.*

high precision. In Figure 5.2 on the left, one can, for example, see that this trade-off point happens at a precision of about 10^{-5}. For growing rank k, the conclusion of the same analysis is very much the same.

The previous conclusion depends critically on the convergence factor and, as we will see later on, this factor is determined by the amount of oversampling (OS). But for difficult problems (OS = 3 is near the lower limit of 2), we can already observe that LRGeomCG is about 50% faster than LMAFit.

5.2. Hybrid strategy. The experimental results from above clearly show that the transient behavior of LRGeomCG's convergence is detrimental if only modest accuracy is required. The reason for this slow phase is the lack of nonlinear CG acceleration (β in Algorithm 4 is almost always zero) which essentially reduces LRGeomCG to a steepest descent algorithm. On the other hand, LMAFit is much less affected by slow convergence during the initial phase.

A simple heuristic to overcome this bad phase is a hybrid solver: For the first I iterations, we use LMAFit; after that, we hope that the nonlinear CG acceleration kicks in immediately and we can efficiently solve with LRGeomCG. In Figure 5.3, we have tested this strategy for different values of I on a problem of the previous section with $n = 16000$ and $k = 40$.

From the left panel of Figure 5.3, we get that the run time to solve for a tolerance of 10^{-12} is reduced from 97 sec (LRGeomCG) and 147 sec (LMAFit) to about 80 sec for the choices $I = 10, 20, 30, 40$. Performing only one iteration of LMAFit is less effective. For $I = 20$, the hybrid strategy has almost no transient behavior anymore and it is always faster than LRGeomCG or LMAFit alone. Furthermore, the choice of I is not very sensitive to the performance as long as it is between 10 and 40, and already for $I = 10$, there is a significant speedup of LRGeomCG noticeable for all tolerances.

The quantity β of PR+ in Algorithm 4 is plotted in the right panel of Figure 5.3. Until the 20th iteration, plain LRGeomCG shows no meaningful CG acceleration. For the hybrid strategies with $I > 10$, the acceleration kicks in almost immediately. Observe that all approaches converge to $\beta \simeq 0.4$.

While this experiment shows that there is potential to speed up LRGeomCG in the early phase, we do not wish to claim that the present strategy is very robust or directly applicable. The main point that we wish to make, however, is that the slow phase can be avoided in a relatively straightforward way and that LRGeomCG can be

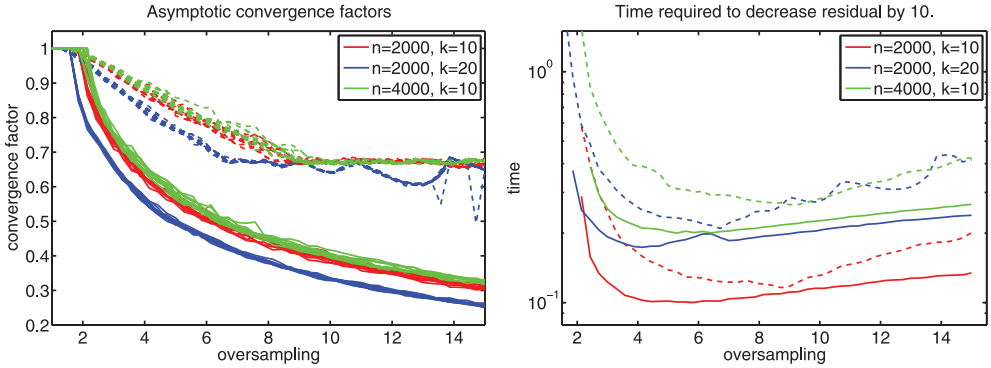

FIG. 5.4. *Asymptotic convergence factor ρ (left) and mean time to decrease the error by a factor of* 10 *(right) for LMAFit (dashed line) and LRGeomCG (full line) in function of the OS.*

warm started by any other method. In particular, this shows that LRGeomCG can be efficiently employed once the correct rank of the solution X is identified by some other method. As explained in the introduction, there are numerous other methods for low-rank matrix completion that can reliably identify this rank but have a slow asymptotic convergence. Combining them with LRGeomCG seems like a promising way forward.

