Secrets of Matrix Factorization: Approximations, Numerics, Manifold Optimization and Random Restarts

Je Hyeong Hong
University of Cambridge
jhh37@cam.ac.uk

Andrew Fitzgibbon
Microsoft, Cambridge, UK
awf@microsoft.com

Abstract

Matrix factorization (or low-rank matrix completion) with missing data is a key computation in many computer vision and machine learning tasks, and is also related to a broader class of nonlinear optimization problems such as bundle adjustment. The problem has received much attention recently, with renewed interest in variable-projection approaches, yielding dramatic improvements in reliability and speed. However, on a wide class of problems, no one approach dominates, and because the various approaches have been derived in a multitude of different ways, it has been difficult to unify them. This paper provides a unified derivation of a number of recent approaches, so that similarities and differences are easily observed. We also present a simple meta-algorithm which wraps any existing algorithm, yielding 100% success rate on many standard datasets. Given 100% success, the focus of evaluation must turn to speed, as 100% success is trivially achieved if we do not care about speed. Again our unification allows a number of generic improvements applicable to all members of the family to be isolated, yielding a unified algorithm that outperforms our re-implementation of existing algorithms, which in some cases already outperform the original authors' publicly available codes.

1. Introduction

Many problems in computer vision and machine learning involve finding low-rank factors of a given $m \times n$ matrix $\mathtt{M}$, where some of the values are unobserved or, more generally, are weighted by another $m \times n$ matrix $\mathtt{W}$. This involves minimization of the error function

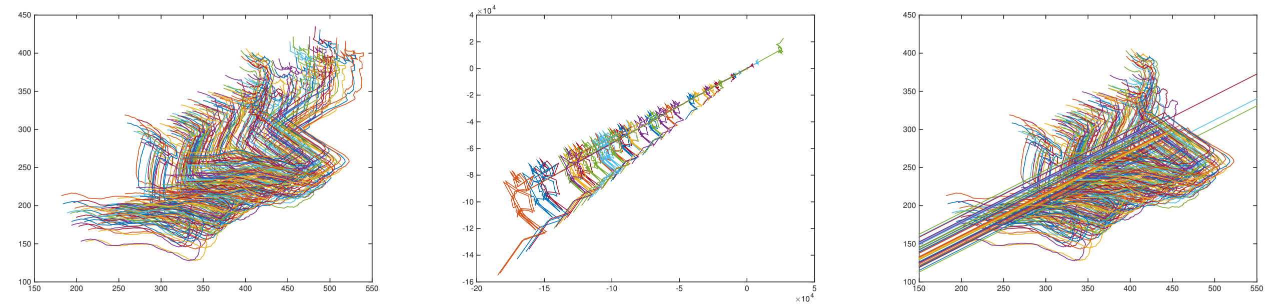
$$f(\mathtt{U},\mathtt{V}) = \|\mathtt{W} \odot (\mathtt{U}\mathtt{V}^\top - \mathtt{M})\|_F^2 + \mu(\|\mathtt{U}\|_F^2 + \|\mathtt{V}\|_F^2) \quad (1)$$

over unknown matrices $\mathtt{U}, \mathtt{V}$ each with rank r. The operator $\odot$ is Hadamard or elementwise product, and the norms are Frobenius. For this paper, we will focus on the L2 problem as stated above which includes the nuclear norm formulation [24, 17, 6]. Investigations on the L1-norm and other variants are for future work. We further note that although the question of choice of hyperparameters r, μ is of great interest, our focus here is on finding the global optimum of (1), assuming the hyperparameters have been chosen.

When the number of nonzero entries of $\mathtt{W}$ is small, and particularly when entries are not missing uniformly, this is a hard problem. In recent years, the revival of algorithms based on variable projection (VarPro) and the Wiberg algorithm has yielded great advances in success rates (that is, percentage of runs from random starting points that reach the global optimum), so that many once-difficult benchmarks are now solved by almost all current algorithms.

For many other benchmarks, however, even the best current algorithms succeed only occasionally, with only a small fraction of runs successful. However, success rates for any algorithm X are easily improved through the use of a meta-algorithm, which we call "RUSSO-X", which simply runs algorithm X from different random starting points until the same best-so-far optimum value is seen twice. (RUSSO-X stands for "Restart X Until Seen Same Optimum".) We show that this procedure dramatically increases success rate. On five of 13 benchmarks, it yields the best known optimum nearly 100% of the time. On other benchmarks, where there are other local optima with large basins of convergence, one might nevertheless hope to choose the best from several runs of RUSSO, and indeed simply repeating it five times brings the number of successes up to 11. The final two benchmarks are not solved by any algorithm we tested.

The natural question then is how to select algorithm X. This must depend on both its success rate and its runtime. For example if X has a 5% success rate and runs in one second, RUSSO-X has an expected runtime of about 44 seconds. If Y has a 95% success rate, and takes an hour, RUSSO-Y will take two hours, so algorithm X is clearly preferable. Thus, the criterion for comparison of candidates must ultimately be mean runtime, and must be *wall-clock time*. Any proxy, such as iteration counts, may not reflect real-world practice. Even floating point operation counts (FLOPs) may not reflect real-world performance due to



(a) Best known minimum (0.3228) (b) Second best solution (0.3230) (c) Second best, zoomed to image

Figure 1: Illustration that a solution with function value just .06% above the optimum can have significantly worse extrapolation properties. This is a reconstruction of point trajectories in the standard "Giraffe" (GIR in Table 4) sequence. Even when zooming in to eliminate gross outliers (not possible for many reconstruction problems where points pass behind the camera), it is clear that numerous tracks have been incorrectly reconstructed.

cache and memory effects, CPU pipelining, etc. Of course, having decided to measure time, we need to know that the algorithms are implemented as efficiently as possible, and that each is taking advantage of all the speedup tricks used by the others. In order to achieve this, we develop a unified derivation of several recent algorithms. The unification process has the epistemological benefit that their similarities and differences are clarified, but also the practical benefit that each algorithm is assembled from common components, for each of which the best performing implementation can be chosen. By doing so, we present new implementations of algorithms which are generally faster and more reliable than the publicly available codes. We also propose some hybrid algorithms which are better again.

The paper's contributions therefore are:

+ A **unified theoretical derivation and analysis** of several existing algorithms.
+ **New re-implementations** of those algorithms which are faster and more reliable.
+ **New implementations** of standard **variable projection (VarPro)** algorithms which offer an order of magnitude performance improvement over the best previously available algorithms.
+ Assessment of the **meta-algorithm** RUSSO which improves success rates over all existing algorithms.
+ Some **new datasets**, including more challenging instances of existing datasets, and new problem classes.

Conversely, there are limitations of the current work: We focus on medium-scale problems, under a million visible entries and tens of thousands of unknowns. While larger scale data have been used (e.g. the Venice dataset [9]), they have been subject only to synthetic missing-data patterns, which correlate poorly with real-life performance. Some of our conclusions will change for larger problems, but it is hoped that our unified derivations will accelerate future research in the large-scale domain.

We do not unify all existing algorithms (although we do run experiments on e.g. augmented Lagrangian methods [9, 6]). Also, our experiments are run on just one architecture.

Generalizations of the Wiberg algorithm [25, 26] are not explicitly treated, although again we hope this analysis will enhance our understanding of those algorithms.

1.1. Do we know the global optimum?

None of the algorithms considered attempt to find global optima, yet it is of interest to determine the extent to which they do in fact do so. We observe that of all the experiments run in the factorization literature on the same standard datasets, the best known optima have been reached many times independently on perhaps hundreds of millions of runs. Furthermore, in our experiments, these best-known optima are the among only a small few that are reached multiple times.

Of course, even though no lower value is known, these might still not be global optima for these standard datasets. The global optimum for a given problem might have a tiny basin of convergence which no random initialization can ever hope to hit. But conversely, if some of the generally-agreed optima *are* just local optima with a very large basin of convergence, then we cannot hope to do better with any currently known methods, so it remains valuable to know how to find such optima reliably and efficiently. Figure 1 shows that it is certainly worth finding the best optimum; it is an example where a local optimum 0.06% above the best known value provides a qualitatively worse solution.

Some datasets in our test *do* have more than one strong local optimum, and for some applications it may be valuable to find more than one, on which there is considerable research [16], beyond our scope here. In these experiments, a non-100% success rate for RUSSO gives a measure of the volume of the basins of convergence of secondary optima. In these benchmarks, this was no larger than 70%, so 3-5 runs of RUSSO does yield the best known optimum.

1.2. Background

It is nearly forty years since Wiberg addressed the problem of principal components analysis in the presence of missing data [30], and twenty since Tomasi and Kanade demonstrated the effectiveness of matrix factorization in 3D reconstruction from images [27]. Then followed numerous approaches to the optimization of (1) based on first order algorithms such as alternating least squares (ALS), an instance of block coordinate descent. While Buchanan and Fitzgibbon [5] showed that damped second-order algorithms could provide significantly improved convergence rates, they also incorrectly classified Wiberg's algorithm as an instance of coordinate descent. Okatani and Deguchi reconsidered Wiberg's algorithm [19] and showed that it too offered second-order convergence rates, and that a damped update further cemented its advantage, leading to dramatic performance improvements over the then state of the art.

Two concepts are key here. First is the Gauss-Newton (GN) algorithm, which make use of the objective's being a sum of squared *residuals*. Writing (1) as the squared norm of a vector function ε yields

$$f(\mathtt{U},\mathtt{V}) = \|\varepsilon(\mathtt{U},\mathtt{V})\|_2^2, \tag{2}$$

and then the GN algorithm approximates the Hessian of f (as used in [5]) by the square of the Jacobian of ε. Further adding a damping term [5, 20] yields the Levenberg-Marquardt algorithm.

The second concept in these recent algorithms [7, 20, 12] is variable projection (VarPro): the problem is solved by eliminating one matrix of larger dimension from the original objective function. Although the derivations of individual algorithms may have been inspired by different sources, they are all related to the approach dubbed variable projection by Golub and Pereyra [11]: in cases where the objective function is bivariate and the cost vector is linear in one block of variables (say V), then perform second-order optimization on a new objective function which eliminates that block:

$$f^*(\mathtt{U}) := \min_{\mathtt{V}} f(\mathtt{U},\mathtt{V}). \tag{3}$$

In (1), the objective is linear in both blocks, so we can choose which to eliminate. We will assume "landscape" matrices where $m < n$, as "portrait" problems can simply be solved transposed, inverting the roles of U and V, so it is always appropriate to eliminate V.

Ruhe and Wedin [23] considered VarPro in the Gauss-Newton scenario, defining a new vector function

$$\varepsilon^*(\mathtt{U}) = \|\varepsilon(\mathtt{U},\mathtt{V}^*(\mathtt{U}))\|_2^2 \tag{4}$$

defined in terms of $\mathtt{V}^*(\mathtt{U}) = \arg\min_{\mathtt{V}} \|\varepsilon(\mathtt{U},\mathtt{V})\|_2^2$. They present three numbered algorithms we call RW1, RW2, RW3. RW1 is the Gauss-Newton solver on the new objective function, using the analytic Jacobian $\frac{\partial \varepsilon}{\partial \operatorname{vec} \mathtt{U}}$. RW2, like Wiberg, approximates this Jacobian by eliminating a term. RW3 is alternation (ALS). Whether using the full Gauss-Newton matrix or its approximation is better is still debated [21], and has been explored previously in the context of matrix factorization [12]. Our work extends these comparisons.

2. Synthesis of current approaches

This section derives several existing approaches in a unified way, including analytic forms of derivatives and other relevant quantities useful for optimization. Some notations that will be used: the space of symmetric $n \times n$ matrices is denoted $\mathbb{S}^n$. The total number of unknowns is $N = mr + nr$. This section depends on several vec and Kronecker product ($\otimes$) identities from [18, 14], which are repeated in [13]. Given the plethora of naming conventions in the literature, we felt it did not reduce complexity to choose one of them, and instead we use mnemonic names: M is the Measurements; W is Weights; and U and V are a pair of unknowns. Finally, some quantities that are used throughout are the $r \times r$ identity matrix $\mathtt{I}_r$, and the constant matrix $\mathtt{K}_{mr}$ which is such that

$$\mathtt{K}_{mr} \operatorname{vec}(\mathtt{U}) = \operatorname{vec}(\mathtt{U}^\top), \tag{5}$$

and the residue matrix R and its transformation Z which are

$$\mathtt{R}(\mathtt{U},\mathtt{V}) := \mathtt{W} \odot (\mathtt{U}\mathtt{V}^\top - \mathtt{M}) \tag{6}$$
$$\mathtt{Z}(\mathtt{U},\mathtt{V}) := (\mathtt{W} \odot \mathtt{R}) \otimes \mathtt{I}_r. \tag{7}$$

When the above occur "starred", e.g. $\mathtt{R}^*$, they are a function of U only i.e. $\mathtt{R}^*(\mathtt{U}) := \mathtt{R}(\mathtt{U},\mathtt{V}^*(\mathtt{U}))$, and when applied to vector arguments, e.g. $\mathtt{R}^*(\mathbf{u})$, they reshape the arguments. A tilde over a variable, such as $\tilde{\mathtt{U}}, \tilde{\mathtt{W}}$ is mnemonic: the variable's definition is a sparse matrix whose nonzero entries are (weighted) copies of the entries of the tilde'd matrix.

2.1. Vectorization of the cost function

We begin by vectorizing the cost function as in (2). Noting that $\|\mathtt{X}\|_F^2 = \operatorname{trace}(\mathtt{X}^\top\mathtt{X}) = \|\operatorname{vec}(\mathtt{X})\|_2^2$, minimizing (1) is equivalent to minimizing

$$\|\varepsilon(\mathtt{U},\mathtt{V})\|_2^2 := \left\| \begin{matrix} \varepsilon_1(\mathtt{U},\mathtt{V}) \\ \varepsilon_2(\mathtt{U}) \\ \varepsilon_3(\mathtt{V}) \end{matrix} \right\|_2^2 = \left\| \begin{matrix} \Pi \operatorname{vec}(\mathtt{W} \odot (\mathtt{U}\mathtt{V}^\top - \mathtt{M})) \\ \sqrt{\mu} \operatorname{vec}(\mathtt{U}) \\ \sqrt{\mu} \operatorname{vec}(\mathtt{V}^\top) \end{matrix} \right\|_2^2$$

where Π is a fixed $p \times mn$ projector matrix, where p is the number of visible elements, which eliminates known-zero entries from ε_1. Using some linear algebra basics from [18] and [14], we define

$$\varepsilon_1(\mathtt{U},\mathtt{V}) = \Pi \operatorname{diag}(\operatorname{vec} \mathtt{W}) \operatorname{vec}(\mathtt{U}\mathtt{V}^\top - \mathtt{M}) \tag{8}$$
$$:= \tilde{\mathtt{W}} \operatorname{vec}(\mathtt{U}\mathtt{V}^\top) - \tilde{\mathtt{W}}\mathbf{m} \tag{9}$$

where $\tilde{\mathtt{W}}$ is $\Pi\,\mathrm{diag}(\mathrm{vec}(\mathtt{W}))$ and $\mathbf{m}$ is $\mathrm{vec}(\mathtt{M})$. Defining $\mathbf{u} := \mathrm{vec}(\mathtt{U})$, $\mathbf{v} := \mathrm{vec}(\mathtt{V}^\top)$ and $\tilde{\mathbf{m}} := \tilde{\mathtt{W}}\mathbf{m}$ yields

$$\begin{aligned}\varepsilon_1(\mathtt{U},\mathtt{V}) &= \tilde{\mathtt{W}}(\mathtt{I}_n \otimes \mathtt{U})\,\mathrm{vec}(\mathtt{V}^\top) - \tilde{\mathbf{m}} && (10)\\ &= \tilde{\mathtt{U}}\mathbf{v} - \tilde{\mathbf{m}}. && /\!/\ \tilde{\mathtt{U}} := \tilde{\mathtt{W}}(\mathtt{I}\otimes\mathtt{U}) \quad (11)\\ (&= \tilde{\mathtt{V}}\mathbf{u} - \tilde{\mathbf{m}}.) && /\!/\ \tilde{\mathtt{V}} := \tilde{\mathtt{W}}(\mathtt{V}\otimes\mathtt{I}) \quad (12)\end{aligned}$$

Again, recall that $\tilde{\mathtt{U}}$ is a rearrangement of the entries of $\mathtt{U}$, and hence of $\mathbf{u}$. The resulting vectorized cost function is

$$\left\|\begin{matrix}\varepsilon_1(\mathbf{u},\mathbf{v})\\ \varepsilon_2(\mathbf{u})\\ \varepsilon_3(\mathbf{v})\end{matrix}\right\|_2^2 = \left\|\begin{matrix}\tilde{\mathtt{U}}\mathbf{v}-\tilde{\mathbf{m}}\\ \sqrt{\mu}\mathbf{u}\\ \sqrt{\mu}\mathbf{v}\end{matrix}\right\|_2^2 = \left\|\begin{matrix}\tilde{\mathtt{V}}\mathbf{u}-\tilde{\mathbf{m}}\\ \sqrt{\mu}\mathbf{u}\\ \sqrt{\mu}\mathbf{v}\end{matrix}\right\|_2^2. \quad (13)$$

2.2. Block coordinate descent

We first review coordinate descent (a.k.a. alternation), partly in support of the VarPro derivation in §2.5. Since the cost function is bilinear, we can eliminate either $\mathbf{u}$ or $\mathbf{v}$, by taking the partial derivative with respect to $\mathtt{U}$ and $\mathtt{V}$ and setting them to 0. i.e.

$$\begin{aligned}\mathbf{v}^*(\mathbf{u}) &= \arg\min_{\mathbf{v}} \left\|\begin{matrix}\tilde{\mathtt{U}}\mathbf{v}-\tilde{\mathbf{m}}\\ \sqrt{\mu}\mathbf{v}\end{matrix}\right\|_2^2 && (14)\\ &= (\tilde{\mathtt{U}}^\top\tilde{\mathtt{U}} + \mu\mathtt{I})^{-1}\tilde{\mathtt{U}}^\top\tilde{\mathbf{m}} && (15)\\ &= \tilde{\mathtt{U}}^{-\mu}\tilde{\mathbf{m}}, && (16)\end{aligned}$$

where $\mathtt{X}^{-\mu} := (\mathtt{X}^\top\mathtt{X} + \mu\mathtt{I})^{-1}\mathtt{X}^\top$. A similar calculation produces $\mathbf{u}^*(\mathbf{v}) = (\tilde{\mathtt{V}}^\top\tilde{\mathtt{V}}+\mu\mathtt{I})^{-1}\tilde{\mathtt{V}}^\top\tilde{\mathbf{m}}$. Alternation updates one matrix at a time, so iteration k is:

$$\begin{aligned}\mathbf{v}_{k+1} &= \tilde{\mathtt{U}}_k^{-\mu}\tilde{\mathbf{m}} && (17)\\ \mathbf{u}_{k+1} &= \tilde{\mathtt{V}}_{k+1}^{-\mu}\tilde{\mathbf{m}}. && (18)\end{aligned}$$

As previously reported in the literature [5], this approach is known to be vulnerable to flatlining.

2.3. Joint optimization

Joint optimization updates all $(m+n)r$ parameters simultaneously by stacking into the vector $\mathbf{x} = [\mathbf{u};\mathbf{v}]$ and using second-order optimization. The iteration k update is

$$\mathbf{x}_{k+1} = \mathbf{x}_k - (\mathtt{H}_k)^{-1}\mathbf{g} \quad (19)$$

where the matrix $\mathtt{H} \in \mathbb{S}^N$ is some function of or approximation to the Hessian $\nabla\nabla^\top f$ at $\mathbf{x}_k$, and $\mathbf{g}$ is the gradient $\nabla f(\mathbf{x}_k)$. Different choices lead to different, but closely related, algorithms.

2.4. "Plug and play" notation

We present a "plug and play" summary of these options in Table 2, where text colouring and bracketing is used to indicate which components are active in each of several variants. For example, an unregularized Gauss-Newton algorithm is given by including only the terms in black. Levenberg-Marquardt (LM) is obtained by adding black and ⟨angle-bracketed pink⟩ terms. Damped Newton [5] combines black with $[\text{bracketed orange}]_{FN}$ and ⟨angle-bracketed pink⟩. While this may appear unnecessarily garish, it allows us quickly to observe that LM is not just the same as adjusting the regularizer μ, because the latter has additional terms in the gradient. It also shows clearly the differences between Gauss-Newton and full Newton.

2.5. Variable projection (VarPro), unregularized

The key derivation in applying VarPro is to compute the derivative of $\mathbf{v}^*(\mathbf{u})$, from (16). From [13], this is

$$\frac{d\mathbf{v}^*}{d\mathbf{u}} = -(\tilde{\mathtt{U}}^\top\tilde{\mathtt{U}} + \mu\mathtt{I}_{nr})^{-1}(\tilde{\mathtt{U}}^\top\tilde{\mathtt{V}}^* + {\mathtt{Z}^*}^\top\mathtt{K}_{mr}) \quad (20)$$

where $\mathtt{Z}^*$ is from (7). Henceforth, to simplify the analysis we will treat only the unregularized ($\mu = 0$) case. Excepting the RTRMC algorithm of Boumal and Absil [3], which applies regularized VarPro using the exact Hessian matrix, the rest of the considered algorithms use no regularization and still obtain good optima with sound extrapolation. Then expanding (20), and colorizing yields

$$\frac{d\mathbf{v}^*}{d\mathbf{u}} = -[\tilde{\mathtt{U}}^\dagger\tilde{\mathtt{V}}^*]_{RW2} - [(\tilde{\mathtt{U}}^\top\tilde{\mathtt{U}})^{-1}{\mathtt{Z}^*}^\top\mathtt{K}_{mr}]_{RW1} \quad (21)$$

where the $[\text{green}]_{RW2}$ term is the approximation used in RW2 and Damped Wiberg, while $[\text{blue}]_{RW1}$ is added for RW1 or Chen's LM-S$_{GN}$. This adds corresponding switches to the Hessian approximation as shown in Table 1. Note that for our modification of Chen's algorithms we have replaced `orth` with retraction using the q-factor (i.e. $\mathtt{U} = \mathtt{qorth}(\mathtt{U})$ is $[\mathtt{U},\sim] = \mathtt{qr}(\mathtt{U},0)$ in MATLAB).

2.6. Manifold optimization

It is clear that the objective function for variable projection has gauge freedom [3, 7] which means that $f^*(\mathtt{U}) = f^*(\mathtt{UA})$ for any orthonormal $r\times r$ matrix $\mathtt{A}$. If $\mu = 0$ then this is true for any invertible $\mathtt{A}$. This is equivalent to solutions lying on the Grassmann manifold [3, 7]. It is natural to ask if poor convergence on the matrix factorization problem could be caused by this gauge freedom, so methods to address it have been proposed. The book on Riemannian manifold optimization by Absil *et al.* [1] suggests that instead of a standard second-order VarPro update

$$\Delta\mathbf{u} = -{\mathtt{H}^*}^{-1}\mathbf{g}^* = \arg\min_{\boldsymbol{\delta}} {\mathbf{g}^*}^\top\boldsymbol{\delta} + \frac{1}{2}\boldsymbol{\delta}^\top\mathtt{H}^*\boldsymbol{\delta}, \quad (22)$$

the update should be the solution to a projected subproblem with projected gradient $\mathbf{g}_p^*$ and Hessian $\mathtt{H}_p^*$

$$\Delta\mathbf{u} = \arg\min_{\boldsymbol{\delta}\perp\mathbf{u}} {\mathbf{g}_p^*}^\top\boldsymbol{\delta} + \frac{1}{2}\boldsymbol{\delta}^\top\mathtt{H}_p^*\boldsymbol{\delta} \quad (23)$$

Algorithm	Framework	Manifold retraction
ALS [4]	RW3 (ALS)	None
PowerFactorization [4, 29]	RW3 (ALS)	q-factor ($\mathtt{U} = \mathtt{qorth}(\mathtt{U})$)
LM-S [7]	Newton + ⟨Damping⟩	$\mathtt{orth}$ (replaced by q-factor)
LM-S$_{GN}$ [8, 12]	RW1 (GN) + ⟨Damping⟩ (DRW1 equiv.)	$\mathtt{orth}$ (replaced by q-factor)
LM-M [7]	Reduced$_r$ Newton + ⟨Damping⟩	$\mathtt{orth}$ (replaced by q-factor)
LM-M$_{GN}$ [7]	Reduced$_r$ RW1 (GN) + ⟨Damping⟩	$\mathtt{orth}$ (replaced by q-factor)
Wiberg [19]	RW2 (Approx. GN)	None
Damped Wiberg [20]	RW2 (Approx. GN) + ⟨Projection const.⟩$_P$ + ⟨Damping⟩	None
CSF [12]	RW2 (Approx. GN) + ⟨Damping⟩ (DRW2 equiv.)	q-factor ($\mathtt{U} = \mathtt{qorth}(\mathtt{U})$)
RTRMC [3]	Projected$_p$ Newton + {Regularization} + ⟨Trust Region⟩	q-factor ($\mathtt{U} = \mathtt{qorth}(\mathtt{U})$)
LM-S$_{RW2}$	RW2 (Approx. GN) + ⟨Damping⟩ (DRW2 equiv.)	q-factor ($\mathtt{U} = \mathtt{qorth}(\mathtt{U})$)
LM-M$_{RW2}$	Reduced$_r$ RW2 (Approx. GN) + ⟨Damping⟩	q-factor ($\mathtt{U} = \mathtt{qorth}(\mathtt{U})$)
DRW1	RW1 (GN) + ⟨Damping⟩	q-factor ($\mathtt{U} = \mathtt{qorth}(\mathtt{U})$)
DRW1P	RW1 (GN) + ⟨Projection const.⟩$_P$ + ⟨Damping⟩	q-factor ($\mathtt{U} = \mathtt{qorth}(\mathtt{U})$)
DRW2	RW2 (Approx. GN) + ⟨Damping⟩	q-factor ($\mathtt{U} = \mathtt{qorth}(\mathtt{U})$)
DRW2P	RW2 (Approx. GN) + ⟨Projection const.⟩$_P$ + ⟨Damping⟩	q-factor ($\mathtt{U} = \mathtt{qorth}(\mathtt{U})$)
$\frac{1}{2}\mathtt{H}^* = \mathtt{P}_r{}^\top \big(\tilde{\mathtt{V}}^{*\top}(\mathtt{I}_p - [\tilde{\mathtt{U}}\tilde{\mathtt{U}}^\dagger]_{RW2})\tilde{\mathtt{V}}^* + [\mathtt{K}_{mr}^\top \mathtt{Z}^*(\tilde{\mathtt{U}}^\top\tilde{\mathtt{U}})^{-1}\mathtt{Z}^{*\top}\mathtt{K}_{mr}]_{RW1} \times [-1]_{FN} + [\mathtt{K}_{mr}^\top \mathtt{Z}^*\tilde{\mathtt{U}}^\dagger\tilde{\mathtt{V}}^*\mathtt{P}_p + \mathtt{P}_p\tilde{\mathtt{V}}^{*\top}\tilde{\mathtt{U}}^{\dagger\top}\mathtt{Z}^{*\top}\mathtt{K}_{mr}]_{FN} + \langle \alpha \mathtt{I}_r \otimes \mathtt{U}\mathtt{U}^\top\rangle_P + \langle\lambda\mathtt{I}_{mr}\rangle\big)\mathtt{P}_r$		

Table 1: Unified analysis of algorithms in the literature (with citations) and our proposals. The bottom row is the Hessian approximation whose terms are switched by the choice of algorithm. More details can be found in the supplementary material.

Gauss-Newton + [Full Newton]$_{FN}$ w/o {Regularization} w/o ⟨Damping⟩
$\mathtt{J} = \begin{bmatrix} \tilde{\mathtt{V}} & \tilde{\mathtt{U}} \\ \{\sqrt{\mu}\mathtt{I}_{mr}\} & \\ & \{\sqrt{\mu}\mathtt{I}_{nr}\} \end{bmatrix}$
$\frac{1}{2}\mathbf{g} = \begin{bmatrix} \tilde{\mathtt{V}}^\top \boldsymbol{\varepsilon}_1 + \{\mu\mathbf{u}\} \\ \tilde{\mathtt{U}}^\top \boldsymbol{\varepsilon}_1 + \{\mu\mathbf{v}\} \end{bmatrix}$
$\frac{1}{2}\mathtt{H} = \begin{bmatrix} \tilde{\mathtt{V}}^\top\tilde{\mathtt{V}} + \{\mu\mathtt{I}_{mr}\} + \langle\lambda\mathtt{I}_{mr}\rangle & \tilde{\mathtt{V}}^\top\tilde{\mathtt{U}} + [\mathtt{K}_{mr}^\top\mathtt{Z}]_{FN} \\ \tilde{\mathtt{U}}^\top\tilde{\mathtt{V}} + [\mathtt{Z}^\top\mathtt{K}_{mr}]_{FN} & \tilde{\mathtt{U}}^\top\tilde{\mathtt{U}} + \{\mu\mathtt{I}_{nr}\} + \langle\lambda\mathtt{I}_{nr}\rangle \end{bmatrix}$

Table 2: Computations for joint optimization. Best viewed in colour, but bracketing is equivalent.

Algorithm	Framework
CE_LM	GN + ⟨Damping⟩
CE_LMI	GN + ⟨Damping⟩ with inner iterations
CE_ALM	10 ALS → CE_LM
CE_ALMI	10 ALS → CE_LMI
CE_ARULM	10 Reg. ALS → Reg. CE_LM → CE_LM
CE_ARULMI	10 Reg. ALS → Reg. CE_LMI → CE_LMI

Table 3: Our proposals of joint optimization algorithms based on the unified analysis. Above algorithms are all implemented using Ceres-solver [2]. The Ceres documentation describes inner iterations as the non-linear generalization of Ruhe & Wedins Algorithm II (RW2).

where $\boldsymbol{\delta} \perp \mathbf{u}$ is the linear constraint $\mathtt{U}^\top \operatorname{unvec}(\boldsymbol{\delta}) = 0$. This constrains the update to be made on the subspace tangential to the current $\mathtt{U}$. It turns out [13] that the projected gradient and the Gauss-Newton matrix are the same as the originals. When $\Delta\mathbf{u}$ is applied, there is also retraction onto the manifold, so $\mathtt{U} = \mathtt{qorth}(\operatorname{unvec}(\mathbf{u} + \Delta\mathbf{u}))$.

Chen [7] introduced the algorithm LM_M along with LM_S, which introduces Grassmann manifold projection using the approach of Manton et al. [15]. This involves solving a reduced subproblem, in which the dimension of the update is reduced from $\mathbb{R}^{mr}$ to $\mathbb{R}^{(m-r)r}$ which is orthogonal to current $\mathtt{U}$, minimizing

$$\Delta\mathbf{u} = \mathtt{P}_r{}^\top \left(\arg\min_{\boldsymbol{\delta}} \mathbf{g}_r^{*\top}\boldsymbol{\delta} + \boldsymbol{\delta}^\top \mathtt{H}_r^* \boldsymbol{\delta} \right) \tag{24}$$

where $\mathtt{P}_r \in \mathbb{R}^{(m-r)r \times mr}$ and $\mathtt{H}_r^* \in \mathbb{S}^{(m-r)r}$ are defined in the supplementary material [13]. A similar connection was also recently made in the field of control theory [28].

The hard constraint $\mathtt{U}^\top \operatorname{unvec}(\boldsymbol{\delta}) = 0$ may also be relaxed by adding a penalty term as follows:

$$\mathbf{g}_p^{*\top}\boldsymbol{\delta} + \boldsymbol{\delta}^\top \mathtt{H}_p^* \boldsymbol{\delta} + \langle \alpha \|\mathtt{U}^\top \operatorname{unvec}(\boldsymbol{\delta})\|_2^2 \rangle_P. \tag{25}$$

This in fact introduces the same term as the one introduced by Okatani et al. [20] when α is 1.

2.7. Summary

Table 1 summarizes several existing algorithms in terms of the above components, showing how each comprises

ID	Dataset	Dimension	r	Filled	(%)	Unique W columns (%)	Best known optimum
DIN	Dinosaur [12]	$72 \times 4,983$	4	32,684	9.2	275 / 4,983 (5.5 %)	1.134558
Din	Dinosaur trimmed [5]	72×319	4	5,302	23.1	106 / 319 (33.2 %)	1.084673
GIR	Giraffe [5]	166×240	6	27,794	69.8	95 / 240 (39.6 %)	0.322795
FAC	Face [5]	$20 \times 2,944$	4	34,318	58.3	679 / 2,944 (23.1 %)	0.022259
Fac	Face trimmed [20]	$20 \times 2,596$	4	33,702	64.9	627 / 2,596 (24.2 %)	0.022461
Scu	Sculpture trimmed [6]	$46 \times 16,301$	3	498,422	66.5	5,395 / 16,301 (33.1 %)	0.089680
UB4	UM boy [22]	$110 \times 1,760$	4	27,902	14.4	418 / 1,760 (23.8 %)	1.266484
UB5	UM boy [22]	$110 \times 1,760$	5	27,902	14.4	418 / 1,760 (23.8 %)	0.795494
UGf	UM gir. fg. [22]	$380 \times 4,885$	6	168,286	9.1	2,381 / 4,885 (48.7 %)	0.774258
UGb	UM gir. bg. [22]	$380 \times 6,310$	4	164,650	6.9	380 / 6,310 (38.0 %)	0.603904*
JE1	Jester 1 [10]	$100 \times 24,983$	7	1,810,455	72.5	13,718 / 24,983 (54.9 %)	3.678013
JE2	Jester 2 [10]	$100 \times 23,500$	7	1,708,993	72.7	12,788 / 23,500 (54.4 %)	3.703549
NET	Netflix 2k	$2,000 \times 50,000$	4	2,606,298	2.7	N/A	0.806635

Table 4: Datasets used for the experiments. *For UGb, this minimum has not been found by any other run.

Algorithm	Successes / 20		Time (ms / iter)	
	Orig.	Mod.	Orig.	Mod.
LM-S	4	4	380	140
LM-M	1	6	369	143
LM-M$_{GN}$	15	19	205	109

Table 5: Comparison on one available codebase [7] before and after profile-guided optimization on the trimmed dinosaur (Din) dataset.

Extension	Algorithm	Code
DW	Damped Wiberg [20]	New
PG_CSF	CSF [12]	Original
TO_DW	Damped Wiberg [20]	Original
NB_RTRMC	RTRMC [3]	Original
CH_LM_S	LM-S [7]	Modified
CH_LM_S_GN	LM-S$_{GN}$ [8, 12]	Modified
CH_LM_S_RW2	LM-S$_{RW2}$	Modified
CH_LM_M	LM-M [7]	Modified
CH_LM_M_GN	LM-M$_{GN}$	Modified
CH_LM_M_RW2	LM-M$_{RW2}$	Modified
DB_BALM	BALM [9]	New
RC_ALM	ALM [6]	New
RC_RCALM	Rank-continuation [6]	New

Table 6: Algorithm tags referred to the original authors' naming. "Code" is "New" for our re-implementations of existing algorithms. DW uses new implementation of VarPro.

some subset. It also includes some "new" algorithms such as DRW2, which is just Ruhe and Wedin's original Algorithm II, but with damping, which has not previously been proposed for matrix factorization.

3. Implementation issues

Having isolated key components of the algorithm, it is now straightforward to re-implement existing methods, and to explore issues of numerical stability and speed. In doing so, we have uncovered some "secrets": three sources of speed improvement, and one important numerical stability enhancement. Of course we don't mean these have been kept deliberately secret, but that they are apparently small elements which can have a large impact on performance.

Numerical issues emerge in the implementation of these algorithms. Foremost is in the inversion of the Hessian approximation, and in the projection onto the manifold. We have found that the use of QR factorization as proposed by Okatani et al. [20] in both of these (rather than Matlab's `chol` or `orth` respectively) significantly improves accuracy and performance. Note that speed improvements might also adversely affect numerical stability, but in fact in all cases we have investigated, it *improves* it.

Profile-guided optimization amounts to exploiting standard Matlab tricks for code speedup, including MEX file implementations of some Kronecker products. This offered a factor of 2-3 improvement on the set of Chen's algorithms as shown in Table 5.

Removal of redundant computations Not all existing implementations took advantage of the symmetry in the Hessian, and in fact there is another internal symmetry [13] which yields a 4-fold speedup.

UW-block coalescing A more subtle speedup is obtained in the QR decomposition of $\tilde{\mathtt{U}}$. For the un-regularized case, $\mathbf{v}^* = \tilde{\mathtt{U}}^\dagger \tilde{\mathbf{m}}$. We note that the use of QR decomposition by Okatani et al. [20] is very useful since the decomposed outputs can be re-used for the computation of the Gauss-Newton matrix. Given that $\tilde{\mathtt{U}} = \tilde{\mathtt{U}}_Q \tilde{\mathtt{U}}_R$, the above equation

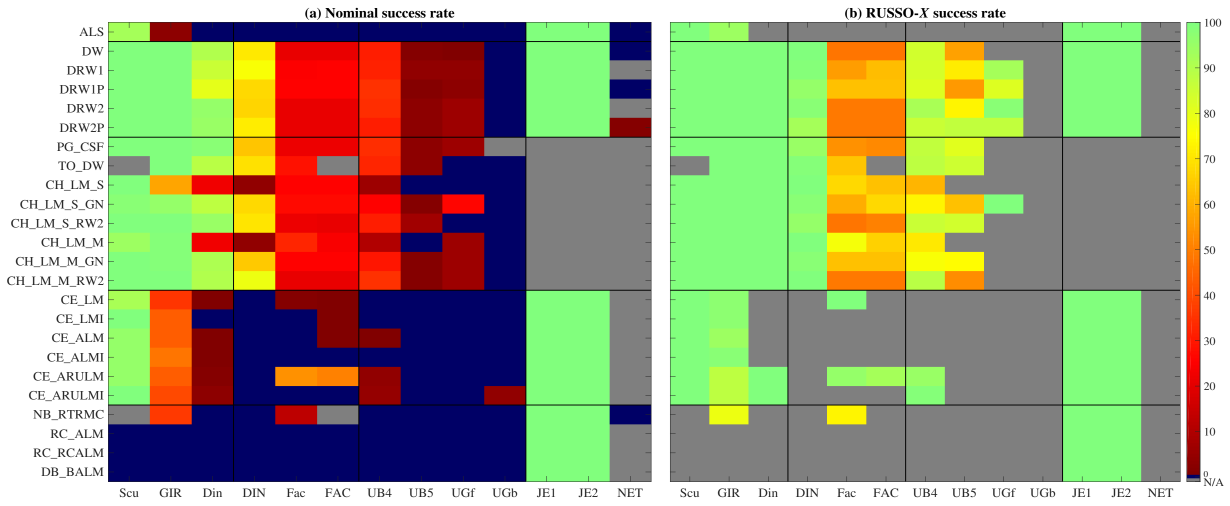


Figure 2: (a) Success rates in reaching the best known optimum. Grey cells crashed or timed out. (b) Success rates of the RUSSO versions. Gray cells now also include timeouts where the optimum was not seen twice (e.g. a success rate of 1% would require thousands of runs, so timeout is more likely). The "Face" and "Unwrap BG" datasets have large secondary minima so 30-50% of runs terminate at those. The RUSSO-DRW family succeed on the largest number of benchmarks.

becomes

$$\mathbf{v}^* = \tilde{\mathtt{U}}_R^{-1}\tilde{\mathtt{U}}_Q^\top\tilde{\mathbf{m}}. \tag{26}$$

Since $\tilde{\mathtt{U}} = \tilde{\mathtt{W}}(\mathtt{I}_n \otimes \mathtt{U})$ where $\tilde{\mathtt{W}}$ is the truncated version of $\operatorname{diag}\operatorname{vec}(\mathtt{W})$, we can observe that $\tilde{\mathtt{U}}$ is block-diagonal with the i-th block being $\tilde{\mathtt{W}}_i\mathtt{U}$ where $\tilde{\mathtt{W}}_i$ is the truncated version of $\operatorname{diag}\operatorname{vec}(\mathtt{W}_i)$ and $\mathtt{W}_i$ is the i-th column of the weight matrix $\mathtt{W}$. Noting that $\tilde{\mathtt{U}}_Q^\top\tilde{\mathtt{U}}_Q = \mathtt{I}$ by definition, $\tilde{\mathtt{U}}_Q$ is also going to have the same block-diagonal structure with the i-th block being the q-factor of the i-th block of $\tilde{\mathtt{U}}$. In other words, if there exists a set of same columns in the weight matrix, we only have to compute the q-factor of the corresponding blocks once. Since all the real datasets we have consist of indicator-type weight matrix which has values either 0 or 1, such repetition is more likely to occur. The speedups obtained from this approach were a factor of 2.3 on average, ranging from 1 (no speedup, on random-like data) to 5 (full dino sequence).

4. Experimental results

All experiments were carried out on a Macbook Pro (Late 2013) with 2.3 GHz Intel Core i7 Haswell processor and 16 GB 1600 MHz DDR3 memory. We used Ceres [2] for joint optimization. All other algorithms used MATLAB R2014b. All experiments were run in single-threaded mode to compare the speed of the algorithms. All algorithms are essentially equally parallelizable.

4.1. Datasets

For the experiments, we have used all the datasets listed in Table 4. The problem classes are: rigid SfM (dinosaur [5]), non-rigid SfM (giraffe [5], UM boy and UM giraffe foreground and background [22]), photometric stereo (face [5] and sculpture [6]) and recommender systems (Jester and Netflix). In previous evaluations, some datasets were trimmed, so we also include the original sets in the evaluation, indicated by the use of all caps for the originals, and mixed case for the trimmed sets.

4.2. Experimental conditions

On each dataset, we ran each algorithm multiple times, usually between 20 and 100, from random starting points. This was done by drawing initial entries of $\mathtt{U}$ from multivariate Normal distribution $\mathcal{N}(\mathbf{0}, \mathtt{I})$. i.e. $\mathtt{U} = \mathtt{randn}(\mathtt{m}, \mathtt{r})$.

In order to set up a fair competing environment on each dataset, all algorithms at the same run number were initialized with the same starting points. For those algorithms requiring initial $\mathtt{V}$ as well (e.g. CE_LM and DB_BALM) were given $\mathtt{V}^*(\mathtt{U})$ computed from (16) so that they would start at the same random points as other VarPro-based algorithms, which require only initial $\mathtt{U}$.

Each run was continued until either when the maximum number of iterations (set to be 300) was reached or the cost function improvement dropped below pre-defined tolerance 10^{-10}. Runtime and success rates were measured as described in the supplementary material [13].

5. Discussions and conclusions

In this paper, we have addressed the important problem of matrix factorization with missing data. We have argued that evaluation of competing algorithms must be based on real-world runtime. In order to be able to discuss runtime,

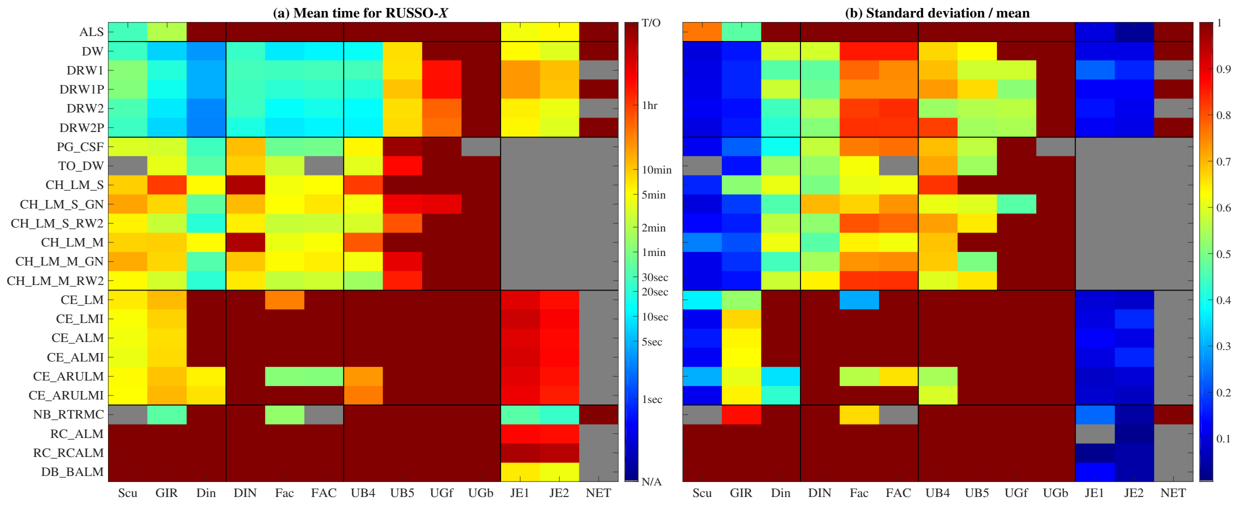


Figure 3: Runtime on all algorithms, all datasets. (a) Mean runtime of RUSSO-X. (b) Standard deviation divided by mean. Where $\sigma = T/O$, too few runs converged to estimate RUSSO's performance. On the UGb dataset, no algorithm converged to the same solution twice.

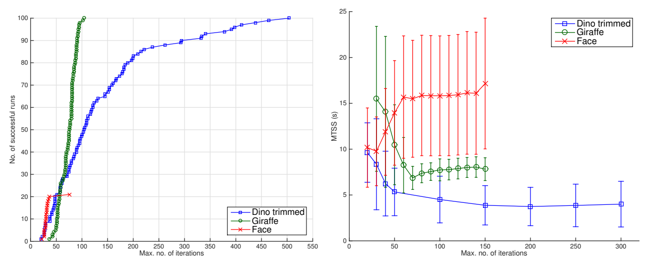
we developed a generalized theoretical framework for several known approaches. We have also shown how a meta-algorithm, RUSSO-X greatly increases performance, when used with standard VarPro.

By discussing runtime, other algorithm parameters may be more sensibly set. For example, the maximum number of iterations taken is an algorithm parameter which affects runtime and accuracy. Figure 4 shows how, while accuracy always increases with MaxIter, the safety net of RUSSO allows smaller values to be chosen, improving overall performance.

By re-implementing standard variable projection algorithms (our "DRW" set), we have increased performance on the small and medium-scale problems over the state of the art by an order of magnitude (factors range from 5 to 15). We have introduced new datasets which seem more tricky to optimise than those previously in use, and open-source implementations of all code, data and evaluations are available [13]. However, we do not propose that these datasets should become a new benchmark for this problem—as individual practitioners find problems where existing algorithms fail, we want these to be incorporated into this framework.

It remains the case that alternation wrapped in RUSSO (RUSSO-ALS) is still competitive for some specialized datasets, for example the "easy" sets with high fill rates (Scu, Gir), but also the large-scale datasets (JE1, JE2). However, even then, it is beaten by RUSSO-DRW2P. However, out-of-core implementation of the latter does not have an easy off-the-shelf implementation to date.

In order to quantify the speedup, we also hand-optimized several existing publicly available codes, and showed that the speedup obtained by optimization, while potentially significant, is not the main reason for our new performance. Conversely, our re-implementation of Damped Wiberg algorithm [20] is comparable to the state of the art, but the new contribution is to cast it as a variable projection algorithm without manifold projection.



(a) Success rate. (b) Mean time to 2nd success.

Figure 4: Performance as a function of MaxIter (algorithm DRW2P). While the success rate metric increases monotonically with MaxIter, it is better in practice to fail early and try another starting point. Runtime to second success allows selection of a value that improves real-world performance for a given class of problem.

A tantalizing note for the future is recent work in robust estimation where VarPro performs significantly worse than joint optimization [31].

Acknowledgements

The work was supported by Microsoft and Toshiba Research Europe. We thank Roberto Cipolla, Christopher Zach, Bamdev Mishra and the anonymous reviewers for their invaluable comments.

References

[1] P.-A. Absil, R. Mahony, and R. Sepulchre. *Optimization Algorithms on Matrix Manifolds*. Princeton University Press, 2008. 4

[2] S. Agarwal, K. Mierle, and Others. Ceres solver. `http://ceres-solver.org`, 2014. 5, 7

[3] N. Boumal and P.-A. Absil. RTRMC: A Riemannian trust-region method for low-rank matrix completion. In *Advances in Neural Information Processing Systems 24 (NIPS 2011)*, pages 406–414. 2011. 4, 5, 6

[4] A. M. Buchanan. Investigation into matrix factorization when elements are unknown. Technical report, University of Oxford, 2004. 5

[5] A. M. Buchanan and A. W. Fitzgibbon. Damped Newton algorithms for matrix factorization with missing data. In *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2, pages 316–322, 2005. 3, 4, 6, 7

[6] R. Cabral, F. De la Torre, J. P. Costeira, and A. Bernardino. Unifying nuclear norm and bilinear factorization approaches for low-rank matrix decomposition. In *Proceedings of the 2013 IEEE Internatonal Conference on Computer Vision (ICCV)*, pages 2488–2495, 2013. 1, 2, 6, 7

[7] P. Chen. Optimization algorithms on subspaces: Revisiting missing data problem in low-rank matrix. *International Journal of Computer Vision (IJCV)*, 80(1):125–142, 2008. 3, 4, 5, 6

[8] P. Chen. Hessian matrix vs. Gauss-Newton Hessian matrix. *SIAM Journal on Numerical Analysis (SINUM)*, 49(4):1417–1435, 2011. 5, 6

[9] A. Del Bue, J. Xavier, L. Agapito, and M. Paladini. Bilinear modeling via augmented Lagrange multipliers (BALM). *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 34(8):1496–1508, Aug 2012. 2, 6

[10] K. Goldberg, T. Roeder, D. Gupta, and C. Perkins. Eigentaste: A constant time collaborative filtering algorithm. *Information Retrieval*, 4(2):133–151, Jul 2001. 6

[11] G. H. Golub and V. Pereyra. The differentiation of pseudo-inverses and nonlinear least squares problems whose variables separate. *SIAM Journal on Numerical Analysis (SINUM)*, 10(2):413–432, 1973. 3

[12] P. F. Gotardo and A. M. Martinez. Computing smooth time trajectories for camera and deformable shape in structure from motion with occlusion. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 33(10):2051–2065, Oct 2011. 3, 5, 6

[13] J. H. Hong and A. Fitzgibbon. Secrets of matrix factorization: Supplementary material to ICCV 2015 submission. Technical report, at `https://github.com/jhh37/matrix-factorization`, 2015. 3, 4, 5, 6, 7, 8

[14] J. R. Magnus and H. Neudecker. *Matrix Differential Calculus with Applications in Statistics and Econometrics*. 3rd edition, 2007. 3

[15] J. H. Manton, R. Mahony, and Y. Hua. The geometry of weighted low-rank approximations. *IEEE Transactions on Signal Processing*, 51(2):500–514, Feb 2003. 5

[16] R. Martí. Multi-start methods. In F. Glover and G. A. Kochenberger, editors, *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research & Management Science*, pages 355–368. 2003. 2

[17] R. Mazumder, T. Hastie, and R. Tibshirani. Spectral regularization algorithms for learning large incomplete matrices. *Journal of Machine Learning Research (JMLR)*, 11(Aug):2287–2322, Aug 2010. 1

[18] T. P. Minka. Old and new matrix algebra useful for statistics. Technical report, Microsoft Research, 2000. 3

[19] T. Okatani and K. Deguchi. On the Wiberg algorithm for matrix factorization in the presence of missing components. *International Journal of Computer Vision (IJCV)*, 72(3):329–337, 2007. 3, 5

[20] T. Okatani, T. Yoshida, and K. Deguchi. Efficient algorithm for low-rank matrix factorization with missing components and performance comparison of latest algorithms. In *Proceedings of the 2011 IEEE International Conference on Computer Vision (ICCV)*, pages 842–849, 2011. 3, 5, 6, 8

[21] D. P. O'Leary and B. W. Rust. Variable projection for nonlinear least squares problems. *Computational Optimization and Applications*, 54(3):579–593, Apr 2013. 3

[22] A. Rav-Acha, P. Kohli, C. Rother, and A. W. Fitzgibbon. Unwrap mosaics: A new representation for video editing. *ACM Transactions on Graphics (TOG) - Proceedings of ACM SIGGRAPH 2008*, 27(3):17:1–17:11, Aug 2008. 6, 7

[23] A. