Linear Algebra and its Applications 475 (2015) 200–239

Contents lists available at ScienceDirect

Linear Algebra and its Applications

www.elsevier.com/locate/laa

Low-rank matrix completion via preconditioned optimization on the Grassmann manifold

Nicolas Boumal [a,*], P.-A. Absil [b]

[a] *Inria & D.I., UMR 8548, Ecole Normale Supérieure, Paris, France*
[b] *Department of Mathematical Engineering, ICTEAM Institute, Université catholique de Louvain, Belgium*

ARTICLE INFO

Article history:
Received 14 July 2014
Accepted 20 February 2015
Available online xxxx
Submitted by V. Mehrmann

MSC:
90C90
53B21
15A83

Keywords:
Low-rank matrix completion
Optimization on manifolds
Grassmann manifold
Preconditioned Riemannian trust-regions
Preconditioned Riemannian conjugate gradients
Second-order methods
Fixed-rank geometry
RTRMC
RCGMC

ABSTRACT

We address the numerical problem of recovering large matrices of low rank when most of the entries are unknown. We exploit the geometry of the low-rank constraint to recast the problem as an unconstrained optimization problem on a single Grassmann manifold. We then apply second-order Riemannian trust-region methods (RTRMC 2) and Riemannian conjugate gradient methods (RCGMC) to solve it. A preconditioner for the Hessian is introduced that helps control the conditioning of the problem and we detail preconditioned versions of Riemannian optimization algorithms. The cost of each iteration is linear in the number of known entries. The proposed methods are competitive with state-of-the-art algorithms on a wide range of problem instances. In particular, they perform well on rectangular matrices. We further note that second-order and preconditioned methods are well suited to solve badly conditioned matrix completion tasks.

* Corresponding author.
E-mail addresses: nicolasboumal@gmail.com (N. Boumal), absil@inma.ucl.ac.be (P.-A. Absil).

http://dx.doi.org/10.1016/j.laa.2015.02.027
0024-3795/

1. Introduction

This paper considers the problem of recovering a low-rank $m \times n$ matrix X ($m \leq n$) of which a few entries are observed, possibly with noise. Throughout, we assume that $r = \text{rank}(X)$ is known and is much smaller than m, and we let $\Omega \subset \{1 \dots m\} \times \{1 \dots n\}$ be the set of indices of the observed entries of X, that is, X_{ij} is known if (i, j) is in Ω.

It was shown in numerous publications referenced below that low-rank matrix completion is applicable in various situations, notably to build recommender systems. In this setting, the rows of the matrix may correspond to items and the columns may correspond to users. The known entries are the ratings given by users to some items. The aim is to predict the unobserved ratings to generate personalized recommendations, that is, the aim is to complete the matrix. Such applications motivate the study of scalable algorithms given the size of practical instances (billions of entries to predict based on millions of observed entries).

In this paper, we propose algorithms for low-rank matrix completion based on preconditioned Riemannian optimization over a single Grassmannian manifold. We test them on synthetic data under different scenarios. Each scenario challenges the proposed methods and prior art on a different aspect of the problem, such as the size of the problem, the conditioning of the target matrix, the sampling process, etc. Finally, we demonstrate the applicability of the proposed algorithms on the Netflix dataset. These combined experiments exhibit some of the advantages of our approach over competing, state-of-the-art algorithms.

1.1. Related work

In the noiseless case, one could state the minimum rank matrix recovery problem as follows:

$$\min_{\hat{X} \in \mathbb{R}^{m \times n}} \text{rank}\, \hat{X}, \quad \text{such that} \quad \hat{X}_{ij} = X_{ij} \quad \forall (i, j) \in \Omega. \tag{1}$$

This problem, however, is NP-hard [11]. A possible convex proxy for (1) introduced by Candès and Recht [11] is to minimize the nuclear norm of $\hat{X}$ instead of the rank, that is, minimize the sum of its singular values. The SVT method [10] for example attempts to solve such a convex problem using singular value thresholding. The ADMiRA method [25] does so using matching pursuit-like techniques. One important advantage of proceeding with convex relaxations is that the resulting algorithms can be analyzed thoroughly. In this line of work, a number of polynomial-time (but nevertheless quite slow) algorithms have been proven to attain exact recovery in noiseless scenarios and stable recovery in the face of noise, even when the sampling set Ω is very small compared to the total size mn.

In noisy scenarios, one may want to minimize a least-squares data fitting term regularized with a nuclear norm term. For example, NNLS [39] is an accelerated proximal gradient method for nuclear norm-regularized least-squares problems of this kind.

Jellyfish [35] is a stochastic gradient method to solve this type of problems on multiple computers in parallel. They focus on obviating fine-grained locking, which enables them to tackle very large scale problems. Another parallel approach for very large scale matrix completion is the divide-and-conquer scheme by Mackey et al. [27], where the errors introduced by the division step are statistically described and their effect on the global problem is controlled during the conquer step.

As an alternative to (1), one may minimize the discrepancy between $\hat{X}$ and X at entries Ω under the constraint that $\operatorname{rank}(\hat{X}) \leq r$ for some small constant r. Since any matrix $\hat{X}$ of rank at most r can be written in the form UW with $U \in \mathbb{R}^{m\times r}$ and $W \in \mathbb{R}^{r\times n}$, a reasonable formulation of the problem reads:

$$\min_{U\in\mathbb{R}^{m\times r}} \min_{W\in\mathbb{R}^{r\times n}} \sum_{(i,j)\in\Omega} \big((UW)_{ij} - X_{ij}\big)^2. \tag{2}$$

This is also NP-hard [19], but only requires manipulating small matrices. The LMaFit method [42] addresses this problem by alternatively fixing either of the variables and solving the resulting least-squares problem efficiently. Hardt [21] proposes a common framework to analyze the convergence of such methods.

The factorization of a matrix $\hat{X}$ into the product UW is not unique. Indeed, for any $r \times r$ invertible matrix M, we have $UW = (UM)(M^{-1}W)$. All the matrices UM share the same column space. Hence, the optimal value of the inner optimization problem in (2) is a function of $\operatorname{col}(U)$—the column space of U—rather than U specifically. Dai et al. [15,16] exploit this to recast (2) on the Grassmann manifold $\operatorname{Gr}(m, r)$, that is, the set of r-dimensional linear subspaces of $\mathbb{R}^m$ (see Section 2):

$$\min_{\mathscr{U}\in\operatorname{Gr}(m,r)} \min_{W\in\mathbb{R}^{r\times n}} \sum_{(i,j)\in\Omega} \big((UW)_{ij} - X_{ij}\big)^2, \tag{3}$$

where $U \in \mathbb{R}^{m\times r}$ is any matrix such that $\operatorname{col}(U) = \mathscr{U}$ and is often chosen to be orthonormal. Unfortunately, the objective function of the outer minimization in (3) may be discontinuous at points $\mathscr{U}$ for which the least-squares problem in W does not have a unique solution. Dai et al. [15] propose ingenious ways to deal with the discontinuity. Their focus, though, is on deriving theoretical performance guarantees rather than developing fast algorithms. Likewise, Balzano et al. [4] introduce GROUSE, a stochastic gradient descent method for subspace identification, applicable to matrix completion. They also work on a single Grassmannian but with more emphasis on computational efficiency.

Keshavan et al. [24] state the problem on the Grassmannian too, but propose to simultaneously optimize on the row and column spaces, yielding a smaller, largely overdetermined least-squares problem which is likely to have a unique solution, resulting in a smooth objective function. In a related paper [23], they solve:

$$\min_{\mathscr{U}\in\operatorname{Gr}(m,r),\mathscr{V}\in\operatorname{Gr}(n,r)} \min_{S\in\mathbb{R}^{r\times r}} \sum_{(i,j)\in\Omega} \big((USV^\top)_{ij} - X_{ij}\big)^2 + \lambda^2 \left\|USV^\top\right\|_{\mathrm{F}}^2, \tag{4}$$

where U and V are any orthonormal bases of $\mathscr{U}$ and $\mathscr{V}$, respectively, $\|M\|_{\mathrm{F}} = \sqrt{\mathrm{trace}(M^\top M)}$ denotes the Frobenius norm and λ is a regularization parameter. The authors propose an efficient SVD-based initial guess for $\mathscr{U}$ and $\mathscr{V}$ which they refine using a Riemannian steepest descent method [2], along with strong theoretical guarantees. Ngo and Saad [33] exploit this idea further by applying a Riemannian conjugate gradient method to this formulation. They endow the Grassmannians with a preconditioned metric in order to better capture the conditioning of low-rank matrix completion, with excellent results.

Mishra et al. [32] propose another geometric approach. They address the problem of low-rank trace norm minimization and propose an algorithm that alternates between fixed-rank optimization and rank-one updates, with applications to low-rank matrix completion.

Vandereycken [41] investigates a Riemannian conjugate gradient approach based on a submanifold geometry for the manifold of fixed-rank matrices. Meyer et al. [28] and Absil et al. [3] propose a few quotient geometries for the manifold of fixed-rank matrices. In recent work, Mishra et al. [31] endow these geometries with preconditioned metrics, akin to the ones developed simultaneously by Ngo and Saad [33] on a double Grassmannian, and use Riemannian conjugate gradient methods, also with excellent results. Schneider and Uschmajew [37] propose a recent analysis of the related problem of optimization over the variety of matrices of bounded rank, rather than fixed rank. Note that the methods we propose below also handle the constraint $\mathrm{rank}(\hat{X}) \leq r$ rather than $\mathrm{rank}(\hat{X}) = r$. However, we have observed in numerical experiments that the methods perform better when $\mathrm{rank}(\hat{X}) = r$. Moreover, the preconditioner we propose actually requires this.

There are many variations on the theme of low-rank matrix completion. For example, Chandrasekaran et al. [12] and Tao and Yuan [38], among others, focus on identifying a sum of low-rank and sparse matrices as the target matrix. This notably applies to system identification and background extraction in videos. Sensor network localization can also be modeled as low-rank matrix completion, with an additional positive semidefiniteness constraint [30]. Another line of research is *bounded* matrix factorization, as investigated by Kannan et al. [22]. In such applications, one assumes the underlying matrix X is low-rank and that its entries belong to a given, bounded interval. Yet another related research venue is *nonnegative* matrix factorization [18], where the entries of the factors U and W are further required to be nonnegative. The abundance of optimization problems involving rank constraints is further motivation to study low-rank matrix completion as a model problem.

1.2. Contribution and outline of the paper

Dai et al.'s initial formulation (3) has a discontinuous objective function on the Grassmannian. The OptSpace formulation (4) on the other hand has a continuous objective, but optimizes on a higher-dimensional search space (two Grassmannians), while it is arguably preferable to keep the dimension of the nonlinear manifold search space low,

even at the expense of a larger least-squares problem. Furthermore, the OptSpace regularization term is efficiently computable since $\|USV^\top\|_{\mathrm{F}} = \|S\|_{\mathrm{F}}$, but it penalizes all entries instead of just the entries $(i, j) \notin \Omega$. The preconditioned metrics introduced in [31] and [33] bring notable improvements, suggesting that one should strive for a formulation which admits efficient preconditioners.

To keep the nonlinear part of the optimization problem small, we favor an approach with a single Grassmannian, that is, we optimize over the column space alone, assuming without loss of generality that it is smaller than the row space ($m \leq n$). This is particularly useful for rectangular matrices ($m < n$), which is often the case in applications. We equip (3) with a regularization term weighted by $\lambda > 0$ as follows:

$$\min_{\mathscr{U} \in \mathrm{Gr}(m,r)} \min_{W \in \mathbb{R}^{r \times n}} \frac{1}{2} \sum_{(i,j) \in \Omega} C_{ij}^2 \big((UW)_{ij} - X_{ij}\big)^2 + \frac{\lambda^2}{2} \sum_{(i,j) \notin \Omega} (UW)_{ij}^2.$$

A positive value of λ ensures that the inner least-squares problem has a unique solution which is a continuous function of U, so that the cost function in $\mathscr{U}$ is smooth—see Section 3.3. The confidence indices $C_{ij} > 0$ for each observation X_{ij} may be useful in applications. Mathematically, introducing a regularization term is essential to ensure smoothness of the objective and hence obtain good convergence properties. For real datasets, regularization is also practically important, as Section 6 demonstrates.

It turns out that computing the Hessian of the resulting cost function on $\mathrm{Gr}(m, r)$ is cheap, motivating the study of second-order methods. More importantly, having access to an explicit expression for the true Hessian, a simplified analysis is carried out to propose a preconditioner for it. The resulting preconditioner is very similar to the preconditioned metrics proposed in [33,31]. The differences are: (i) it operates over a single Grassmannian, and (ii) we do not scale the Riemannian metric, but rather precondition the iterations of the optimization algorithms used. This means the standard geometry of the Grassmannian is all we need. Preconditioning for Riemannian optimization is an interesting research topic in and of itself; see for example [29].

The cost function is then minimized using a preconditioned Riemannian trust-region method (RTRMC 2p) or a preconditioned Riemannian conjugate gradient method (RCGMCp), as described in Section 4. In practice, we use the Manopt toolbox [9] to execute the optimization. Convergence to critical points is guaranteed, but the quality of the points toward which we converge is not. In that sense, the proposed methods are heuristic, but we will observe excellent behavior in practice.

Section 2 covers essential geometric tools on the Grassmann manifold. Section 3 specifies the cost function and develops expressions for its gradient and Hessian, along with a preconditioner for the latter. Section 4 details how the Riemannian optimization algorithms are set in place with preconditioning. Sections 5 and 6 show results of numerical experiments demonstrating the effectiveness of the proposed approach.

This paper, adapted from the first author's PhD thesis [7], is an extension of a previously published conference paper [8]. It features more detailed mathematical de-

velopments and a number of new numerical experiments. The preconditioner is new as well as the explicit treatment of preconditioned trust-regions and conjugate gradients in the Riemannian setting.

2. Geometry of the Grassmann manifold

We tackle low-rank matrix completion as an optimization problem on the Grassmann manifold. The cost function f (19)—which we construct in the next section—is defined over said manifold $\mathrm{Gr}(m,r)$: the set of r-dimensional linear subspaces of $\mathbb{R}^m$. Absil et al. [2] give a computation-oriented description of the geometry of this manifold. This section only gives a summary of the required tools. We assume some familiarity with standard differential geometric concepts, which are described for example in [2, Ch. 3] and [6]. The former reference is freely available online.

Each point $\mathscr{U} \in \mathrm{Gr}(m,r)$ is a linear subspace that may be represented numerically as the column space of a full-rank matrix U:

$$\mathrm{Gr}(m,r) = \{\mathscr{U} = \mathrm{col}(U) : U \in \mathbb{R}_*^{m\times r}\}.$$

The notation $\mathbb{R}_*^{m\times r}$ stands for the set of full-rank $m \times r$ matrices. For numerical reasons, we only use orthonormal matrices U to represent subspaces. The set $\mathrm{St}(m,r)$ of orthonormal matrices is the (compact) Stiefel manifold:

$$\mathrm{St}(m,r) = \{U \in \mathbb{R}_*^{m\times r} : U^\top U = I_r\}. \tag{5}$$

We view $\mathrm{St}(m,r)$ as a Riemannian submanifold of the Euclidean space $\mathbb{R}^{m\times r}$, endowed with the classical metric $\langle H_1, H_2\rangle = \mathrm{trace}(H_1^\top H_2)$. The submersion $\mathrm{col}\colon \mathrm{St}(m,r) \to \mathrm{Gr}(m,r)$ induces an equivalence relation such that U and U' are equivalent if $\mathrm{col}(U) = \mathrm{col}(U')$, that is, if U and U' have the same column space. Let

$$\mathrm{O}(r) = \{Q \in \mathbb{R}_*^{r\times r} : Q^\top Q = I_r\}$$

denote the set of $r \times r$ orthogonal matrices. Since U and U' are equivalent if and only if there exists some $Q \in \mathrm{O}(r)$ such that $U' = UQ$, we say that $\mathrm{Gr}(m,r)$ is a quotient of $\mathrm{St}(m,r)$ by the action of $\mathrm{O}(r)$:

$$\mathrm{Gr}(m,r) = \mathrm{St}(m,r)/\mathrm{O}(r). \tag{6}$$

Hence, from this perspective, points on the Grassmann manifold are equivalence classes. Numerically, an equivalence class is represented by one of its members. Numerical algorithms can be said to operate on the Grassmann manifold if their behavior only depends on the equivalence classes encountered, and not on the arbitrarily chosen representatives. This property will be enforced in this paper.

The Grassmannian is a manifold, and as such admits a tangent space at each point $\mathscr{U}$, noted $\mathrm{T}_{\mathscr{U}}\,\mathrm{Gr}(m,r)$. The latter is a linear subspace of dimension $\dim \mathrm{Gr}(m,r) = r(m-r)$. Given $U \in \mathrm{St}(m,r)$ that represents $\mathscr{U}$, a tangent vector $\mathscr{H} \in \mathrm{T}_{\mathscr{U}}\,\mathrm{Gr}(m,r)$ is represented by a matrix $H \in \mathbb{R}^{m\times r}$ verifying $\frac{\mathrm{d}}{\mathrm{d}t}\,\mathrm{col}(U+tH)\big|_{t=0} = \mathscr{H}$. This representation H of $\mathscr{H}$, known as its *horizontal lift* at U, is one-to-one if we further impose $U^\top H = 0$. For practical purposes, we often refer to $\mathscr{U}$ and $\mathscr{H}$ using their matrix counterparts U and H instead. Likewise, we abusively call

$$\mathrm{T}_U\,\mathrm{Gr}(m,r) = \{H \in \mathbb{R}^{m\times r} : U^\top H = 0\} \tag{7}$$

the "tangent space to $\mathrm{Gr}(m,r)$ at U"; the abuse of notation and language is readily spotted since U, a matrix, cannot be an element of $\mathrm{Gr}(m,r)$.[1] This abuse of notation and language has the benefit of conveying the intuition while making it clearer how one can numerically work with the abstract objects on $\mathrm{Gr}(m,r)$. However, though we will seldom do so explicitly below for the sake of keeping the exposition streamlined, it remains essential to make sure that the numerical representations induce objects on $\mathrm{Gr}(m,r)$ in a consistent way, i.e., that does not depend on the representation U of $\mathscr{U}$ chosen in the equivalence class $\mathrm{col}^{-1}(\mathscr{U})$. For example, let $\mathscr{U}$ be a subspace represented by U and let $\mathscr{H}$ be a tangent vector at $\mathscr{U}$ correspondingly represented as H. For any orthogonal matrix Q, we may choose to represent $\mathscr{U}$ as UQ instead, in which case $\mathscr{H}$ must be represented as HQ. All numerical operations must be consistent with respect to this freedom of representation.

Each tangent space is endowed with an inner product (the Riemannian metric) that varies smoothly from point to point. It is inherited from the embedding space $\mathbb{R}^{m\times r}$ of the matrix representation of tangent vectors:

$$\forall H_1, H_2 \in \mathrm{T}_U\,\mathrm{Gr}(m,r), \quad \langle H_1, H_2\rangle_U = \mathrm{trace}(H_1^\top H_2).$$

This construction turns col into a Riemannian submersion. As a result, the proposed Riemannian metric on $\mathrm{Gr}(m,r)$ is the (essentially) unique metric that is invariant by rotation of $\mathbb{R}^m$ [26]—an arguably natural property to desire.

The orthogonal projector from $\mathbb{R}^{m\times r}$ onto the tangent space $\mathrm{T}_U\,\mathrm{Gr}(m,r)$ is given by:

$$\mathrm{Proj}_U\colon \mathbb{R}^{m\times r} \to \mathrm{T}_U\,\mathrm{Gr}(m,r)\colon H \mapsto \mathrm{Proj}_U\,H = (I - UU^\top)H.$$

One can similarly define the tangent space at U to the Stiefel manifold:

$$\mathrm{T}_U\,\mathrm{St}(m,r) = \{H \in \mathbb{R}^{m\times r} : U^\top H + H^\top U = 0\}.$$

The projector from the ambient space $\mathbb{R}^{m\times r}$ onto the tangent space of the Stiefel manifold is given by:

[1] In the parlance of differential geometry, (7) is called a *horizontal space* at U.

$$\begin{aligned}\mathrm{Proj}_U^{\mathrm{St}} &: \mathbb{R}^{m\times r} \to \mathrm{T}_U \,\mathrm{St}(m,r) \\ &: H \mapsto \mathrm{Proj}_U^{\mathrm{St}}\, H = (I - UU^\top)H + U\,\mathrm{skew}\left(U^\top H\right),\end{aligned}$$

where $\mathrm{skew}\,(A) = (A - A^\top)/2$ extracts the skew-symmetric part of A.

We now consider the differentiation of functions defined on the Grassmannian. Let $\bar{f}$ be a suitably smooth mapping from $\mathbb{R}_*^{m\times r}$ to $\mathbb{R}$. Let $\bar{f}\big|_{\mathrm{St}}$ denote its restriction to the Stiefel manifold and let us further assume that

$$\forall U \in \mathrm{St}(m,r), Q \in \mathrm{O}(r), \quad \bar{f}\big|_{\mathrm{St}}(U) = \bar{f}\big|_{\mathrm{St}}(UQ).$$

Under this assumption, $\bar{f}\big|_{\mathrm{St}}$ is only a function of the column space of its argument, hence

$$f\colon \mathrm{Gr}(m,r) \to \mathbb{R}\colon \mathrm{col}(U) \mapsto f(\mathrm{col}(U)) = \bar{f}\big|_{\mathrm{St}}(U)$$

is well-defined. The gradient of f at U is the unique tangent vector $\mathrm{grad}\, f(U)$ in $\mathrm{T}_U \,\mathrm{Gr}(m,r)$ satisfying

$$\forall H \in \mathrm{T}_U \,\mathrm{Gr}(m,r), \quad \langle \mathrm{grad}\, f(U), H\rangle_U = \mathrm{D}f(U)[H],$$

where $\mathrm{D}f(U)[H]$ is the directional derivative of f at U along H,

$$\mathrm{D}f(U)[H] = \lim_{t\to 0} \frac{f(\mathrm{col}(U+tH)) - f(\mathrm{col}(U))}{t}.$$

Observe that $\mathrm{grad}\, f(U)$ is again an abuse of notation. In fact, $\mathrm{grad}\, f(U)$ is the horizontal lift of $\mathrm{grad}\, f(\mathscr{U})$ at U, and the way we abuse notations is justified by the theory of Riemannian submersions, see [2, §3.6.2, §5.3.4]. A similar definition holds for $\mathrm{grad}\, \bar{f}$ (the usual gradient) and $\mathrm{grad}\, \bar{f}\big|_{\mathrm{St}}$. Since $\mathrm{St}(m,r)$ is a Riemannian submanifold of $\mathbb{R}_*^{m\times r}$, Ref. [2, Eq. (3.37)] has it that

$$\mathrm{grad}\, \bar{f}\big|_{\mathrm{St}}(U) = \mathrm{Proj}_U^{\mathrm{St}}\left(\mathrm{grad}\, \bar{f}(U)\right). \tag{8}$$

That is, the gradient of the restricted function is obtained by computing the gradient of $\bar{f}$ in the usual way, then projecting the resulting vector onto the tangent space to the Stiefel manifold. Furthermore, since $\mathrm{Gr}(m,r)$ is a Riemannian quotient manifold of $\mathrm{St}(m,r)$, the same reference [2, Eq. (3.39)] has it that

$$\mathrm{grad}\, f(U) = \mathrm{grad}\, \bar{f}\big|_{\mathrm{St}}(U). \tag{9}$$

The notation $\mathrm{grad}\, f(U)$ stands for the numerical (matrix) representation of the abstract tangent vector $\mathrm{grad}\, f(\mathrm{col}(U))$ with respect to the (arbitrary) choice of orthonormal basis U. They are related by:

$$\frac{\mathrm{d}}{\mathrm{d}t}\operatorname{col}(U + t\operatorname{grad} f(U))\Big|_{t=0} = \operatorname{grad} f(\operatorname{col}(U)). \tag{10}$$

Notice that $\mathrm{T}_U \operatorname{Gr}(m,r)$ is a linear subspace of $\mathrm{T}_U \operatorname{St}(m,r)$, so that $\operatorname{Proj}_U \circ \operatorname{Proj}_U^{\mathrm{St}} = \operatorname{Proj}_U$. Since $\operatorname{grad} f(U)$ belongs to $\mathrm{T}_U \operatorname{Gr}(m,r)$, it is invariant under Proj_U. Combining (8) and (9) and applying Proj_U on both sides, we finally obtain a practical means of computing the gradient of f:

$$\operatorname{grad} f(U) = \operatorname{Proj}_U \left(\operatorname{grad} \bar{f}(U)\right) = (I - UU^\top) \operatorname{grad} \bar{f}(U). \tag{11}$$

In practice, this means that we need only compute the gradient of $\bar{f}$ in the usual way and then project with $I - UU^\top$.

Similar techniques apply to derive the Hessian of f at U along H in the tangent space $\mathrm{T}_U \operatorname{Gr}(m,r)$. Define the vector field $\bar{F}\colon \mathbb{R}_*^{m\times r} \to \mathbb{R}^{m\times r}$:

$$\bar{F}(U) = (I - UU^\top) \operatorname{grad} \bar{f}(U).$$

The restriction of $\bar{F}$ to the Stiefel manifold, $\bar{F}\big|_{\mathrm{St}}$, is a tangent vector field, i.e., $\bar{F}(U) \in \mathrm{T}_U \operatorname{St}(m,r)$ for all $U \in \operatorname{St}(m,r)$. Then, for all H in $\mathrm{T}_U \operatorname{Gr}(m,r) \subset \mathrm{T}_U \operatorname{St}(m,r)$, following [2, Eq. (5.14)],

$$\overline{\nabla}_H \, \bar{F}\big|_{\mathrm{St}}(U) = \operatorname{Proj}_U^{\mathrm{St}} \left(\mathrm{D}\bar{F}(U)[H]\right),$$

where $\mathrm{D}\bar{F}(U)[H]$ is the usual directional derivative of $\bar{F}$ at U along H and $\overline{\nabla}_H$ denotes the Levi-Civita connection on the Stiefel manifold w.r.t. any smooth tangent vector field X such that $X(U) = H$. This is the analog on manifolds of directional derivatives of vector-valued functions. Furthermore, the Hessian is expressed in terms of the connection as [2, Eq. (5.18)]:

$$\operatorname{Hess} f(U)[H] = \operatorname{Proj}_U \left(\overline{\nabla}_H \, \bar{F}\big|_{\mathrm{St}}(U)\right),$$

where $\operatorname{Hess} f(U)[H]$ is the derivative at U along H (w.r.t. the Levi-Civita connection on the Grassmannian) of the gradient vector field $\operatorname{grad} f$. Putting these two statements together and remembering that $\operatorname{Proj}_U \circ \operatorname{Proj}_U^{\mathrm{St}} = \operatorname{Proj}_U$, we find a simple expression for the Hessian of f at $\operatorname{col}(U)$ along H w.r.t. the (arbitrary) choice of orthonormal basis U (that is, lifted at U):

$$\operatorname{Hess} f(U)[H] = \operatorname{Proj}_U \left(\mathrm{D}\bar{F}(U)[H]\right) = (I - UU^\top)\mathrm{D}\bar{F}(U)[H]. \tag{12}$$

In practice then, we simply need to differentiate the expression for $\operatorname{grad} f(U)$ "as if it were defined on $\mathbb{R}_*^{m\times r}$" and project accordingly.

For iterative optimization algorithms, it is important to be able to move on the manifold from some initial point U along some prescribed direction specified by a tangent

vector H: this is indeed the basic operation required by all descent methods. The theoretically appealing way of doing this would be to follow the geodesic originating at U and moving in the direction prescribed by H. This is often numerically expensive or difficult to compute. Fortunately, cheaper ways of moving on a manifold, known as *retractions*, can be used instead of geodesics without compromising the convergence properties of many optimization algorithms [2]. We use the following retraction on $\mathrm{Gr}(m, r)$ to move away from U along H while remaining on the manifold:

$$\mathrm{Retraction}_U(H) = \mathrm{polar}(U + H), \tag{13}$$

where $\mathrm{polar}(A) \in \mathrm{St}(m, r)$ designates the $m \times r$ orthonormal factor of the polar decomposition of $A \in \mathbb{R}^{m\times r}$. This is computed using the thin SVD for example: $A = U\Sigma V^\top$ and $\mathrm{polar}(A) = UV^\top$. In abstract terms, this corresponds to having $\mathrm{col}(\mathrm{Retraction}_U(H)) = \mathrm{col}(U + H)$. For tangent vectors, $U^\top H = 0$ so that $U + H$ is always full rank and this is well-defined. Notice that, as required for consistency, if H is a tangent vector at U such that $\mathrm{Retraction}_U(H) = V$, then $\mathrm{Retraction}_{UQ} HQ = VQ$ for all orthogonal matrices Q.

For the Riemannian conjugate gradient method, it is necessary to compare vectors belonging to different tangent spaces. Typically, this happens when one wants to combine the gradient at the present iterate with the search direction used at the previous iterate. One proper way of achieving this is to use a *vector transport* in accordance with the chosen retraction [2, §8.1]. We use the following simple procedure to transport a tangent vector H at U to the tangent space at V:

$$\mathrm{Transport}_{V\leftarrow U}(H) = (I - VV^\top)H. \tag{14}$$

It is readily checked that $\mathrm{Transport}_{VQ\leftarrow UQ}(HQ) = (\mathrm{Transport}_{V\leftarrow U}(H))Q$ for all orthogonal matrices Q. This invariance property guarantees that (14) consistently induces a vector transport on the Grassmann manifold.

3. The cost function and its derivatives

We seek an $m \times n$ matrix $\hat{X}$ of rank at most r (and usually exactly r) such that $\hat{X}$ agrees as much as possible with a matrix X whose entries at the observation set Ω are given. Furthermore, we are given a weight matrix $C \in \mathbb{R}^{m\times n}$ indicating the confidence we have in each observed entry of X. The matrix C is positive at entries in Ω and zero elsewhere. To this end, we propose to minimize the following cost function w.r.t. $U \in \mathbb{R}^{m\times r}_*$ and $W \in \mathbb{R}^{r\times n}$, where $(X_\Omega)_{ij}$ equals X_{ij} if $(i, j) \in \Omega$ and is zero otherwise:

$$\begin{aligned} g &: \mathbb{R}^{m\times r}_* \times \mathbb{R}^{r\times n} \to \mathbb{R} \\ &: (U, W) \mapsto g(U, W) = \frac{1}{2}\left\|C \odot (UW - X_\Omega)\right\|^2_\Omega + \frac{\lambda^2}{2}\left\|UW\right\|^2_{\Omega_c}. \end{aligned} \tag{15}$$

The notation $\odot$ denotes the entry-wise product, $\lambda > 0$ is a regularization parameter, Ω_c is the complement of the set Ω and

$$\|M\|_\Omega^2 \triangleq \sum_{(i,j)\in\Omega} M_{ij}^2.$$

The interpretation is as follows: we are looking for an optimal matrix $\hat{X} = UW$ of rank at most r; we have confidence C_{ij} that $\hat{X}_{ij}$ should equal X_{ij} for $(i,j) \in \Omega$ and smaller confidence λ that $\hat{X}_{ij}$ should equal 0 for $(i,j) \notin \Omega$.

For a fixed U, computing the matrix W that minimizes (15) is a least-squares problem. As we shall see in Section 3.3, the solution to that problem exists and is unique since we assume $\lambda > 0$. Let us note $g_U(W) \triangleq g(U,W)$. As shown later, the mapping between U and this unique, optimal W,

$$W_U = W(U) = \operatorname*{argmin}_{W\in\mathbb{R}^{r\times n}} g_U(W),$$

is smooth and easily computable. It is thus natural to consider the following cost function defined over the set of full-rank matrices $U \in \mathbb{R}_*^{m\times r}$:

$$\hat{f}\colon \mathbb{R}_*^{m\times r} \to \mathbb{R}\colon U \mapsto \hat{f}(U) = \frac{1}{2}\left\|C \odot (UW_U - X_\Omega)\right\|_\Omega^2 + \frac{\lambda^2}{2}\left\|UW_U\right\|_{\Omega_c}^2. \tag{16}$$

By virtue of the discussion in Section 1, we expect that the function $\hat{f}$ be constant over sets of full-rank matrices U spanning the same column space. The following holds:

$$\forall M \in \mathbb{R}_*^{r\times r}, \quad W_{UM} = M^{-1}W_U.$$

Indeed, since $g(U,W)$ merely depends on the product UW, for any $M \in \mathbb{R}_*^{r\times r}$ we have that $g_U(W)$ and $g_{UM}(M^{-1}W)$ are two identical functions of W. Hence, since W_U is the unique minimizer of g_U, it holds that $M^{-1}W_U$ is the unique minimizer of g_{UM}, i.e., $W_{UM} = M^{-1}W_U$. As a consequence, $\hat{X} = UW_U = (UM)W_{UM}$ for all $M \in \mathbb{R}_*^{r\times r}$. For such matrices M, it then follows as expected that:

$$\hat{f}(UM) = \hat{f}(U).$$

This induces an equivalence relation $\sim$ over the $m \times r$ matrices of full rank, $\mathbb{R}_*^{m\times r}$. Two such matrices are equivalent if and only if they have the same column space:

$$U \sim U' \Leftrightarrow \exists M \in \mathbb{R}_*^{r\times r} \text{ s.t. } U' = UM \Leftrightarrow \operatorname{col}(U) = \operatorname{col}(U').$$

Thus, if U and U' are equivalent, they lead to the same estimation of X: $\hat{X} = UW_U = U'W_{U'}$, and it certainly makes sense for our purpose to group them in an equivalence class. Each such equivalence class is identified with $\operatorname{col}(U)$, so that the set of equivalence

classes is the Grassmann manifold $\mathrm{Gr}(m,r)$: the set of r-dimensional linear subspaces embedded in $\mathbb{R}^m$ described in Section 2. Under this description, the Grassmannian is seen as the quotient space $\mathbb{R}_*^{m\times r}/\mathbb{R}_*^{r\times r}$, which is an alternative to the quotient structure $\mathrm{St}(m,r)/\mathrm{O}(r)$ (6) developed in Section 2.

Consequently, $\hat{f}$ descends to a well-defined function over the Grassmann manifold. Our task is to minimize this function. Doing so singles out a column space $\mathrm{col}(U)$. We may then pick any basis of that column space, say U, and compute W_U. The product $\hat{X} = UW_U$ (which is invariant w.r.t. the choice of basis U of $\mathrm{col}(U)$) is then our completion of the matrix X. In the next section, we rearrange the terms in $\hat{f}$ to make it easier to compute and give a slightly modified definition of the cost function.

Note that f is a smooth function defined over a smooth and compact manifold. Hence, it always admits a global optimizer, where the gradient must vanish.

3.1. Rearranging the cost function

Considering (16), it may seem that evaluating $\hat{f}(U)$ requires the computation of the product UW_U at the entries in Ω *and* Ω_c, i.e., we would need to compute the whole matrix UW_U, which cannot cost much less than $\mathcal{O}(mnr)$. Since applications typically involve very large values of the product mn, this is not acceptable. Fortunately, the regularization term $\|UW_U\|_{\Omega_c}^2$ can be computed cheaply based on the computations that need to be executed for the principal term. Indeed, observe that:

$$\|UW_U\|_\Omega^2 + \|UW_U\|_{\Omega_c}^2 = \|UW_U\|_{\mathrm{F}}^2 = \mathrm{trace}(U^\top UW_U W_U^\top). \tag{17}$$

The right-most quantity is computable in $\mathcal{O}((m+n)r^2)$ flops and since $(UW_U)_\Omega$ has to be computed for the first term in the objective function, $\|UW_U\|_{\Omega_c}^2$ turns out to be cheap to obtain. As a result, we see that computing $\hat{f}(U)$ as a whole only requires the computation of $(UW_U)_\Omega$ as opposed to the whole product UW_U, conferring to $\hat{f}$ a computational cost that is linear in the number of observed entries $k = |\Omega|$.

We have the freedom to represent a column space with any of its bases. From a numerical standpoint, it is sound to restrict our attention to orthonormal bases. The set of orthonormal bases U is termed the Stiefel manifold (5). Assuming $U \in \mathrm{St}(m,r)$, Eq. (17) and $U^\top U = I_r$ yield a simple expression for the regularization term:

$$\|UW_U\|_{\Omega_c}^2 = \|W_U\|_{\mathrm{F}}^2 - \|UW_U\|_\Omega^2 .$$

Based on this observation, we introduce the following function over $\mathbb{R}_*^{m\times r}$:

$$\bar{f}\colon \mathbb{R}_*^{m\times r} \to \mathbb{R}\colon U \mapsto \bar{f}(U) = \frac{1}{2}\|C \odot (UW_U - X_\Omega)\|_\Omega^2 + \frac{\lambda^2}{2}\left(\|W_U\|_{\mathrm{F}}^2 - \|UW_U\|_\Omega^2\right). \tag{18}$$

In particular, it is the restriction of $\bar{f}$ to $\mathrm{St}(m,r) \subset \mathbb{R}_*^{m\times r}$ that makes sense for our problem:

$$\bar{f}|_{\mathrm{St}}\colon \mathrm{St}(m,r) \to \mathbb{R}\colon U \mapsto \bar{f}|_{\mathrm{St}}(U) = \bar{f}(U).$$

Notice that this restriction coincides with the original cost function: $\hat{f}|_{\mathrm{St}} \equiv \bar{f}|_{\mathrm{St}}$. We then define our objective function f over the Grassmannian:

$$f\colon \mathrm{Gr}(m,r) \to \mathbb{R}\colon \mathrm{col}(U) \mapsto f(\mathrm{col}(U)) = \bar{f}|_{\mathrm{St}}(U), \tag{19}$$

where U is any orthonormal basis of the column space $\mathrm{col}(U)$. This is well-defined since $\bar{f}|_{\mathrm{St}}(U) = \bar{f}|_{\mathrm{St}}(UQ)$ for all orthogonal Q. On the other hand, notice that $\bar{f}$ does *not* reduce to a function on the Grassmannian (it does not have the invariance property $\bar{f}(UM) = \bar{f}(U)\ \forall M \in \mathbb{R}_*^{r\times r}$), which explains why we had to first go through the Stiefel manifold.

Assuming W_U is available, computing $f(\mathrm{col}(U))$ only requires the computation of UW_U at entries in Ω, at a cost of $\mathcal{O}(kr)$ flops, where $k = |\Omega|$ is the number of known entries. Computing $\|W_U\|_{\mathrm{F}}^2$ costs $\mathcal{O}(nr)$ flops, hence a total evaluation cost of $\mathcal{O}((k+n)r)$ flops. This cost is dominated by the computation of W_U, to be detailed in Section 3.3 to obtain the total complexity of evaluating f.

3.2. Gradient and Hessian of the objective function

We now obtain the first- and second-order derivatives of f (19). As outlined in Section 2, $\mathrm{grad}\, f(\mathrm{col}(U))$ is a tangent vector to the quotient manifold $\mathrm{Gr}(m,r)$. Because of the abstract nature of quotient manifolds, this vector is an abstract object too. In practice, we represent it as a concrete matrix $\mathrm{grad}\, f(U)$ w.r.t. an (arbitrary) orthonormal basis U of $\mathrm{col}(U)$. Eq. (10) establishes the link between $\mathrm{grad}\, f(\mathrm{col}(U))$ and $\mathrm{grad}\, f(U)$. Following (11), we have a convenient expression for $\mathrm{grad}\, f(U)$:

$$\mathrm{grad}\, f(U) = (I - UU^\top)\, \mathrm{grad}\, \bar{f}(U).$$

We thus first set out to compute $\mathrm{grad}\, \bar{f}(U)$, which is a classical gradient.

Introduce the function $h\colon \mathbb{R}_*^{m\times r} \times \mathbb{R}^{r\times n} \to \mathbb{R}$ as follows:

$$h(U,W) = \frac{1}{2}\, \|C \odot (UW - X_\Omega)\|_\Omega^2 + \frac{\lambda^2}{2} \left(\|W\|_{\mathrm{F}}^2 - \|UW\|_\Omega^2 \right). \tag{20}$$

Obviously, h is related to $\bar{f}$ via

$$\bar{f}(U) = \min_W h(U,W) = h(U, W_U).$$

By the definition of the classical gradient, $\mathrm{grad}\, \bar{f}(U) \in \mathbb{R}^{m\times r}$ is the unique vector that satisfies the following condition:

$$\forall H \in \mathbb{R}^{m\times r}, \quad \mathrm{D}\bar{f}(U)[H] = \left\langle H, \mathrm{grad}\, \bar{f}(U) \right\rangle,$$

where $\mathrm{D}\bar{f}(U)[H]$ is the directional derivative of $\bar{f}$ at U along H and $\langle A, B\rangle = \mathrm{trace}(A^\top B)$ is the usual inner product on $\mathbb{R}^{m\times r}$. We thus need to compute the directional derivatives of $\bar{f}$, which can be done in terms of the directional derivatives of h. Indeed, by the chain rule, it holds that:

$$\mathrm{D}\bar{f}(U)[H] = \mathrm{D}_1 h(U, W_U)[H] + \mathrm{D}_2 h(U, W_U)[W_{U,H}],$$

where D_i indicates differentiation w.r.t. the ith argument and

$$W_{U,H} \triangleq \mathrm{D}W(U)[H]$$

is the directional derivative of the mapping $U \mapsto W_U$ at U along H. Since

$$W_U = \operatorname*{argmin}_W h(U, W),$$

W_U is a critical point of $h(U, \cdot)$ and it holds that $\mathrm{D}_2 h(U, W_U) = 0$. This substantially simplifies the computations as now $\mathrm{D}\bar{f}(U)[H] = \mathrm{D}_1 h(U, W_U)[H]$: it suffices to differentiate h w.r.t. U, considering W_U as constant. Define the mask $\Lambda \in \mathbb{R}^{m\times n}$ as:

$$\Lambda_{ij} = \begin{cases} \lambda & \text{if } (i,j) \in \Omega, \\ 0 & \text{otherwise.} \end{cases}$$

Using this notation, we may rewrite h in terms of Frobenius norms only:

$$h(U, W) = \frac{1}{2}\left\|C \odot (UW - X_\Omega)\right\|_{\mathrm{F}}^2 + \frac{\lambda^2}{2}\left\|W\right\|_{\mathrm{F}}^2 - \frac{1}{2}\left\|\Lambda \odot (UW)\right\|_{\mathrm{F}}^2.$$

This is convenient for differentiation, since for suitably smooth mappings g,

$$\mathrm{D}\big(X \mapsto 1/2\left\|g(X)\right\|_{\mathrm{F}}^2\big)(X)[H] = \langle \mathrm{D}g(X)[H], g(X)\rangle.$$

The following holds for all real matrices A, B, C of identical sizes:

$$\langle A \odot B, C\rangle = \langle B, A \odot C\rangle.$$

Throughout, we use the notation $M^{(n)}$ for entry-wise exponentiation, i.e.,

$$(M^{(n)})_{ij} \triangleq (M_{ij})^n.$$

Combining the last few statements, it follows that:

$$\begin{aligned}
\mathrm{D}\bar{f}(U)[H] &= \mathrm{D}_1 h(U, W_U)[H] \\
&= \langle C \odot (HW_U), C \odot (UW_U - X_\Omega)\rangle - \langle \Lambda \odot (HW_U), \Lambda \odot (UW_U)\rangle \\
&= \left\langle H, \left[C^{(2)} \odot (UW_U - X_\Omega)\right] W_U^\top - \left[\Lambda^{(2)} \odot (UW_U)\right] W_U^\top \right\rangle \\
&= \left\langle H, \left[(C^{(2)} - \Lambda^{(2)}) \odot (UW_U - X_\Omega)\right] W_U^\top - \lambda^2 X_\Omega W_U^\top \right\rangle. \qquad (21)
\end{aligned}$$

For ease of notation, define the following $m \times n$ matrix with the sparsity structure induced by Ω:

$$\hat{C} = C^{(2)} - \Lambda^{(2)}. \tag{22}$$

Further introduce the sparse residue matrix R_U:

$$R_U = \hat{C} \odot (UW_U - X_\Omega) - \lambda^2 X_\Omega. \tag{23}$$

By definition, $\operatorname{grad} \bar{f}(U)$ is the unique vector satisfying $\mathrm{D}\bar{f}(U)[H] = \left\langle H, \operatorname{grad} \bar{f}(U) \right\rangle$ for all tangent H. Since (21) holds for all H, by identification, we obtain a simple expression for the sought gradient:

$$\operatorname{grad} \bar{f}(U) = R_U W_U^\top.$$

Recall that $\mathrm{D}_2 h(U, W_U) = 0$ because W_U is a critical point of $h(U, \cdot)$. This translates into the following matrix statement:

$$\begin{aligned} \forall H \in \mathbb{R}^{r \times n}, \quad 0 &= \mathrm{D}_2 h(U, W_U)[H] \\ &= \langle C \odot (UH), C \odot (UW_U - X_\Omega) \rangle + \lambda^2 \langle H, W_U \rangle \\ &\quad - \langle \Lambda \odot (UH), \Lambda \odot (UW_U) \rangle \\ &= \left\langle H, U^\top R_U + \lambda^2 W_U \right\rangle. \end{aligned}$$

Hence,

$$U^\top R_U + \lambda^2 W_U = 0. \tag{24}$$

Summing up, we obtain the gradient of f (19):

$$\operatorname{grad} f(U) = (I - UU^\top) R_U W_U^\top = R_U W_U^\top + \lambda^2 U (W_U W_U^\top). \tag{25}$$

We now differentiate (25) according to the identity (12) for the Hessian of f. To this end, consider $\bar{F} \colon \mathbb{R}_*^{m \times r} \to \mathbb{R}^{m \times r}$:

$$\bar{F}(U) = R_U W_U^\top + \lambda^2 U (W_U W_U^\top).$$

According to (12), the Hessian of f is given by:

$$\operatorname{Hess} f(U)[H] = (I - UU^\top) \mathrm{D}\bar{F}(U)[H]. \tag{26}$$

Let us compute the differential of $\bar{F}$:

$$\mathrm{D}\bar{F}(U)[H] = [\hat{C} \odot (HW_U + UW_{U,H})]W_U^\top + R_U W_{U,H}^\top + \lambda^2 H(W_U W_U^\top) + \lambda^2 U(W_{U,H} W_U^\top + W_U W_{U,H}^\top).$$

Applying the projector $I - UU^\top$ to $\mathrm{D}\bar{F}(U)[H]$ cancels out all terms of the form UM (since $(I - UU^\top)U = 0$) and leaves all terms of the form HM unaffected (since $U^\top H = 0$). As a consequence of (24), applying the projector to $R_U W_{U,H}^\top$ yields:

$$(I - UU^\top)R_U W_{U,H}^\top = R_U W_{U,H}^\top + \lambda^2 U W_U W_{U,H}^\top.$$

Applying these observations to (26), we obtain an expression for the Hessian of the cost function on the Grassmann manifold:

$$\mathrm{Hess}\, f(U)[H] = (I - UU^\top)\left[\hat{C} \odot (HW_U + UW_{U,H})\right] W_U^\top + R_U W_{U,H}^\top + \lambda^2 H(W_U W_U^\top) + \lambda^2 U(W_U W_{U,H}^\top). \tag{27}$$

Not surprisingly, the formula for the Hessian requires the computation of $W_{U,H}$, the differential of the mapping $U \mapsto W_U$ along H. The next section provides formulas for W_U and $W_{U,H}$.

3.3. W_U and its derivative $W_{U,H}$

We still need to provide explicit formulas for W_U and $W_{U,H}$. We assume $U \in \mathrm{St}(m, r)$ since we use orthonormal matrices to represent points on the Grassmannian and $U^\top H = 0$ since H represents a tangent vector at U (the horizontal lift of a tangent vector at $\mathrm{col}(U)$).

The vectorization operator, vec, transforms matrices into vectors by stacking their columns—in Matlab notation, $\mathrm{vec}(A) = A(:)$. Denoting the Kronecker product of two matrices by $\otimes$, the following well-known identity holds, for matrices A, Y, B of appropriate sizes:

$$\mathrm{vec}(AYB) = (B^\top \otimes A)\,\mathrm{vec}(Y).$$

We also write I_Ω for the orthonormal $k \times mn$ matrix such that

$$\mathrm{vec}_\Omega(M) = I_\Omega\,\mathrm{vec}(M)$$

is a vector of length $k = |\Omega|$ corresponding to the entries M_{ij} for $(i, j) \in \Omega$, taken in order from $\mathrm{vec}(M)$.

Computing W_U comes down to minimizing the least-squares cost $h(U, W)$ (20) with respect to W. We manipulate h to reach a standard form for least-squares. To this end, first define $S \in \mathbb{R}^{k \times mn}$:

$$S = I_\Omega \operatorname{diag}(\operatorname{vec}(C)).$$

This will come in handy through the identity

$$\begin{aligned}\|C \odot M\|_\Omega^2 &= \|\operatorname{vec}_\Omega(C \odot M)\|_2^2 = \|I_\Omega \operatorname{vec}(C \odot M)\|_2^2 \\ &= \|I_\Omega \operatorname{diag}(\operatorname{vec}(C)) \operatorname{vec}(M)\|_2^2 = \|S \operatorname{vec}(M)\|_2^2 .\end{aligned}$$

We use this in the following transformation of h:

$$\begin{aligned}h(U,W) &= \frac{1}{2}\|C \odot (UW - X_\Omega)\|_\Omega^2 + \frac{\lambda^2}{2}\|W\|_{\mathrm{F}}^2 - \frac{\lambda^2}{2}\|UW\|_\Omega^2 \\ &= \frac{1}{2}\|S \operatorname{vec}(UW) - \operatorname{vec}_\Omega(C \odot X_\Omega)\|_2^2 + \frac{\lambda^2}{2}\|\operatorname{vec}(W)\|_2^2 - \frac{\lambda^2}{2}\|\operatorname{vec}_\Omega(UW)\|_2^2 \\ &= \frac{1}{2}\|S(I_n \otimes U)\operatorname{vec}(W) - \operatorname{vec}_\Omega(C \odot X_\Omega)\|_2^2 + \frac{1}{2}\|\lambda I_{rn} \operatorname{vec}(W)\|_2^2 \\ &\quad - \frac{1}{2}\|\lambda I_\Omega (I_n \otimes U)\operatorname{vec}(W)\|_2^2 \\ &= \frac{1}{2}\left\|\begin{bmatrix} S(I_n \otimes U) \\ \lambda I_{rn}\end{bmatrix}\operatorname{vec}(W) - \begin{bmatrix}\operatorname{vec}_\Omega(C \odot X_\Omega) \\ 0_{rn}\end{bmatrix}\right\|_2^2 \\ &\quad - \frac{1}{2}\|[\,\lambda I_\Omega (I_n \otimes U)\,]\operatorname{vec}(W)\|_2^2 \\ &= \frac{1}{2}\|A_1 w - b_1\|_2^2 - \frac{1}{2}\|A_2 w\|_2^2 \\ &= \frac{1}{2}w^\top (A_1^\top A_1 - A_2^\top A_2)w - b_1^\top A_1 w + \frac{1}{2}b_1^\top b_1,\end{aligned}$$

where $w = \operatorname{vec}(W) \in \mathbb{R}^{rn}$, $0_{rn} \in \mathbb{R}^{rn}$ is the zero-vector and the obvious definitions for A_1, A_2 and b_1. If $A_1^\top A_1 - A_2^\top A_2$ is positive definite there is a unique minimizing vector $\operatorname{vec}(W_U)$, given by:

$$\operatorname{vec}(W_U) = (A_1^\top A_1 - A_2^\top A_2)^{-1} A_1^\top b_1.$$

It is easy to compute the following:

$$\begin{aligned}A_1^\top A_1 &= (I_n \otimes U^\top)(S^\top S)(I_n \otimes U) + \lambda^2 I_{rn}, \\ A_2^\top A_2 &= (I_n \otimes U^\top)(\lambda^2 I_\Omega^\top I_\Omega)(I_n \otimes U), \\ A_1^\top b_1 &= (I_n \otimes U^\top) S^\top \operatorname{vec}_\Omega(C \odot X_\Omega) = (I_n \otimes U^\top)\operatorname{vec}(C^{(2)} \odot X_\Omega).\end{aligned}$$

Note that $S^\top S - \lambda^2 I_\Omega^\top I_\Omega = \operatorname{diag}(\operatorname{vec}(\hat{C}))$. Let us call this matrix B:

$$B \triangleq S^\top S - \lambda^2 I_\Omega^\top I_\Omega = \operatorname{diag}(\operatorname{vec}(\hat{C})).$$

Then define $A \in \mathbb{R}^{rn \times rn}$ as:

$$A \triangleq A_1^\top A_1 - A_2^\top A_2 = (I_n \otimes U^\top) B (I_n \otimes U) + \lambda^2 I_{rn}. \tag{28}$$

Observe that the matrix A is block-diagonal, with n symmetric blocks of size r. This structure stems from the fact that each column of W_U can be computed separately from the others. Each block is indeed positive definite provided $\lambda > 0$ (making A positive definite too). Owing to the sparsity of $\hat{C}$, we can compute these n blocks with $\mathcal{O}(kr^2)$ flops. To solve systems in A, we compute the Cholesky factorization of each block, at a total cost of $\mathcal{O}(nr^3)$ flops. Once these factorizations are computed, each system only costs $\mathcal{O}(nr^2)$ flops to solve [40]. Collecting all equations in this section, we obtain a closed-form formula for W_U:

$$\begin{aligned} \operatorname{vec}(W_U) &= A^{-1}(I_n \otimes U^\top) \operatorname{vec}(C^{(2)} \odot X_\Omega) \\ &= A^{-1} \operatorname{vec}\left(U^\top [C^{(2)} \odot X_\Omega]\right), \end{aligned} \tag{29}$$

where A is a function of U and we have a fast way of applying A^{-1} to vectors.

We set out to differentiate W_U with respect to U, that is, compute

$$\begin{aligned} \operatorname{vec}(W_{U,H}) &= \mathrm{D}\left(U \mapsto \operatorname{vec}(W_U)\right)(U)[H] \\ &= \mathrm{D}\left(U \mapsto A^{-1}\right)(U)[H] \cdot \operatorname{vec}\left(U^\top [C^{(2)} \odot X_\Omega]\right) \\ &\quad + A^{-1} \operatorname{vec}\left(H^\top [C^{(2)} \odot X_\Omega]\right). \end{aligned} \tag{30}$$

Using the formula $\mathrm{D}(Y \mapsto Y^{-1})(X)[H] = -X^{-1} H X^{-1}$ for the differential of the inverse of a matrix, it follows that

$$\begin{aligned} \mathrm{D}\left(U \mapsto A^{-1}\right)(U)[H] &= -A^{-1} \cdot \mathrm{D}\left(U \mapsto A\right)(U)[H] \cdot A^{-1} \\ &= -A^{-1}\Big[(I_n \otimes H^\top) B (I_n \otimes U) + (I_n \otimes U^\top) B (I_n \otimes H)\Big] A^{-1}. \end{aligned}$$

Plug this back in (30), recalling (29) for W_U:

$$\begin{aligned} \operatorname{vec}(W_{U,H}) &= -A^{-1}\Big[(I_n \otimes H^\top) B (I_n \otimes U) + (I_n \otimes U^\top) B (I_n \otimes H)\Big] \operatorname{vec}(W_U) \\ &\quad + A^{-1} \operatorname{vec}\left(H^\top [C^{(2)} \odot X_\Omega]\right) \\ &= -A^{-1}\Big[(I_n \otimes H^\top) B \operatorname{vec}(U W_U) + (I_n \otimes U^\top) B \operatorname{vec}(H W_U)\Big] \\ &\quad + A^{-1} \operatorname{vec}\left(H^\top [C^{(2)} \odot X_\Omega]\right) \\ &= -A^{-1}\Big[(I_n \otimes H^\top) \operatorname{vec}(\hat{C} \odot U W_U) + (I_n \otimes U^\top) \operatorname{vec}(\hat{C} \odot H W_U)\Big] \end{aligned}$$

$$
\begin{aligned}
&+ A^{-1} \operatorname{vec}\left(H^\top [C^{(2)} \odot X_\Omega]\right) \\
&= -A^{-1}\left[\operatorname{vec}(H^\top [\hat{C} \odot UW_U]) + \operatorname{vec}(U^\top [\hat{C} \odot HW_U])\right] \\
&\quad + A^{-1} \operatorname{vec}\left(H^\top [C^{(2)} \odot X_\Omega]\right). \qquad (31)
\end{aligned}
$$

Now recall the definition of R_U (23) and observe that

$$\hat{C} \odot UW_U - C^{(2)} \odot X_\Omega = \hat{C} \odot UW_U - \hat{C} \odot X_\Omega - \Lambda^{(2)} \odot X_\Omega = R_U.$$

Injecting the latter in (31) yields a compact expression for the directional derivative $W_{U,H}$:

$$\operatorname{vec}(W_{U,H}) = -A^{-1} \operatorname{vec}\left(H^\top R_U + U^\top \left(\hat{C} \odot (HW_U)\right)\right). \qquad (32)$$

The most expensive operation involved in computing $W_{U,H}$ ought to be solving a linear system in A. Fortunately, we already factored the n small diagonal blocks of A in Cholesky form to compute W_U. Consequently, after computing W_U, computing $W_{U,H}$ is cheap. This hints that using second-order methods to optimize f may be profitable. The computational complexities are summarized in Section 3.5.

3.4. Preconditioning the Hessian

As Section 5 will demonstrate with the numerical experiment "Scenario 3", the Hessian of the cost function Hess f (27) can be badly conditioned when the target matrix X is itself badly conditioned. Such conditioning issues slow down optimization algorithms, and it is known that good preconditioners can have a dramatic effect on performance in such cases [14,20].

In this section, we consider a simplified version of the low-rank matrix completion problem, which allows to simplify the expression of the Hessian. This, in turn, yields an approximation of the inverse of the Hessian. That operator is then used to precondition the Hessian in the optimization algorithms discussed in Section 4. Fig. 4 later in this paper demonstrates the effectiveness of this preconditioner.

In order to (markedly) simplify the problem at hand, assume all entries of X are observed, with identical confidence $C_{ij} = c$. Hence, $\hat{C} = (c^2 - \lambda^2)\mathbb{1}_{m\times n}$ where $\mathbb{1}$ is the all-ones matrix. Then, remembering that $U^\top H = 0$, we get successively for R_U (23) and the Hessian (27):

$$
\begin{aligned}
R_U &= c^2(UW_U - X_\Omega) - \lambda^2 UW_U, \quad \text{and} \\
\operatorname{Hess} f(U)[H] &= c^2 H(W_U W_U^\top) + (R_U + \lambda^2 UW_U)W_{U,H}^\top \\
&= c^2 H(W_U W_U^\top) + c^2(UW_U - X_\Omega)W_{U,H}^\top.
\end{aligned}
$$

Now consider $W_{U,H}$. Recalling Eqs. (32) for $W_{U,H}$ and (28) for A, still under the same assumption on $\hat{C}$, it follows that:

$$A \operatorname{vec}(W_{U,H}) = -\operatorname{vec}\left(H^\top R_U\right) = \operatorname{vec}\left(\lambda^2 W_{U,H} + (c^2 - \lambda^2) W_{U,H}\right) = c^2 \operatorname{vec}(W_{U,H}).$$

Hence,

$$W_{U,H} = -\frac{1}{c^2} H^\top R_U = H^\top X_\Omega, \quad \text{and}$$
$$\operatorname{Hess} f(U)[H] = c^2 H(W_U W_U^\top) + c^2 (U W_U - X_\Omega) X_\Omega^\top H.$$

Further assuming we are close to convergence and the observations are not too noisy, that is, $X_\Omega \approx U W_U$, then $U W_U - X_\Omega \approx 0$ and $X_\Omega^\top H \approx 0$. The second term is thus small and the Hessian may be approximated by the mapping $H \mapsto c^2 H(W_U W_U^\top)$. Notice that this mapping is linear and symmetric, from and to the tangent space at U. This observation prompts the following formula for a preconditioner:

$$\operatorname{Precon} f(U)[H] = \frac{1}{c^2} H(W_U W_U^\top)^{-1}. \tag{33}$$

The small $r \times r$ matrix $W_U W_U^\top$ is already computed when the gradient at U is computed. Applying the preconditioner further requires solving linear systems in that matrix. The cost of this is $\mathcal{O}(r^3)$ to prepare a Cholesky factorization of $W_U W_U^\top$ (once per iteration) and an additional $\mathcal{O}(mr^2)$ per application. Notice that this cheap cost is independent of the number of observed entries k. In practice, c can be chosen to be the average value of the positive C_{ij}'s.

It is important that a preconditioner be symmetric and positive definite on the tangent space at U. The proposed preconditioner indeed fulfills these requirements provided W_U is full-rank. The factor W_U is expected to be full-rank near convergence provided the lowest-rank matrix compatible with the observation X_Ω is of rank at least r. In practice, we could monitor the condition number of $W_U W_U^\top$ at each iteration, and decrease r if it becomes too large (indicating the true rank of the sought matrix X was overshot).

Preconditioning the Hessian with (33) is tightly related to the approaches favored by Mishra et al. [31] and by Ngo and Saad [33]. In the latter reference, the authors pose low-rank matrix completion as an optimization problem on two Grassmannians (one for the row space and one for the column space). If (U, V) is a couple of orthonormal matrices representing the row and column spaces of the current estimate $USV^\top$, then the metric on the first Grassmannian is scaled by $SS^\top = (SV^\top)(SV^\top)^\top$ and likewise the metric on the second Grassmannian is scaled by $S^\top S = (US)^\top (US)$ (notice the cross-talk between U and V, akin to our preconditioning iterations on U using the W factor). Mishra et al. [31] represent low-rank matrices on a quotient space of factorizations of the form $GH^\top$. The metric on G is scaled by $H^\top H$ and likewise the metric on H is scaled by $G^\top G$. The effect of changing the metric is similar to the effect of preconditioning, in that it makes the cost function "look more isotropic" by locally distorting the space.

Table 1
All complexities are at most linear in $k = |\Omega|$, the number of observed entries.

Computation	Complexity	By-products	Formulas
W_U and $f(U)$	$\mathcal{O}(kr^2 + nr^3)$	Cholesky of A	(18)–(19), (28), (29)
grad $f(U)$	$\mathcal{O}(kr + (m+n)r^2)$	R_U, $W_U W_U^\top$	(22), (23), (25)
Hess $f(U)[H]$	$\mathcal{O}(kr + (m+n)r^2)$	$W_{U,H}$	(27), (32)
Precon $f(U)[H]$	$\mathcal{O}(mr^2)$		(33)

Underlying the preconditioner (33) is the approximation of the Hessian at U as

$$\operatorname{Hess} f(U)[H] \approx (\operatorname{Precon} f(U))^{-1}[H] = c^2 H(W_U W_U^\top).$$

This operator induces a norm on the tangent space at U, which is called the M-norm (following notation of [14, Alg. 7.5.1]):

$$\|H\|_M^2 = \left\langle H, (\operatorname{Precon} f(U))^{-1}[H] \right\rangle_U = c^2 \left\langle HW_U, HW_U \right\rangle. \tag{34}$$

This norm appears in the theoretical description of the preconditioned Riemannian trust-region method (Section 4) but needs never be computed explicitly.

3.5. Numerical complexities

By exploiting the sparsity of many of the matrices involved as well as the block-diagonal structure of the matrix A (28) appearing in the computation of W_U and $W_{U,H}$, it is possible to compute the objective function f (19), its gradient (25) and its Hessian (27) on the Grassmannian in time linear in the size of the data $k = |\Omega|$. Memory complexities are also linear in k. Table 1 summarizes the computational complexities. Note that most computations are easily parallelizable, but we do not take advantage of it here.

The cost of computing W_U (29) is dominated by the computation of the n diagonal blocks of A of size $r \times r$, $\mathcal{O}(kr^2)$, and by the Cholesky factorization of these, $\mathcal{O}(nr^3)$, hence a total cost of $\mathcal{O}(kr^2 + nr^3)$ flops. The computation of $f(U)$ is dominated by the cost of computing W_U, hence they have the same complexity. Computing the gradient of f once W_U is known involves just a few supplementary matrix–matrix multiplications. Exploiting the sparsity of these matrices keeps the cost low: $\mathcal{O}(kr + (m+n)r^2)$ flops. Computing the Hessian of f along H requires (on top of W_U) the computation of $W_{U,H}$ and a few (structured) matrix–matrix products. Computing $W_{U,H}$ involves solving a linear system in A. Since W_U is computed already, a Cholesky-factored representation of A is available, hence solving a system in A is cheap: $\mathcal{O}(nr^2)$ flops. The total cost of computing $W_{U,H}$ and $\operatorname{Hess} f(U)[H]$ is $\mathcal{O}(kr + (m+n)r^2)$ flops. Interestingly, the cost of applying the preconditioner is independent of k.

Notice that computing the gradient and the Hessian is *cheaper* than computing f. This stems from the fact that once we have computed f at a certain point U, much of the work

(such as computing and factoring the diagonal blocks of A) can be reused to compute higher-order information. This is motivation to investigate optimization methods that exploit second-order information.

4. Riemannian optimization setup

To minimize the cost function f on the Grassmann manifold, we choose Riemannian optimization algorithms which can be preconditioned: the Riemannian trust-region method (RTR) and the Riemannian conjugate gradient method (RCG). Both of these methods are implemented in Manopt [9], an open-source Matlab toolbox for optimization on manifolds, available at http://www.manopt.org. RTR can make full use of the Hessian information, while RCG does not use the Hessian at all. One of the benefits of using Manopt is its built-in caching capabilities, which help prevent redundant computations. After discussing initial iterates, we describe RCG first (as it is easier to expose) and RTR second.

To highlight the generality of the algorithms, we will name $\mathcal{M}$ the manifold over which the cost function f is minimized. In the present case, $\mathcal{M} = \mathrm{Gr}(m, r)$. Points on the manifold $\mathcal{M}$ are designated by lower-case Latin letters such as x. Tangent vectors to $\mathcal{M}$ are designated by lower-case Greek letters such as η. In the absence of a preconditioner, assume $\mathrm{Precon}\, f(x)$ is the identity operator for all x.

The first generic implementation of RTR is the GenRTR package [1]. The Manopt package implements RTR based on that code.

4.1. Initial iterate

Both the RTR and the RCG methods require an initial guess for the column space, $\mathrm{col}(U_0)$. To this end, we compute the r dominant left singular vectors of the masked matrix X_Ω. In Matlab, this is achieved by calling $[U_0, S_0, V_0] = \texttt{svds}(X_\Omega, r)$. Only U_0 is used; S_0 and V_0 are discarded. Since X_Ω is sparse, this has a reasonable cost (essentially linear in k and in r). The initial model is thus $U_0 W_{U_0}$. Note that W_{U_0} depends both on U_0 and on λ.

Alternative methods to compute an initial guess, with analysis, as well as to guess the rank r if it is unknown, can be found in [24] and, more recently, in [13].

4.2. Preconditioned Riemannian conjugate gradients

When it comes to solving a continuous, unconstrained, nonlinear optimization problem of the form

$$\min_{x \in \mathbb{R}^n} f(x),$$

such that f is continuously differentiable, the steepest descent method (SD) is arguably one of the simplest algorithms available. Given an initial iterate $x_0 \in \mathbb{R}^n$, it iteratively improves its predicament by generating a sequence of iterates $x_0, x_1, \ldots \in \mathbb{R}^n$ according to the update equation

$$x_{k+1} = x_k + \alpha_k \eta_k, \tag{35}$$

where $\eta_k = -\nabla f(x_k)$ is the steepest-descent direction at x_k and $\alpha_k > 0$ is a well-chosen step size. The nonlinear conjugate gradients (CG) method adds a sophistication layer to this simple algorithm by constructing an alternative search direction η_k which is a linear combination of both $-\nabla f(x_k)$ and the previous search direction η_{k-1}, thus incorporating a form of inertia in the search procedure:

$$\eta_k = -\nabla f(x_k) + \beta_{k-1}\eta_{k-1}. \tag{36}$$

SD can be conceived as a special case of CG by letting $\beta_k = 0$ for all k.

From the update equation (35) and the search direction equation (36), it is apparent that the CG method relies on the vector space structure of $\mathbb{R}^n$, by composing points and vectors using linear combinations. This dependence is not fundamental though, and both equations can be modified so that they will still make sense for optimization problems on Riemannian manifolds. It is not easy to work with less than the Riemannian structure though, because a notion of gradient is required.

The update equation (35) produces x_{k+1}, a new point on the search space, by moving away from x_k along the direction $\alpha_k \eta_k$. The notion of retraction (13) embodies this very same idea and suggests the more general update formula:

$$x_{k+1} = \mathrm{Retraction}_{x_k}(\alpha_k \eta_k),$$

where $\eta_k \in \mathrm{T}_{x_k}\mathcal{M}$ is a tangent vector at x_k. Similarly, the search direction equation (36) produces the tangent vector η_k by combining two vectors: $-\operatorname{grad} f(x_k)$ and η_{k-1}, where the former is the Riemannian gradient of f at x_k (25). Those are, respectively, tangent vectors at x_k and x_{k-1}. As a result, they cannot be combined directly: they do not belong to the same subspace. One way of fixing this issue is to transport η_{k-1} to x_k using a vector transport (14):

$$\eta_{k-1}^{+} = \mathrm{Transport}_{x_k \leftarrow x_{k-1}}(\eta_{k-1}).$$

The search direction equation then becomes:

$$\eta_k = -\operatorname{grad} f(x_k) + \beta_{k-1}\eta_{k-1}^{+}.$$

See [2] and [36] for discussions of RCG.

A standard trick to accelerate the CG algorithm is to precondition the iterations by operating a change of variables on the tangent spaces $\mathrm{T}_{x_k}\mathcal{M}$ [20, §8]. With a preconditioner (33), the search direction equation reads:

$$\eta_k = -\operatorname{Precon} f(x_k)[\operatorname{grad} f(x_k)] + \beta_{k-1}\eta_{k-1}^{+}.$$

Notice that if $\operatorname{Precon} f(x) = (\operatorname{Hess} f(x))^{-1}$ and $\beta_{k-1} = 0$, this is a Newton step. When no preconditioner is available or necessary, it is replaced by the identity operator.

The step size α_k is chosen by a line search algorithm which approximately solves the one-dimensional optimization problem

$$\min_{\alpha>0} \phi(\alpha) := f(\operatorname{Retraction}_{x_k}(\alpha\eta_k)). \tag{37}$$

If η_k is a descent direction for f (which is typically enforced), then $\phi'(0) < 0$ and it is necessarily possible to decrease ϕ (and hence f) with a positive step size. It does not matter whether (37) is solved exactly or not. Typically, it is sufficient to compute a large enough step size such that a *sufficient decrease* is obtained, according to the Armijo criterion:

$$\begin{aligned} f(x_{k+1}) = \phi(\alpha_k) &\leq \phi(0) + c_{\text{decrease}}\alpha_k\phi'(0) \\ &= f(x_k) + c_{\text{decrease}} \cdot \mathrm{D}f(x_k)[\alpha_k\eta_k]. \end{aligned}$$

The constant $0 < c_{\text{decrease}} < 1$ is the sufficient decrease parameter, set to 10^{-4} by default in our case. The simple backtracking line search, Algorithm 2, guarantees this condition is satisfied. Default values for the other parameters, $0 < c_{\text{initial}}$, $0 < c_{\text{optimism}}$ and $0 < c_{\text{contraction}} < 1$, are $c_{\text{initial}} = 1$, $c_{\text{optimism}} = 1.1$ and $c_{\text{contraction}} = 0.5$.

The line search problem (37) is not any different from the standard line search problem studied in classical textbooks. Algorithm 2, as implemented in Manopt, is based on recommendations in [34, §3.5]. Notice that it is invariant under offsetting and positive scaling of f (assuming a fixed preconditioner). This is a good property: if the cost function changes from $f(x)$ to $8f(x) + 3$, arguably, any reasonable optimization algorithm should still make the same steps. The line search algorithm is also invariant under rescaling of the search direction η in the following sense: the output α is such that the candidate step $\alpha\eta$ is not a function of $\|\eta\|$. Consequently, the combination of Algorithms 1 and 2 as a whole is invariant under offsetting and positive scaling: it is *scale-free*.

It remains to specify how the inertia parameters β_k are computed. The survey paper by Hager and Zhang [20] covers a number of suggestions that have appeared in the literature for the Euclidean case. Those are readily adapted to the Riemannian setting, with special care as outlined in [20, §8] in the presence of a preconditioner. As was already mentioned, the trivial choice $\beta_k = 0$ yields the SD method. A more sophisticated choice known as the modified Hestenes–Stiefel rule is displayed in Algorithm 1. This choice is motivated by its automatic restart property. Indeed, when a negative β_k would be

Algorithm 1 RCG: preconditioned Riemannian conjugate gradients

Given: $x_0 \in \mathcal{M}$
Init: $g_0 = \operatorname{grad} f(x_0), p_0 = \operatorname{Precon} f(x_0)[g_0], \eta_0 = -p_0, k = 0$
repeat
  if $\langle g_k, \eta_k \rangle \geq 0$ **then** ▷ if η_k is not a descent direction
    $\eta_k = -p_k$ ▷ restart
  end if
  $\alpha_k = \text{LINESEARCH}(x_k, \eta_k, x_{k-1})$ ▷ Armijo backtracking
  $x_{k+1} = \operatorname{Retraction}_{x_k}(\alpha_k \eta_k)$ ▷ make the step
  $g_{k+1} = \operatorname{grad} f(x_{k+1})$
  $p_{k+1} = \operatorname{Precon} f(x_{k+1})[g_{k+1}]$
  $\eta_k^+ = \operatorname{Transport}_{x_{k+1} \leftarrow x_k}(\eta_k)$ ▷ transport to the new tangent space
  $g_k^+ = \operatorname{Transport}_{x_{k+1} \leftarrow x_k}(g_k)$
  $\beta_k = \max\left(0, \left\langle g_{k+1} - g_k^+, p_{k+1} \right\rangle / \left\langle g_{k+1} - g_k^+, \eta_k^+ \right\rangle\right)$ ▷ Hestenes–Stiefel +
  $\eta_{k+1} = -p_{k+1} + \beta_k \eta_k^+$ ▷ new search direction
  $k = k + 1$
until a stopping criterion triggers

Algorithm 2 Linesearch: modified Armijo backtracking

Given: $x \in \mathcal{M}$, $\eta \in \mathrm{T}_x\mathcal{M}$ (optional: $x_{\text{prev}} \in \mathcal{M}$)
$\alpha := \begin{cases} c_{\text{optimism}} \cdot 2\frac{f(x) - f(x_{\text{prev}})}{\mathrm{D}f(x)[\eta]} & \text{if } x_{\text{prev}} \text{ is available,} \\ c_{\text{initial}} / \|\eta\| & \text{otherwise.} \end{cases}$
if $\alpha\|\eta\| < 10^{-12}$ **then** ▷ Make sure α is neither negative nor too small
  $\alpha := c_{\text{initial}} / \|\eta\|$
end if
while $f(\operatorname{Retraction}_x(\alpha\eta)) > f(x) + c_{\text{decrease}} \cdot \mathrm{D}f(x)[\alpha\eta]$ **do**
  $\alpha := c_{\text{contraction}} \cdot \alpha$
end while
return α

produced (meaning that the next step would revert some of the previous progress), β_k is set to zero instead. This induces a steepest descent step, often considered a restart of the CG algorithm. Refer to [20] for more rules together with an analysis of when which rules work best.

Even in the Euclidean case $\mathcal{M} = \mathbb{R}^n$, the convergence analysis of nonlinear CG is not a simple matter, see for example [17]. In recent work, Sato and Iwai [36] show how a careful choice of both the β_k coefficients (following the Fletcher–Reeves rule) and the vector transport, together with a line search which satisfies strong Wolfe conditions, can lead to global convergence guarantees for Riemannian CG. The overall algorithm is more involved than the combination of Algorithms 1 and 2 proposed here, especially in its requirements regarding vector transports. In view of the satisfactory numerical

performance of the latter combination of algorithms in applications, we choose to carry on with the simple implementation.

The application of RCG with and without preconditioner to the low-rank matrix completion problem at hand are referred to as RCGMCp and RCGMC, respectively.

4.3. Preconditioned Riemannian trust-regions

The Riemannian trust-region (RTR) method [1], [2, Ch. 7] is a generalization of the classical trust-region scheme [14], [34, Ch. 4] to the class of optimization problems over Riemannian manifolds. Under mild conditions on the cost function, the convergence analysis for RTR guarantees global convergence toward critical points, that is, the algorithm converges toward critical points regardless of the initial iterate. In particular, for smooth cost functions on compact manifolds (which is the case here), the conditions are satisfied [1, Thm 4.4, Cor. 4.6]. Furthermore, when the true Hessian is available, the local convergence rate to nondegenerate local optimizers is superlinear [1, Thm 4.14] (quadratic even, if the parameter θ defined below is set to 1, which is typically the case).

The RTR method is an iterative descent method. Similarly to the classical trust-region method, it consists in an outer algorithm (Algorithm 3) which uses an inner algorithm (Algorithm 4) to (approximately) minimize a model of the cost function within a trust-region around the current iterate. Depending on the performance of the inner solve, the outer algorithm decides to accept or reject the proposed step, and possibly decides to increase or reduce the size of the trust-region. It can be thought of as a Newton method with a safeguard. We give a description of the *preconditioned* Riemannian trust-region method.

The inner problem at the current iterate $x \in \mathcal{M}$ is the following:

$$\min_{\substack{\eta \in \mathrm{T}_x\mathcal{M} \\ \|\eta\|_M \leq \Delta}} m_x(\eta) := f(x) + \langle \eta, \operatorname{grad} f(x) \rangle_x + \frac{1}{2} \langle \eta, \operatorname{Hess} f(x)[\eta] \rangle_x , \tag{38}$$

where $m_x \colon \mathrm{T}_x\mathcal{M} \to \mathbb{R}$ is a quadratic model of the *lifted cost* function $f \circ \mathrm{Retraction}_x$ defined on the same space and the M-norm on $\mathrm{T}_x\mathcal{M}$ is defined via the preconditioner as:

$$\|\eta\|_M^2 := \langle \eta, (\operatorname{Precon} f(x))^{-1}[\eta] \rangle_x .$$

See Eq. (34) for the application in this paper. The preconditioner is a positive definite operator supposed to approximate the inverse of the Hessian. The trust-region constraint $\|\eta\|_M \leq \Delta$ corresponds more or less to a bound on the quadratic term in m_x. Another point of view is that we *trust* the quadratic model only in a ball of radius Δ, the ball in question being distorted into an ellipsoid by the preconditioner. Because the lifted cost and the model are both defined over a linear subspace, the classical methods to solve this inner problem are available for the task. Algorithm 4 is the truncated

Algorithm 3 RTR: preconditioned Riemannian trust-region method

Given: $x_0 \in \mathcal{M}$, $0 < \Delta_0 \leq \bar{\Delta}$ and $0 < \rho' < 1/4$
Init: $k = 0$
repeat
  $\eta_k = \mathrm{tCG}(x_k, \Delta_k)$ ▷ solve inner problem (approximately)
  $x_k^+ = \mathrm{Retraction}_{x_k}(\eta_k)$ ▷ candidate next iterate
  $\rho_1 = f(x_k) - f(x_k^+)$ ▷ actual improvement
  $\rho_2 = m_{x_k}(0) - m_{x_k}(\eta_k)$ ▷ model improvement
  if $\rho_1/\rho_2 < 1/4$ **then** ▷ if the model made a poor prediction
    $\Delta_{k+1} = \Delta_k/4$ ▷ reduce the trust region radius
  ▷ if the model is good but the region is too small
  else if $\rho_1/\rho_2 > 3/4$ **and** tCG hit the boundary **then**
    $\Delta_{k+1} = \min(2\Delta_k, \bar{\Delta})$ ▷ enlarge the radius
  else
    $\Delta_{k+1} = \Delta_k$
  end if
  if $\rho_1/\rho_2 > \rho'$ **then** ▷ if the relative decrease is sufficient
    $x_{k+1} = x_k^+$ ▷ accept the step
  else ▷ otherwise
    $x_{k+1} = x_k$ ▷ reject it
  end if
  $k = k + 1$
until a stopping criterion triggers

Steihaug–Toint method (tCG), as championed in [1], based on [14, Alg. 7.5.1]. The resulting (optimal or suboptimal) vector η is retracted to produce a candidate next iterate $x^+ = \mathrm{Retraction}_x(\eta)$, with the guarantee that $m_x(\eta) \leq m_x(0)$ (strictly if x is not a critical point). Algorithm 3 dictates when this candidate is accepted.

As detailed in the notes following [14, Alg. 7.5.1], it is never necessary to apply the inverse of the preconditioner in practice to compute the M-norm: access to $\mathrm{Precon}\, f(x)$ as a black box is sufficient.

RTR requires three parameters. The step acceptance threshold ρ' is set to 0.1 by default. The other two are the maximum and initial trust-region radii, respectively $\bar{\Delta}$ and Δ_0. The trust-region radius at a given iterate is the upper bound on the M-norm of acceptable steps—see Eq. (38).

The two parameters for the tCG algorithm are θ and κ (see [1]), which we set to 1 and 0.1 respectively by default. These serve in the stopping criterion of tCG. Setting $\theta = 1$ forces a locally quadratic convergence rate for RTR when the true Hessian is available. The number of inner iterations can be limited too.

We note that, close to convergence, the ratio ρ_1/ρ_2 becomes challenging to evaluate accurately, given that both numbers become small and ρ_1 is obtained as the difference

Algorithm 4 tCG(x, Δ): Steihaug–Toint truncated CG method

Given: $x \in \mathcal{M}$ and $\Delta, \theta, \kappa > 0$
Init: $\eta_0 = 0 \in \mathrm{T}_x\mathcal{M}, r_0 = \operatorname{grad} f(x), z_0 = \operatorname{Precon} f(x)[r_0], \delta_0 = -z_0$
for $k = 0 \ldots \max$ inner iterations $- 1$ **do**
  $\kappa_k = \langle \delta_k, \operatorname{Hess} f(x)[\delta_k] \rangle$
  $\alpha_k = \langle z_k, r_k \rangle / \kappa_k$
  if $\kappa_k \leq 0$ **or** $\|\eta_k + \alpha_k \delta_k\|_M \geq \Delta$ **then**
    ▷ the model Hessian has negative curvature or TR exceeded:
    Set τ to be the positive root of $\|\eta_k + \tau\delta_k\|_M^2 = \Delta^2$,
    as in [14, Eqs. (7.5.5)–(7.5.7)]
    $\eta_{k+1} = \eta_k + \tau\delta_k$ ▷ hit the boundary
    return η_{k+1}
  end if
  $\eta_{k+1} = \eta_k + \alpha_k \delta_k$
  $r_{k+1} = r_k + \alpha_k \operatorname{Hess} f(x)[\delta_k]$
  if $\|r_{k+1}\| \leq \|r_0\| \cdot \min(\|r_0\|^\theta, \kappa)$ **then**
    return η_{k+1} ▷ this approximate solution is good enough
  end if
  $z_{k+1} = \operatorname{Precon} f(x)[r_{k+1}]$
  $\beta_k = \langle z_{k+1}, r_{k+1} \rangle / \langle z_k, r_k \rangle$
  $\delta_{k+1} = -z_{k+1} + \beta_k \delta_k$
end for
return η_{last}

between two possibly large numbers. Manopt uses heuristics such as the ones proposed in [14, §17.4.2] to address this issue.

In Section 5, we use RTR with and without preconditioner, and call the resulting methods RTRMC 2p and RTRMC 2, respectively. The number 2 refers to the usage of second-order information. To keep things proportioned when talking about both the preconditioned and the unpreconditioned variants, let

$$s^2 = \lambda_{\max}(c^2 W_{U_0} W_{U_0}^\top)$$

when the preconditioner is used, and let $s = 1$ otherwise. That is: s is the 2-norm of the approximation of the square root of the Hessian underlying the preconditioner. Since the Grassmann manifold is compact, it makes sense to choose the maximum trust-region radius $\bar{\Delta}$ in proportion to the diameter of this manifold, i.e., the largest geodesic distance between any two points on $\mathrm{Gr}(m, r)$. The distance between two subspaces is $\sqrt{\theta_1^2 + \cdots + \theta_r^2}$, where the θ_i's are the principal angles between these spaces. Since these angles are bounded by $\pi/2$, we set $\bar{\Delta} = s\pi\sqrt{r}/2$. Accordingly, we set the initial trust-region radius as $\Delta_0 = \bar{\Delta}/8$.

The number of inner iterations for the tCG algorithm (inner solve) is limited to 500. While this limit may jeopardize the quadratic convergence rate of RTR, we find that it is seldom (if ever) reached for reasonably well-conditioned problems, and otherwise prevents excessive running times.

All other parameters are set to their default values.

As a means to investigate the role of second-order information in this algorithm, we also experiment with RTRMC 1, which is the same method but the Hessian is "approximated" by the identity matrix. The analysis of the RTR method still guarantees global convergence for this setup.

5. Numerical experiments on synthetic data

The proposed algorithms are tested on synthetic data and compared against GROUSE [4], LMaFit [42], LRGeom [41], qGeomMC-CG [31] and ScGrass-CG [33] in terms of accuracy and computation time.

All algorithms are run sequentially by Matlab on the same computer.[2] This computer has 12 cores. Even though none of the tested codes are explicitly multithreaded, some of them get some mileage out of the multicore architecture owing to Matlab's built-in parallelization of some tasks. All Matlab implementations except GROUSE call subroutines in C-Mex code to efficiently deal with the sparsity of the matrices involved. All of these C-Mex codes are single-threaded.

We significantly enhanced the implementation of GROUSE by implementing the rank-one updates it performs using a C-Mex file in Matlab. This file calls the BLAS level 2 routine `dger` directly. In our experience, GROUSE's performance is sensitive to its step size parameter. After experiments on a wide range of values, we decided to set it to 0.3 or 0.5, whichever performs best.

The proposed methods (RTRMC 2 and 2p, RCGMC and RCGMCp) as well as the competing methods GROUSE, LRGeom, qGeomMC-CG and ScGrass-CG require knowledge of the target rank r. LMaFit includes a mechanism to guess the rank, but benefits from knowing it, hence we provide the target rank to LMaFit too.

Note that all these algorithms could be parallelized but this was not the focus of their authors, nor is it ours. We thus did not compare with parallel implementations. Readers interested in parallel algorithms for low-rank matrix completion are encouraged to refer to Jellyfish [35] and the divide-and-conquer approach of Mackey et al. [27], for example.

Remark 5.1 *(About guessing the rank).* If one over-estimates the rank, the factorization UW_U results in a rank deficient factor W_U. This is detectable by monitoring the condition number of the $r \times r$ matrix $W_U W_U^\top$. If this number becomes too large, r could be reduced. If one under-estimates the rank, the algorithm is expected to converge toward a lower-rank approximation of the target matrix, which typically is the desired

[2] HP DL180 + Intel Xeon X5670 2.93 GHz (12 core), 144Go RAM, Matlab 7.10 (R2010a), Linux (64 bits).

behavior—see Scenario 7 below. Guessing strategies for the rank (some of them rather refined) have been proposed that can be used with any fixed-rank matrix completion algorithm [42,13]. It has also been shown that starting from a rank 1 approximation and iteratively increasing the rank until no significant improvement is detected can be beneficial, see for example the so-called homotopy strategy in [41]. All algorithms tested here could be adapted to work iteratively with increasing rank.

We use the root mean square error (RMSE) criterion to assess the quality of reconstruction of X with $\hat{X}$ (note that this is a total error, on all of X):

$$\mathrm{RMSE}(X, \hat{X}) = \|X - \hat{X}\|_{\mathrm{F}}/\sqrt{mn}.$$

This quantity is cheap to compute when the target matrix is given in factored low-rank form $X = AB$ and $\hat{X}$ is (by construction) in the same form $\hat{X} = UW$. The RMSE may then be computed accurately in $\mathcal{O}((m+n)r^2)$ flops observing that $AB - UW = [\,A \quad U\,]\,[\,B^\top \quad -W^\top\,]^\top$. Computing thin (rank $2r$) QR decompositions of both terms as $Q_1R_1 = [\,A \quad U\,]$ and $Q_2R_2 = [\,B^\top \quad -W^\top\,]$ yields the following formula: $\|AB - UW\|_{\mathrm{F}} = \left\|Q_1R_1R_2^\top Q_2^\top\right\|_{\mathrm{F}} = \left\|R_1R_2^\top\right\|_{\mathrm{F}}$. This is much more accurate than the algorithm we used previously in [8]. For the purpose of comparing the algorithms, we add code to all implementations so that the RMSE is computed at each iterate. The time spent in this calculation is discounted from the reported timings.

A number of factors intervene in the difficulty of a low-rank matrix completion task. Obviously, the *size* $m \times n$ of the matrix X to recover and its *rank* r are fundamental quantities. Among others, the presence or absence of *noise* is important. If the observations X_Ω are noisy, then X_Ω is not the masked version of a low-rank matrix X, but of a matrix which is close to being low-rank: $X + \text{noise}$. Of course, different noise distributions (with and without outliers etc.) can be investigated. The search space—the manifold of $m \times n$ matrices of rank r—has dimension $d = r(m+n-r)$. The *oversampling ratio* k/d is a crucial quantity: the larger it is, the easier the task is. The *sampling process* also plays a role in the difficulty of matrix completion. Under uniform sampling for example, all entries of the matrix X have identical probability of being observed. Uniform sampling prevents pathological cases (where some rows or columns have no observed entry at all for example) from happening with high probability. Real datasets often have nonuniform samplings. For example, some movies are particularly popular and some users rate particularly many movies. Finally, the *conditioning* of the low-rank matrix X (the ratio of its largest to its smallest nonzero singular values) may affect the difficulty of matrix completion too.

The following numerical experiments explore these various pitfalls. For the noiseless scenarios, in our methods, we let $\lambda = 0$ (no regularization). All observed entries are trusted with the same confidence $C_{ij} = 1$. For the preconditioner, let $c = 1$ (the average confidence). In practice, because it is numerically convenient, we scale the whole cost function by $1/k$.

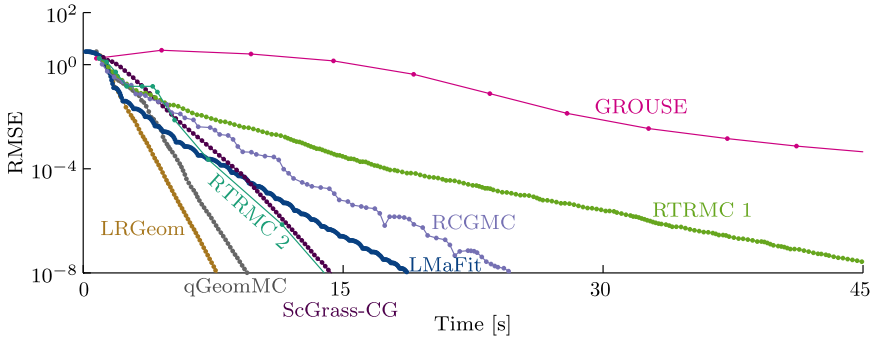

Fig. 1. Scenario 1: standard completion task on a square $10\,000 \times 10\,000$ matrix of rank 10, with a low oversampling ratio of 3, that is, 99.4% of the entries are unknown. Most methods perform well. As the test is repeated, the ranking of the top-performing algorithms varies a little. Since the problem is well-conditioned, the preconditioned variants of our algorithms perform almost the same as the standard algorithms. They are omitted for legibility.

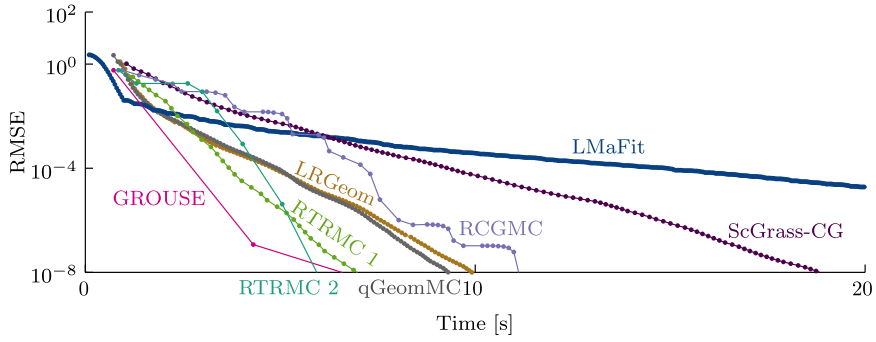

Fig. 2. Scenario 2: completion task on a rectangular matrix of size $1000 \times 30\,000$ of rank 5, with an oversampling ratio of 5. For rectangular matrices, RTRMC and GROUSE are especially efficient since they optimize over a single Grassmann manifold. As a consequence, the dimension of their nonlinear search space grows linearly in $\min(m, n)$, whereas for most methods the growth is linear in $m + n$. Preconditioned variants perform about the same and are omitted for legibility.

Scenario 1: low oversampling ratio. We first compare the convergence behavior of the different methods with square matrices $m = n = 10\,000$ and rank $r = 10$. We generate $A \in \mathbb{R}^{m \times r}$ and $B \in \mathbb{R}^{r \times n}$ with i.i.d. normal entries of zero mean and unit variance. The target matrix is $X = AB$. We sample $3d$ entries uniformly at random without noise, which yields a sampling ratio of 0.6%. This is fairly low. Fig. 1 is typical and shows the evolution of the RMSE as a function of time. Most modern methods are efficient on such a standard task.

Scenario 2: rectangular matrices. In this second test, we repeat the previous experiment with rectangular matrices: $m = 1000$, $n = 30\,000$, $r = 5$ and a sampling ratio of 2.6% ($5d$ known entries), see Fig. 2. We expect and confirm that RTRMC, RCGMC and GROUSE perform well on rectangular matrices since they optimize over the smallest of either the column space or the row space, and not both at the same time.

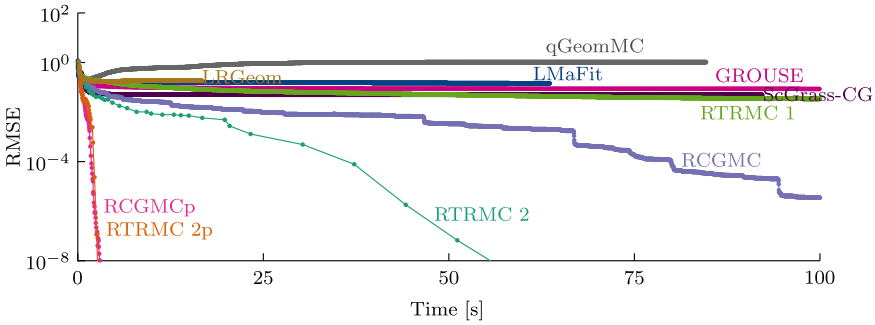

Fig. 3. Scenario 3: completion task on a square 1000×1000 matrix of rank 10 with an oversampling factor of 5 and a condition number of about 150. RTRMC 2, using more Hessian applications than on better conditioned problems, shows good convergence quality. Our preconditioned algorithms RCGMCp and RTRMC 2p perform even better. Surprisingly, ScGrassCG and qGeomMC, which both are (in a slightly different way) preconditioned too, sometimes solve the problem too, but less often than not and require more time.

Scenario 3: bad conditioning. For this third test, we generate A and B as in Scenario 1 with $m = n = 1000$, $r = 10$. We then compute the thin SVD of the product $AB = USV^\top$, which can be done efficiently using economic QR factorizations of A and B separately. The diagonal $r \times r$ matrix S is replaced with a diagonal matrix S_+ whose diagonal entries decay exponentially as follows: $(S_+)_{ii} = \sqrt{mn}\exp(-5(i-1)/(r-1))$, for $i = 1 \ldots r$. The product $X = US_+V^\top$ is then formed and this is the rank-r target matrix, of which we observe $5d$ entries uniformly at random (that is about 10%). Notice that X has much worse conditioning[3] ($e^5 \approx 148$) than the original product AB (which has a typical conditioning below 2) without being unrealistically bad. From the numerical results in Fig. 3, it appears that most methods have difficulties solving this task, while our preconditioned algorithms RTRMC 2p and RCGMCp quickly solve them to high accuracy. RTRMC 2 also succeeds, at the cost of more Hessian evaluations than before.

We venture an explanation of the better performance of the proposed second-order and preconditioned methods here by studying the condition number of the Hessian of the cost function f at the solution $\mathrm{col}(U)$—see Fig. 4. This Hessian is a symmetric linear operator on the linear subspace $\mathrm{T}_U \mathrm{Gr}(m,r)$ of dimension $r(m-r) = 9900$. For the present experiment (target $US_+V^\top$), we compute the 9900 associated eigenvalues with Matlab's `eigs`. They are all positive. The condition number of the Hessian is 72 120. The fact that the bad conditioning of X translates into even worse conditioning of the Hessian at the solution can be explained by the approximate expression for the Hessian, $c^2 W_U W_U^\top$ (Section 3.4). Then, at the solution U, such that $X = UW_U$,

$$\mathrm{cond}(\mathrm{Hess}\, f(U)) \approx \mathrm{cond}(W_U W_U^\top) = \mathrm{cond}^2(X). \tag{39}$$

As seen from Fig. 4, the preconditioner nicely reduces the condition number of the Hessian to 7.3 by an appropriate change of variable.

[3] Here, the conditioning of a nonzero matrix M refers to the ratio between the largest and the smallest *positive* singular values of M.

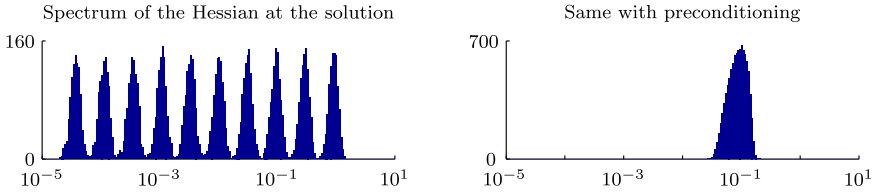

Fig. 4. Spectrum (in $\log_{10}$) of the Hessian and of the preconditioned Hessian at the solution of Scenario 3. The target matrix has a condition number of about 150. This translates into a challenging Hessian condition number of more than 72 000. Preconditioning the Hessian controls this condition number back to 7.3, explaining the success of RTRMC 2p and RCGMCp on Scenario 3. Notice that the spectrum of the Hessian has 10 modes and $r = 10$.

Fig. 5. Proposed nonuniform sampling density for Scenario 4. This image represents a $1000 \times 10\,000$ matrix. Each entry is colored on a grayscale. The lighter the color, the slimmer the chances that this entry is observed. We see that entries in the top 100 rows are much less likely to be observed than in the bottom 100 rows. Columns on the right are also more densely sampled than columns on the left. This artificial sampling process mimics a situation where some objects are more popular than others (and hence more often rated) and some raters are more active than others.

Scenario 4: nonuniform sampling. As a fourth test, we generate A and B as in Scenario 1 with $m = 1000$, $n = 10\,000$ and $r = 10$. The target matrix is $X = AB$, of which we observe entries with a *nonuniform* distribution. The chosen artificial sampling mimics a situation where rows correspond to movies and columns correspond to raters. Some of the movies are much more often rated than others, and some of the raters rate many more movies than others. Each of the 100 first movies (they are the least popular ones) has a probability of being rated that is 5 times smaller than the 800 following movies. The 100 last movies are 5 times as likely to be rated as the 800 latter (they are the popular ones). Furthermore, each rater rates between 15 and 50 movies, uniformly at random, resulting in an oversampling ratio of 2.94 (3.2%). Fig. 5 shows the associated mask probability, where raters (columns) have been sorted by number of given ratings.

Fig. 6 shows the behavior of the various methods tested on this instance of the problem. It appears that the proposed algorithms can cope with some non-uniformity in the sampling procedure.

Scenario 5: larger instances. In this fifth test, we try out the various algorithms on a larger instance of matrix completion: $m = 10\,000$, $n = 100\,000$, $r = 20$ with oversampling ratio of 5, that is, 1.1% of the entries are observed, sampled uniformly at random. The target matrix $X = AB$ (formed as previously) has a billion entries. Fig. 7 shows that RTRMC 2(p) performs well on such instances.

Scenario 6: noisy observations. As a sixth test, we try out RTRMC on a class of noisy instances of matrix completion with $m = n = 5000$, $r = 10$ and oversampling ratio of 4, that is, 1.6% of the entries are observed, sampled uniformly at random. The target

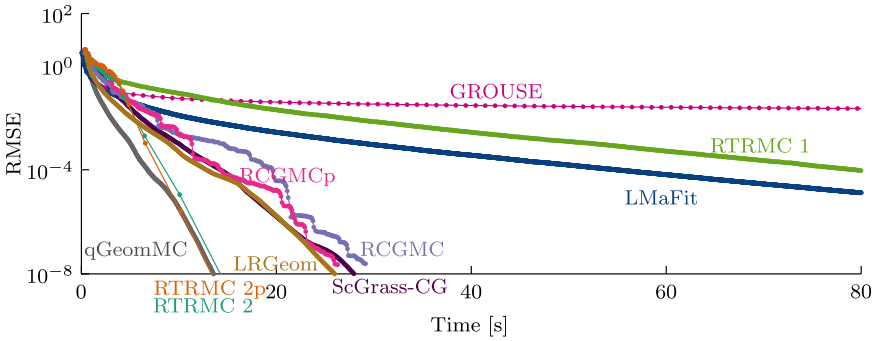

Fig. 6. Scenario 4: completion task on a rectangular $1000 \times 10\,000$ matrix or rank 10, with 3.2% (OS = 2.9) of the entries revealed following a nonuniform sampling as depicted in Fig. 5. It appears most methods can withstand some non-uniformity. GROUSE is slowed down by the non-uniformity, possibly because it operates one column at a time.

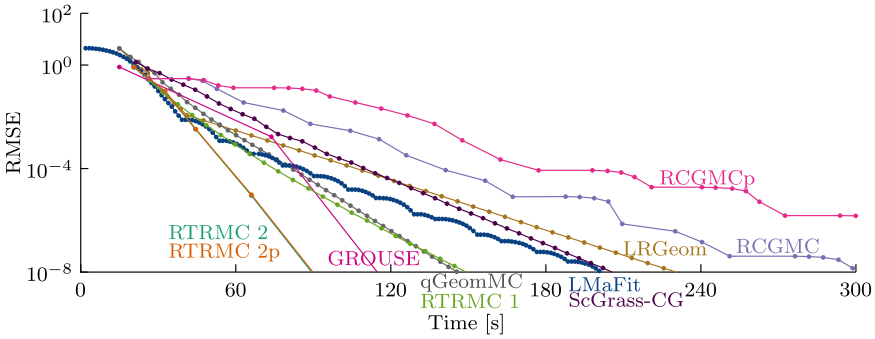

Fig. 7. Scenario 5: completion task on a larger $10\,000 \times 100\,000$ matrix of rank 20 with an oversampling ratio of 5.

matrix $X = AB$ is formed as before with $A \in \mathbb{R}^{m \times r}$, $B \in \mathbb{R}^{r \times n}$ whose entries are i.i.d. normal random variables. Notice that this implies the X_{ij}'s are also zero-mean Gaussian variables but with variance r and not independent. We then generate a noise matrix N_Ω, such that the $(N_\Omega)_{ij}$'s for (i, j) in Ω are i.i.d. normal random variables (Gaussian distribution with zero mean, unit variance), and the other entries of N are zero. The observed matrix is $X_\Omega + \sigma N_\Omega$, where σ^2 is the noise variance. The signal to noise ratio (SNR) is thus r/σ^2. This is the same setup as the standard scenario in [24].

All algorithms based on a least-squares strategy should perform rather well on this scenario, since least-squares are particularly well-suited to filter out Gaussian noise. And indeed, as they perform almost the same, we only show results for RTRMC 2p, see Fig. 8. For comparison, we use the same oracle as in [24], that is: we compare the RMSE obtained by RTRMC with the RMSE we could obtain if we knew the column space $\mathrm{col}(X)$. This is known to be equal to $\mathrm{RMSE}_{\mathrm{oracle}} = \sigma\sqrt{(2nr - r^2)/k}$ (in expectation). Fig. 8 illustrates the fact that, not surprisingly, RTRMC reaches almost the same RMSE as the oracle as soon as the SNR is large enough.

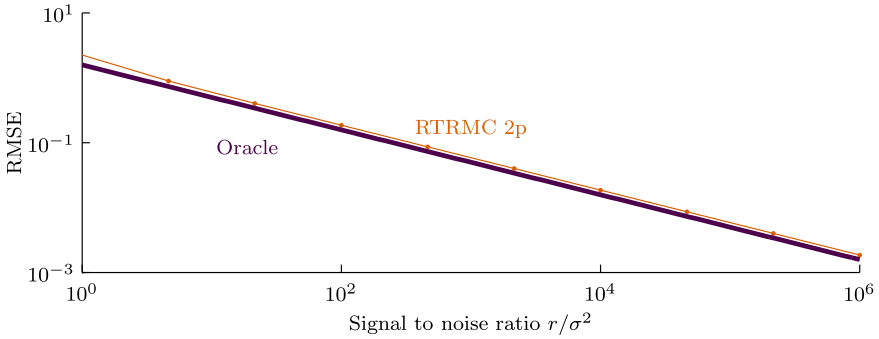

Fig. 8. Scenario 6: RTRMC is well suited to solve matrix completion tasks under Gaussian noise, owing to its least-squares objective function ($m = n = 5000$, $r = 10$, $|\Omega|/(mn) = 1.6\%$). The thick dark line indicates the expected RMSE that an oracle who knows the column space of the target matrix X would reach (this is a lower bound on the performance of any practical algorithm). For different values of SNR, the thin line reports the RMSE reached by RTRMC 2p, averaged over 10 problem instances. The performance is close to the oracle quality and seems to vary at the same rate, which shows that Gaussian noise is easily filtered out.

Scenario 7: underestimating the rank. As a seventh test, the target matrix X has true rank 32, with positive singular values decaying exponentially from $\sqrt{mn}$ to $\sqrt{mn} \times 10^{-10}$ (see Scenario 3). We challenge the algorithms to reconstruct a matrix of rank 8 which best approximates X. Such a scenario is motivated in [41], in the context of approximating almost separable functions of two variables.

The size of X is given by $m = n = 5000$ and $10d$ entries are observed uniformly at random, that is, 3.2%. An oracle knowing the matrix X perfectly would simply return the SVD of X truncated to rank 8, committing an RMSE of 3×10^{-3}. In repeated realizations of this test, the only methods we observed converging close to the oracle bound are preconditioned methods, see Fig. 9.

6. Application to the Netflix dataset

The competing algorithms are now tested on the Netflix dataset [5]. The complete matrix has size $17\,770 \times 480\,189$ with $99\,072\,112$ entries revealed for training (that is, these are given to the algorithms to build the low-rank model) and $1\,408\,395$ additional entries reserved for testing (that is, these are only used after the model was built, to assess its quality). Each row corresponds to a movie and each column corresponds to a user. Entries are integer ratings between 1 and 5. The training data is centered around the mean of the training entries (3.6033), since the regularized cost function (19) implicitly puts a prior on the value of each unknown entry being 0.

On this larger dataset, we found it useful to compute the initial guess U_0 with a permissive tolerance when calling `svds` in Matlab. Namely, we set the (relative) tolerance to 0.1. We use RCGMCp to produce Table 2, where various values for the regularization λ as well as the reconstruction rank r are tested. From this table, it appears that, for our algorithms, regularization is necessary.

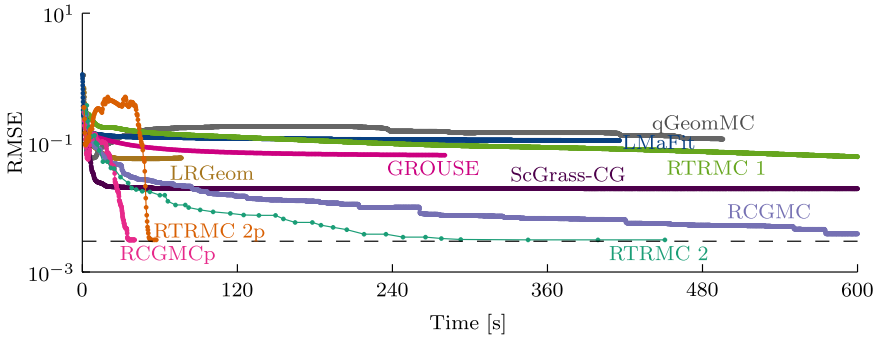

Fig. 9. Scenario 7: in this challenging completion task, the target matrix is square, 5000 × 5000, and has rank 32 with singular values decaying exponentially from $\sqrt{mn}$ to $\sqrt{mn} \times 10^{-10}$. The various algorithms attempt to construct a matrix of rank 8 best approximating the ill-conditioned target matrix, based on 3.2% of revealed entries (OS = 10). The best methods almost reach the oracle RMSE (dashed line), which corresponds to the RMSE reached by an SVD of the true matrix X truncated to rank 8. The outcome of this test is less constant than the others. Over many runs, the typical result is that RCGMCp and RTRMC 2p almost always converge as depicted, while qGeomMC and ScGrass-CG (the only two other preconditioned methods in this test) sometimes converge as well, but less often than not. RTRMC 2 and RCGMC sometimes achieve good reconstructions too (as depicted) but are always much slower than their preconditioned counterparts. Over many runs, we did not witness the other methods reach values close to the oracle bound.

Table 2
Test RMSE of the initial guess, then best test RMSE reached by RCGMCp on the Netflix dataset, with various values of the regularization parameter λ and of the reconstruction rank r. Reported timings include the computation of the initial guess (with low `svds` tolerance) and of the RCGMCp iterations. A star indicates a forced stop. A first conclusion is that regularization is necessary on this dataset with our method. Another conclusion, looking at the stagnating RMSE going from rank 15 to 20, is that aiming at large rank reconstructions "from scratch" may not be efficient. This suggests looking at incremental rank procedures such as the ones described in [41].

Rank $r = 10$

Regularization λ	0.001	0.01	0.1	0.2	1
RMSE (initial)	2.762	1.221	1.023	1.027	1.090
RMSE	1.183	0.987	0.963	0.994	1.091
Time [min]	101*	101*	11	7	5

Regularization $\lambda = 0.1$

Rank r	1	2	5	10	15	20
RMSE (initial)	1.089	1.052	1.030	1.026	1.018	1.019
RMSE	1.071	1.013	0.976	0.964	0.962	0.962
Time [min]	3	2	4	8	23	47

Fig. 10 shows the behavior of the various algorithms considered, with test RMSE as a function of elapsed computation time. The test RMSE is computed over the test entries only, not over the training entries. The rank is set to $r = 10$ (corresponding to an oversampling factor of 19.9). The regularization parameter λ of our algorithms is set to 0.1. It is interesting to note that the preconditioner for the proposed methods improves the performance.

Unfortunately, we could not obtain results with GROUSE, as NaN's are produced at the first iteration.

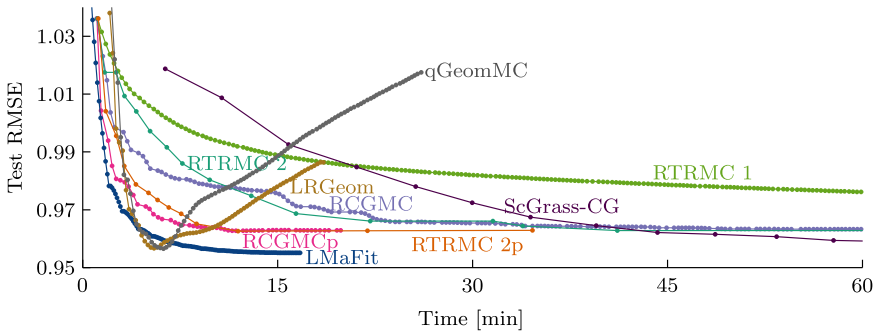

Fig. 10. Convergence of our algorithms on the Netflix dataset: $17\,770 \times 480\,189$ with $k = 99\,072\,112$ known ratings. The test RMSE is evaluated on $1\,408\,395$ separate test ratings (not used for training). The algorithms aim for a rank 10 fitting of the data, with regularization parameter $\lambda = 0.1$ for our algorithms. Note how preconditioning helps our algorithms on this real dataset. The other algorithms, which work well without added regularization, attain better RMSE's. LRGeom and qGeomMC seem to overfit as a result, and so does ScGrass-CG after a longer time.

The best algorithms on this dataset appear to be LMaFit, LRGeom and qGeomMC: they attain test RMSE's of, respectively, 0.955, 0.957 and 0.957. The latter two do so in about 7 minutes. ScGrass-CG also attains this RMSE after 85 minutes, then its performance degrades slowly, similarly to LRGeom and qGeomMC. Interestingly, none of these algorithms use regularization (besides imposing a rank of $r = 10$). This explains why they reach a different RMSE value than the proposed methods, and could explain why (except for LMaFit) they seem to exhibit overfitting (the RMSE goes up after reaching its minimum). This observation also triggers the question: why is regularization necessary for RTRMC/RCGMC (and, possibly, also for GROUSE), which are the only tested algorithms who optimize solely over the column space Grassmannian?

A baseline RMSE to compare with is the one reached by an algorithm which simply returns the average training rating. This RMSE is 1.130. The RMSE reached by RCGMCp for rank 10, $\lambda = 0.1$, is 0.963. The best RMSE reached in Fig. 10 is 0.955. From the literature on the Netflix competition, it is known that plain low-rank approximation is not sufficient to reach ideal scores, although it can provide an important basis for better predictors. For example, temporal information (how recent a rating is) should be taken into account. Furthermore, it is well-known that least-squares loss functions (used by all methods tested here) are not ideal for outlier rejection. Nevertheless, the positive impact of preconditioning for our methods on this real dataset is interesting to note.

7. Conclusions

The contribution of this paper is a regularized cost function on a single Grassmann manifold for low-rank matrix completion, along with a set of practical numerical methods to minimize it: RTRMC 2(p) and RCGMC(p). These are respectively second-order Riemannian trust-region methods and Riemannian conjugate gradient methods, with or without preconditioning. We detailed these algorithms in the general setting of optimiza-

tion on manifolds. These algorithms compete with the state-of-the-art. The trust-region methods further enjoy global convergence to critical points, with a quadratic local convergence rate for RTRMC 2(p).

The methods we propose are efficient on rectangular matrices. We believe this is because the dimension of the nonlinear search space, $\mathrm{Gr}(m, r)$, grows as $\min(m, n)$, whereas for most competing methods the growth is in $m + n$. We also observe that second-order and preconditioned methods perform better than first-order methods when the matrix to complete is badly conditioned. We believe this is because the bad conditioning of the target matrix translates into an even worse conditioning of the Hessian of the cost function at the solution, as shown in the numerical experiments (39). Furthermore, the proposed algorithms can withstand low oversampling ratios or non-uniformity in the sampling process. The proposed cost function is effective against Gaussian noise, which is not surprising given its least-squares nature. Those combined strengths make RTRMC and RCGMC reasonably efficient on the Netflix dataset too, where the proposed preconditioner has a noteworthy positive impact.

If speed is of essence, note that all the proposed methods could be parallelized. Indeed, the expensive operations involved in computing the cost and its derivatives are inherently parallelizable over the columns.

Matlab code for RTRMC 2(p) and RCGMC(p) is available at:
http://sites.uclouvain.be/absil/RTRMC/.

Acknowledgements

The authors thank Bart Vandereycken and Bamdev Mishra for useful discussions. This research was conducted while N.B. was a research fellow with the FNRS (aspirant) at UCLouvain. This paper presents research results of the Belgian Network DYSCO (Dynamical Systems, Control, and Optimization), funded by the Interuniversity Attraction Poles Programme initiated by the Belgian Science Policy Office. It was financially supported by the Belgian FRFC.

References

[1] P.-A. Absil, C.G. Baker, K.A. Gallivan, Trust-region methods on Riemannian manifolds, Found. Comput. Math. 7 (3) (2007) 303–330, http://dx.doi.org/10.1007/s10208-005-0179-9.
[2] P.-A. Absil, R. Mahony, R. Sepulchre, Optimization Algorithms on Matrix Manifolds, Princeton University Press, Princeton, NJ, ISBN 978-0-691-13298-3, 2008.
[3] P.-A. Absil, L. Amodei, G. Meyer, Two Newton methods on the manifold of fixed-rank matrices endowed with Riemannian quotient geometries, Comput. Statist. 29 (3–4) (2014) 569–590, http://dx.doi.org/10.1007/s00180-013-0441-6.
[4] L. Balzano, R. Nowak, B. Recht, Online identification and tracking of subspaces from highly incomplete information, in: 48th Annual Allerton Conference on Communication, Control, and Computing, Allerton, 2010, IEEE, 2010, pp. 704–711.
[5] J. Bennett, S. Lanning, The Netflix prize, in: Proceedings of the KDD Cup Workshop 2007, ACM, New York, August 2007, pp. 3–6.
[6] W.M. Boothby, An Introduction to Differentiable Manifolds and Riemannian Geometry, Pure Appl. Math., vol. 120, Elsevier, ISBN 978-0121160531, 1986.

[7] N. Boumal, Optimization and estimation on manifolds, PhD thesis, Université catholique de Louvain, Louvain-la-Neuve, Belgium, 2014.
[8] N. Boumal, P.-A. Absil, RTRMC: a Riemannian trust-region method for low-rank matrix completion, in: J. Shawe-Taylor, R.S. Zemel, P. Bartlett, F.C.N. Pereira, K.Q. Weinberger (Eds.), Advances in Neural Information Processing Systems 24, NIPS, 2011, pp. 406–414.
[9] N. Boumal, B. Mishra, P.-A. Absil, R. Sepulchre, Manopt, a Matlab toolbox for optimization on manifolds, J. Mach. Learn. Res. 15 (2014) 1455–1459, http://www.manopt.org.
[10] J.F. Cai, E.J. Candès, Z. Shen, A singular value thresholding algorithm for matrix completion, SIAM J. Optim. 20 (4) (2010) 1956–1982, http://dx.doi.org/10.1137/080738970.
[11] E.J. Candès, B. Recht, Exact matrix completion via convex optimization, Found. Comput. Math. 9 (6) (2009) 717–772, http://dx.doi.org/10.1007/s10208-009-9045-5.
[12] V. Chandrasekaran, S. Sanghavi, P.A. Parrilo, A.S. Willsky, Rank-sparsity incoherence for matrix decomposition, SIAM J. Optim. 21 (2) (2011) 572–596, http://dx.doi.org/10.1137/090761793.
[13] S. Chatterjee, Matrix estimation by universal singular value thresholding, Ann. Statist. 43 (1) (2015) 177–214.
[14] A.R. Conn, N.I.M. Gould, P.L. Toint, Trust-Region Methods, MPS/SIAM Ser. Optim., Society for Industrial and Applied Mathematics, ISBN 978-0-89871-460-9, 2000.
[15] W. Dai, O. Milenkovic, E. Kerman, Subspace evolution and transfer (SET) for low-rank matrix completion, IEEE Trans. Signal Process. 59 (7) (2011) 3120–3132, http://dx.doi.org/10.1109/TSP.2011.2144977.
[16] W. Dai, E. Kerman, O. Milenkovic, A geometric approach to low-rank matrix completion, IEEE Trans. Inform. Theory 58 (1) (2012) 237–247, http://dx.doi.org/10.1109/TIT.2011.2171521.
[17] J.C. Gilbert, J. Nocedal, Global convergence properties of conjugate gradient methods for optimization, SIAM J. Optim. 2 (1) (1992) 21–42, http://dx.doi.org/10.1137/0802003.
[18] N. Gillis, The why and how of nonnegative matrix factorization, in: J.A.K. Suykens, M. Signoretto, A. Argyriou (Eds.), Regularization, Optimization, Kernels, and Support Vector Machines, in: Mach. Learn. Pattern Recogn. Ser., Chapman and Hall/CRC, ISBN 9781482241396, 2014, pp. 257–291.
[19] N. Gillis, F. Glineur, Low-rank matrix approximation with weights or missing data is NP-hard, SIAM J. Matrix Anal. Appl. 32 (4) (2011) 1149–1165, http://dx.doi.org/10.1137/110820361.
[20] W.W. Hager, H. Zhang, A survey of nonlinear conjugate gradient methods, Pac. J. Optim. 2 (1) (2006) 35–58.
[21] M. Hardt, Understanding alternating minimization for matrix completion, in: IEEE 55th Annual Symposium on Foundations of Computer Science, FOCS, 2014, IEEE, Oct 2014, pp. 651–660.
[22] R. Kannan, M. Ishteva, H. Park, Bounded matrix factorization for recommender system, Knowl. Inf. Syst. 39 (2014) 491–511, http://dx.doi.org/10.1007/s10115-013-0710-2.
[23] R.H. Keshavan, A. Montanari, Regularization for matrix completion, in: IEEE International Symposium on Information Theory Proceedings, ISIT, 2010, IEEE, 2010, pp. 1503–1507.
[24] R.H. Keshavan, A. Montanari, S. Oh, Matrix completion from noisy entries, J. Mach. Learn. Res. 99 (2010) 2057–2078.
[25] K. Lee, Y. Bresler, ADMiRA: atomic decomposition for minimum rank approximation, IEEE Trans. Inform. Theory 56 (9) (2010) 4402–4416, http://dx.doi.org/10.1109/TIT.2010.2054251.
[26] K. Leichtweiss, Zur Riemannschen Geometrie in Grassmannschen Mannigfaltigkeiten, Math. Z. 76 (1961) 334–366, http://dx.doi.org/10.1007/BF01210982.
[27] L. Mackey, A. Talwalkar, M.I. Jordan, Divide-and-conquer matrix factorization, in: J. Shawe-Taylor, R.S. Zemel, P. Bartlett, F.C.N. Pereira, K.Q. Weinberger (Eds.), Advances in Neural Information Processing Systems 24, NIPS, 2011, pp. 1134–1142.
[28] G. Meyer, S. Bonnabel, R. Sepulchre, Linear regression under fixed-rank constraints: a Riemannian approach, in: 28th International Conference on Machine Learning, ICML, 2011.
[29] B. Mishra, R. Sepulchre, Riemannian preconditioning, preprint, arXiv:1405.6055, 2014.
[30] B. Mishra, G. Meyer, R. Sepulchre, Low-rank optimization for distance matrix completion, in: 50th IEEE Conference on Decision and Control and European Control Conference, CDC-ECC, 2011, IEEE, 2011, pp. 4455–4460.
[31] B. Mishra, K. Adithya Apuroop, R. Sepulchre, A Riemannian geometry for low-rank matrix completion, preprint, arXiv:1211.1550, 2012.
[32] B. Mishra, G. Meyer, F. Bach, R. Sepulchre, Low-rank optimization with trace norm penalty, SIAM J. Optim. 23 (4) (2013) 2124–2149, http://dx.doi.org/10.1137/110859646.
[33] T. Ngo, Y. Saad, Scaled gradients on Grassmann manifolds for matrix completion, in: F. Pereira, C.J.C. Burges, L. Bottou, K.Q. Weinberger (Eds.), Advances in Neural Information Processing Systems 25, 2012, pp. 1412–1420.

[34] J. Nocedal, S.J. Wright, Numerical Optimization, Springer-Verlag, 1999.
[35] B. Recht, C. Ré, Parallel stochastic gradient algorithms for large-scale matrix completion, Math. Program. Comput. 5 (2) (2013) 201–226, http://dx.doi.org/10.1007/s12532-013-0053-8.
[36] H. Sato, T. Iwai, A new, globally convergent Riemannian conjugate gradient method, Optimization 64 (4) (2015) 1011–1031, http://dx.doi.org/10.1080/02331934.2013.836650.
[37] R. Schneider, A. Uschmajew, Convergence results for projected line-search methods on varieties of low-rank matrices via Łojasiewicz inequality, SIAM J. Optim. (2015), in press, preprint, arXiv:1402.5284, 2014.
[38] M. Tao, X. Yuan, Recovering low-rank and sparse components of matrices from incomplete and noisy observations, SIAM J. Optim. 21 (1) (2011) 57–81, http://dx.doi.org/10.1137/100781894.
[39] K.C. Toh, S. Yun, An accelerated proximal gradient algorithm for nuclear norm regularized linear least squares problems, Pac. J. Optim. 6 (15) (2010) 615–640.
[40] L.N. Trefethen, D. Bau, Numerical Linear Algebra, Society for Industrial and Applied Mathematics, ISBN 978-0898713619, 1997.
[41] B. Vandereycken, Low-rank matrix completion by Riemannian optimization, SIAM J. Optim. 23 (2) (2013) 1214–1236, http://dx.doi.org/10.1137/110845768.
[42] Z. Wen, W. Yin, Y. Zhang, Solving a low-rank factorization model for matrix completion by a nonlinear successive over-relaxation algorithm, Math. Program. Comput. 4 (4) (2012) 333–361, http://dx.doi.org/10.1007/s12532-012-0044-1.