5.3. Influence of oversampling. Next, we investigate the influence of oversampling on the convergence speed. We take 10 random problems with fixed rank and size, but vary the OS using 50 values between $1, \ldots, 15$. In Figure 5.4, on the left, the asymptotic convergence factor ρ is visible for LMAFit and LRGeomCG for three different combinations of the size and rank. Since the convergence is linear and we want to filter out any transient behavior, this factor was computed as

$$\rho = \left(\frac{\| \mathrm{P}_\Omega(X_{i_{\mathrm{end}}} - A)\|_{\mathrm{F}}}{\| \mathrm{P}_\Omega(X_{10} - A)\|_{\mathrm{F}}} \right)^{1/(i_{\mathrm{end}}-10)},$$

where i_{end} indicates the last iteration. A factor of one indicates failure to converge within 4000 steps. One can observe that all methods become slower as OS $\to 2$. Further, the convergence factor of LMAFit seems to stagnate for large values of OS while LRGeomCG's actually becomes better. Since for growing OS the completion problems become easier as more entries are available, only LRGeomCG shows the expected behavior. In contrast to our parameter-free choice of nonlinear CG, LMAFit needs to determine and adjust a certain acceleration factor dynamically based on the performance of the iteration. We believe the stagnation in Figure 5.4 on the left is due to a suboptimal choice of this factor in LMAFit.

In the right panel of Figure 5.4, the mean time to decrease the relative residual by a factor of 10 is displayed. These timings were again determined by neglecting the first 10 iterations and then interpolating the time needed for a reduction of the residual by 10. Similarly as before, we can observe that since the cost per iteration is cheaper for LMAFit, there is a range for OS around $7, \ldots, 9$ where LMAFit is only slightly slower than LRGeomCG. However, for smaller and larger values of OS, LRGeomCG is always faster by a significant margin (observe the logarithmic scale).

5.4. Influence of noise. Next, we investigate the influence of noise by adding random perturbations to the rank-k matrix A. We define the noisy matrix $A^{(\varepsilon)}$ with

noise level ε as

$$A^{(\varepsilon)} := A + \varepsilon \frac{\|A_\Omega\|_{\mathrm{F}}}{\|N_\Omega\|_{\mathrm{F}}} N,$$

where N is a standard Gaussian matrix and Ω is the usual random sampling set. (See also [50, 53] for a similar setup.) The reason for defining $A^{(\varepsilon)}$ in this way is that we have

$$\|\,\mathrm{P}_\Omega(A - A^{(\varepsilon)})\|_{\mathrm{F}} = \varepsilon \frac{\|A_\Omega\|_{\mathrm{F}}}{\|N_\Omega\|_{\mathrm{F}}} \|\,\mathrm{P}_\Omega(N)\|_{\mathrm{F}} \simeq \varepsilon \sqrt{|\Omega| k}.$$

So, the best relative residual we may expect from an approximation $X^{\mathrm{opt}} \simeq A$ is

$$\|\,\mathrm{P}_\Omega(X^{\mathrm{opt}} - A^{(\varepsilon)})\|_{\mathrm{F}} / \|\,\mathrm{P}_\Omega(A)\|_{\mathrm{F}} \simeq \|\,\mathrm{P}_\Omega(A - A^{(\varepsilon)})\|_{\mathrm{F}} / \|\,\mathrm{P}_\Omega(A)\|_{\mathrm{F}} \simeq \varepsilon. \tag{5.4}$$

Similarly, the best relative error of X^{opt} should be on the order of the noise ratio too since

$$\|A - A^{(\varepsilon)}\|_{\mathrm{F}} / \|A\|_{\mathrm{F}} \simeq \varepsilon. \tag{5.5}$$

Due to the presence of noise, the relative residual (5.1) cannot go to zero but the Riemannian gradient of our optimization problem can. In principle, this suffices to detect convergence of LRGeomCG. In practice, however, we have noticed that this wastes a lot of iterations in case the iteration stagnates. A simply remedy is to monitor

$$\text{relative change at step } i = \left|1 - \sqrt{f(X_i)/f(X_{i-1})}\right| \tag{5.6}$$

and stop the iteration when this value drops below a certain threshold. After some experimentation, we fixed this threshold to 10^{-3}. Such a stagnation detection is applied by most other low-rank completion solvers [50, 53], although its specific form varies. Our choice coincides with the one from [53], but we have lowered the threshold.

After equipping LMAFit and LRGeomCG with this stagnation detection, we can display the experimental results in Figure 5.5 and Table 5.2. We have only reported on one choice for the rank, size, and oversampling since the same conclusions can be reached for any other choice. Based on Figure 5.5, it is clear that the stagnation is effectively detected by both methods, except for the very noisy case of $\varepsilon = 1$. (Recall that we changed the original stagnation detection procedure in LMAFit to ours of (5.6) to be able to draw a fair comparison.)

Comparing the results with those of the previous section, the only difference is that the iteration stagnates when the error reaches the noise level. In particular, the iterations are undisturbed by the presence of the noise up until the very last iterations. In Table 5.2, one can observe that both methods solve all problems up to the noise level, which is the best that can be expected from (5.4) and (5.5). Again, LRGeomCG is faster when a higher accuracy is required, which, in this case, corresponds to small noise levels. Since the iteration is unaffected by the noise until the very end, the hybrid strategy of the previous section should be effective here too.

5.5. Exponentially decaying singular values. In the previous problems, the rank of the matrices could be unambiguously defined. Even in the noisy case, there was still a sufficiently large gap between the original nonzero singular values and the

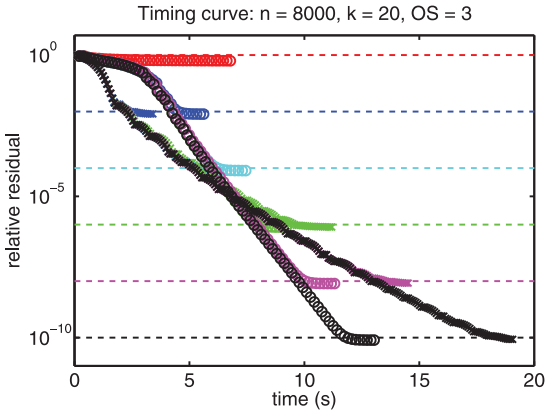

FIG. 5.5. *Timing curves for LMAFit (crosses) and LRGeomCG (circles) for different noise levels with size $n = 8000$, rank $k = 20$, and fixed oversampling of $OS = 3$. The noise levels ε are indicated by dashed lines in different color.*

TABLE 5.2
Computational results for Figure 5.5.

ε	LMAFit				LRGeomCG			
	Time(s.)	#its.	Error	Residual	Time(s.)	#its.	Error	Residual
10^{-0}	2.6	31	$1.04\,\varepsilon$	$0.59\,\varepsilon$	6.8	42	$1.52\,\varepsilon$	$0.63\,\varepsilon$
10^{-2}	3.4	42	$0.77\,\varepsilon$	$0.83\,\varepsilon$	5.6	36	$0.72\,\varepsilon$	$0.82\,\varepsilon$
10^{-4}	6.4	79	$0.83\,\varepsilon$	$0.83\,\varepsilon$	7.5	46	$0.72\,\varepsilon$	$0.82\,\varepsilon$
10^{-6}	11	133	$0.74\,\varepsilon$	$0.82\,\varepsilon$	9.2	58	$0.72\,\varepsilon$	$0.82\,\varepsilon$
10^{-8}	15	179	$0.96\,\varepsilon$	$0.85\,\varepsilon$	11	70	$0.72\,\varepsilon$	$0.82\,\varepsilon$
10^{-10}	19	235	$1.09\,\varepsilon$	$0.87\,\varepsilon$	13	81	$0.72\,\varepsilon$	$0.82\,\varepsilon$

ones affected by noise. In this section, we will complete a matrix for which the singular values decay but do not become exactly zero.

Although a different setting than the previous problems, this type of matrix occurs frequently in the context of approximation of discretized functions or solutions of PDEs on tensorized domains; see, e.g., [30]. In this case, the problem of approximating discretized functions by low-rank matrices is primarily motivated by reducing the amount of data to store. On a continuous level, low-rank approximation corresponds in this setting to approximation by sums of separable functions. As such, one is typically interested in approximating the matrix up to a certain tolerance using the lowest rank possible.

The (exact) rank of a matrix is now better replaced by the numerical rank, or ε-rank [22, Chapter 2.5.5]. Depending on the required accuracy ε, the ε-rank is the quantity

$$k_\varepsilon(A) = \min_{\|A-B\|_2 \leq \varepsilon} \operatorname{rank}(B). \tag{5.7}$$

Obviously, when $\varepsilon = 0$, one recovers the (exact) rank of a matrix. It is well known that the ε-rank of $A \in \mathbb{R}^{m\times n}$ can be determined from the singular values σ_i of A as follows

$$\sigma_1 \geq \sigma_{k_\varepsilon} > \varepsilon \geq \sigma_{k_\varepsilon+1} \geq \cdots \geq \sigma_p, \quad p = \min(m,n).$$

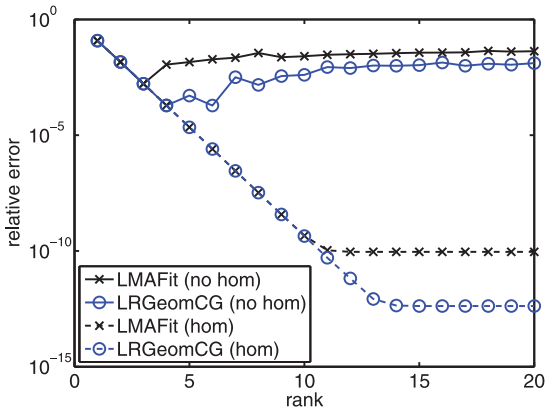

FIG. 5.6. *Relative error of LMAFit and LRGeomCG after completion of a discretization of* (5.8) *for* $OS = 8$ *with and without a homotopy strategy.*

In the context of the approximation of bivariate functions, the relation (5.7) is very useful in the following way. Let $f(x, y)$ be a function defined on a tensorized domain $(x, y) \in \mathcal{X} \times \mathcal{Y}$. After discretization of x and y, we arrive at a matrix A. If we allow for an absolute error in the spectral norm of the size ε, then (5.7) tells us that we can approximate M by a rank k_ε matrix. For example, we take the following simple bivariate function to complete:

$$f(x, y) = \frac{1}{1 + \|x - y\|_2^2}, \qquad (x, y) \in [0, 1]^2. \tag{5.8}$$

Such functions occur as two-point correlations related to second-order elliptic problems with stochastic source terms; see [24].

We ran LRGeomCG and LMAFit after uniformly discretizing x and y in (5.8) resulting in a matrix of size $n = 8000$ and then sampling this matrix with OS $= 8$ (calculated for rank 20). Since a stopping condition on the residual has no meaning for this example, we only used a tolerance of 10^{-3} on the relative change (5.6) or a maximum of 500 iterations. In Figure 5.6, the final accuracy for different choices of the rank is visible for these two problem sets.

The results labeled "no hom" use the standard strategy of starting with random initial guesses for each value of the rank k. Clearly, this results in a very unsatisfactory performance of LRGeomCG and LMAFit since the error of the completion does not decrease with increasing rank. Remark that we have measured the relative error as

$$\|\operatorname{P}_\Gamma(X_* - A)\|_{\mathrm{F}} / \|\operatorname{P}_\Gamma(A)\|_{\mathrm{F}},$$

where Γ is a random sampling set different from Ω but equally large.

The previous behavior is a clear example that the local optimizers of LMAFit and LRGeomCG are very far away from the global ones. However, there is a straightforward solution to this problem: Instead of taking a random initial guess for each k, we only take a random initial guess for $k = 1$. For all other $k > 1$, we use $X = U\Sigma V^T$, where

$$U = \begin{bmatrix} \widetilde{U} & u \end{bmatrix}, \quad \Sigma = \begin{bmatrix} \widetilde{\Sigma} & 0 \\ 0 & \widetilde{\Sigma}_{k-1,k-1} \end{bmatrix}, \quad V = \begin{bmatrix} \widetilde{V} & v \end{bmatrix},$$

$\widetilde{X} = \widetilde{U}\widetilde{\Sigma}\widetilde{V}^T$ is the local optimizer for rank $k-1$, and u and v are random unit-norm vectors orthogonal to U and V, respectively. We call this the *homotopy strategy* which is also used in one of LMAFit's rank adaptivity strategies (but with zero vectors u and v). The error using this homotopy strategy is labeled "hom" in Figure 5.6 and it clearly performs much more favorably. In addition, LRGeomCG is able to obtain a smaller error.

6. Conclusions. The matrix completion consists of recovering a low-rank matrix based on a very sparse set of entries of this matrix. In this paper, we have presented a new method based on optimization on manifolds to solve large-scale matrix completion problems. Compared to most other existing methods in the literature, our approach consists of directly minimizing the least-square error of the fit directly over $\mathcal{M}_k$, the set of matrices of rank k.

The main contribution of this paper is to show that the lack of vector space structure of $\mathcal{M}_k$ does not need be an issue when optimizing over this set and to illustrate this for low-rank matrix completion. Using the framework of retraction-based optimization, the necessary expressions were derived in order to minimize any smooth objective function over the Riemannian manifold $\mathcal{M}_k$. The geometry chosen for $\mathcal{M}_k$, namely, a submanifold embedded in the space of matrices, allowed for an efficient implementation of nonlinear CG for matrix completion. Indeed, the numerical experiments illustrated that this approach outperforms state-of-the-art solvers for matrix completion. In particular, the method achieved very fast asymptotic convergence factors without any tuning of parameters thanks to a simple computation of the initial guess for the line search.

A drawback of the proposed approach is, however, that the rank of the manifold is fixed. Although there are several problems where choosing the rank is straightforward, an integrated manifold-based solution, like in [17, 43], is desirable. In addition, our convergence proof relied on several safeguards and modifications in the algorithm that seem completely unnecessary in practice. Closing these gaps are currently topics of further research.

Acknowledgments. Parts of this paper have been prepared while the author was affiliated with the Seminar of Applied Mathematics, ETH Zürich. In addition, he gratefully acknowledges the helpful discussions with Daniel Kressner regarding low-rank matrix completion.

REFERENCES

[1] P.-A. Absil, C. G. Baker, and K. A. Gallivan, *Trust-region methods on Riemannian manifolds*, Found. Comput. Math., 7 (2007), pp. 303–330.
[2] P.-A. Absil, R. Mahony, and R. Sepulchre, *Optimization Algorithms on Matrix Manifolds*, Princeton University Press, Princeton, NJ, 2008.
[3] P.-A. Absil and J. Malick, *Projection-like retractions on matrix manifolds*, SIAM J. Optim., 22 (2012), pp. 135–158.
[4] R. L. Adler, J.-P. Dedieu, J. Y. Margulies, M. Martens, and M. Shub, *Newton's method on Riemannian manifolds and a geometric model for the human spine*, IMA J. Numer. Anal., 22 (2002), pp. 359–390.
[5] L. Balzano, R. Nowak, and B. Recht, *Online identification and tracking of subspaces from highly incomplete information*, in Proceedings of the Allerton Conference on Communication, Control, and Computing, Momticello, IL, 2010.
[6] J. Bennett and S. Lanning, *The Netflix Prize*, in Proceedings of the KDD-Cup and Workshop at the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Jose, CA, 2007.
[7] D. P. Bertsekas, *Nonlinear Programming*, Athena Scientific, Nashua, NH, 1999.

[8] N. Boumal and P.-A. Absil, *RTRMC: A Riemannian trust-region method for low-rank matrix completion*, in Proceedings of the Neural Information Processing Systems Conference (NIPS), Granada, Spain, 2011.

[9] N. Boumal and P.-A. Absil, *Low-Rank Matrix Completion via Trust-Regions on the Grassmann Manifold*, Technical report 2012.07, Université catholique de Louvain, INMA, Louvain-La-Neuve, Belgium, 2012.

[10] W. Bruns and U. Vetter, *Determinantal rings*, Lecture Notes in Math. 1327, Springer-Verlag, Berlin, 1988.

[11] S. Burer and R. D. C. Monteiro, *A nonlinear programming algorithm for solving semidefinite programs via low-rank factorization*, Math. Program., 95 (2003), pp. 329–357.

[12] S. Burer and R. D. C. Monteiro, *Local minima and convergence in low-rank semidefinite programming*, Math. Program., 103 (2005), pp. 427–444.

[13] J.-F. Cai, E. J. Candès, and Z. Shen, *A singular value thresholding algorithm for matrix completion*, SIAM J. Optim., 20 (2010), pp. 1956–1982.

[14] E. J. Candés and Y. Plan, *Matrix completion with noise*, Proceedings of the IEEE, 98 (2010), pp. 925–936.

[15] E. J. Candès and T. Tao, *The power of convex relaxation: Near-optimal matrix completion*, IEEE Trans. Inform. Theory, 56 (2009), pp. 2053–2080.

[16] E. Candès and B. Recht, *Exact matrix completion via convex optimization*, Found. Comput. Math., 9 (2009), pp. 717–772.

[17] T. P. Cason, P.-A. Absil, and P. Van Dooren, *Iterative methods for low rank approximation of graph similarity matrices*, Linear Algebra Appl., 438 (2013), pp. 1863–1882.

[18] W. Dai, O. Milenkovic, and E. Kerman, *Subspace evolution and transfer (SET) for low-rank matrix completion*, IEEE Trans. Signal Process., 59 (2011), pp. 3120–3132.

[19] W. Dai and O. Milenkovic, *SET: An algorithm for consistent matrix completion*, in International Conference on Acoustics, Speech, and Signal Processing (ICASSP), Dallas, TX, 2010, pp. 3646–3649.

[20] A. Edelman, T. A. Arias, and S. T. Smith, *The geometry of algorithms with orthogonality constraints*, SIAM J. Matrix Anal. Appl., 20 (1999), pp. 303–353.

[21] D. Goldfarb and S. Ma, *Convergence of fixed point continuation algorithms for matrix rank minimization*, Found. Comput. Math., 11 (2011), pp. 183–210.

[22] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 3rd ed., Johns Hopkins Stud. Math. Sci., Baltimore, MD, 1996.

[23] G. H. Golub and V. Pereyra, *The differentiation of pseudo-inverses and nonlinear least squares problems whose variables separate*, SIAM J. Numer. Anal., 10 (1973), pp. 413–432.

[24] H. Harbrecht, M. Peters, and R. Schneider, *On the low-rank approximation by the pivoted Cholesky decomposition*, Appl. Numer. Math., 62 (2012), pp. 428–440.

[25] U. Helmke and J. B. Moore, *Optimization and Dynamical Systems*, Springer-Verlag, London, 1994.

[26] M. Journée, F. Bach, P.-A. Absil, and R. Sepulchre, *Low-rank optimization on the cone of positive semidefinite matrices*, SIAM J. Optim., 20 (2010), pp. 2327–2351.

[27] R. H. Keshavan, A. Montanari, and S. Oh, *Matrix completion from a few entries*, IEEE Trans. Inform. Theory, 56 (2010), pp. 2980–2998.

[28] R. Keshavan, A. Montanari, and S. Oh, *Matrix completion from noisy entries*, J. Mach. Learn. Res., 11 (2010), pp. 2057–2078.

[29] O. Koch and C. Lubich, *Dynamical low-rank approximation*, SIAM J. Matrix Anal. Appl., 29 (2007), pp. 434–454.

[30] D. Kressner and C. Tobler, *Krylov subspace methods for linear systems with tensor product structure*, SIAM J. Matrix Anal. Appl., 31 (2010), pp. 1688–1714.

[31] R. M. Larsen, *PROPACK—Software for Large and Sparse SVD Calculations*. http://soi.stanford.edu/~rmunk/PROPACK (2004).

[32] J. M. Lee, *Introduction to smooth manifolds*, Grad. Texts in Math. 218, Springer-Verlag, New York, 2003.

[33] K. Lee and Y. Bresler, *ADMiRA: Atomic decomposition for minimum rank approximation*, IEEE Trans. Inform. Theory, 56 (2010), pp. 4402–4416.

[34] A. S. Lewis and J. Malick, *Alternating projections on manifolds*, Math. Oper. Res., 33 (2008), pp. 216–234.

[35] Z. Lin, M. Chen, L. Wu, and Yi Ma, *The augmented Lagrange multiplier method for exact recovery of corrupted low-rank matrices*, Technical report UILU-ENG-09-2215, University of Illinois, Urbana, IL, 2009.

[36] Y.-J. Liu, D. Sun, and K.-C. Toh, *An implementable proximal point algorithmic framework for nuclear norm minimization*, Math. Program., 133 (2012), pp. 399–436.

[37] S. Ma, D. Goldfarb, and L. Chen, *Fixed point and Bregman iterative methods for matrix rank minimization*, Math. Program., 128 (2011), pp. 321–353.
[38] R. Meka, P. Jain, and I. S. Dhillon, *Guaranteed rank minimization via singular value projection*, in Proceedings of the Neural Information Processing Systems Conference (NIPS), 2010.
[39] 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 (ICML2011), Bellevue, WA, 2011.
[40] G. Meyer, S. Bonnabel, and R. Sepulchre, *Regression on fixed-rank positive semidefinite matrices: A Riemannian approach*, J. Mach. Learn. Res., 12 (2011), pp. 593–625.
[41] G. Meyer, *Geometric Optimization Algorithms for Linear Regression on Fixed-Rank Matrices*, Ph.D. thesis, University of Liège, Liège, Belgium, 2011.
[42] M. Michenková, *Numerical algorithms for low-rank matrix completion problems*, http://www.math.ethz.ch/~kressner/students/michenkova.pdf (2011).
[43] B. Mishra, G. Meyer, F. Bach, and R. Sepulchre, *Low-Rank Optimization with Trace Norm Penalty*, preprint, 2011, http://arxiv.org/abs/1112.2318.
[44] B. Mishra, G. Meyer, S. Bonnabel, and R. Sepulchre, *Fixed-Rank Matrix Factorizations and Riemannian Low-Rank Optimization*, preprint, 2012, http:arxiv.org/abs/1209.0430.
[45] R. Orsi, U. Helmke, and J. B. Moore, *A Newton-like method for solving rank constrained linear matrix inequalities*, Automatica, 42 (2006), pp. 1875–1882.
[46] C. Qi, K. A. Gallivan, and P.-A. Absil, *Riemannian BFGS algorithm with applications*, in Recent Advances in Optimization and its Applications in Engineering, Springer, to appear.
[47] U. Shalit, D. Weinshall, and G. Chechik, *Online learning in the manifold of low-rank matrices*, in Advances in Neural Inf. Process. Syst. 23, J. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, eds., 2010, pp. 2128–2136.
[48] U. Shalit, D. Weinshall, and G. Chechik, *Online learning in the embedded manifold of low-rank matrices*, J. Mach. Learn. Res., 13 (2012), pp. 429–458.
[49] S. T. Smith, *Optimization techniques on Riemannian manifold*, in Hamiltonian and Gradient Flows, Algorithms and Control, vol. 3, AMS, Providence, RI, 1994, pp. 113–136.
[50] K. Toh and S. Yun, *An accelerated proximal gradient algorithm for nuclear norm regularized least squares problems*, Pacific J. Optimization, 6 (2010), pp. 615–640.
[51] B. Vandereycken and S. Vandewalle, *A Riemannian optimization approach for computing low-rank solutions of Lyapunov equations*, SIAM J. Matrix Anal. Appl., 31 (2010), pp. 2553–2579.
[52] B. Vandereycken, *Low-rank matrix completion by Riemannian optimization (extended version)*, preprint, http://arxiv.org/abs/1209.3834.
[53] Z. Wen, W. Yin, and Yin Zhang, *Solving a low-rank factorization model for matrix completion by a non-linear successive over-relaxation algorithm*, Math. Program. Comput., 4 (2012), pp. 333–361.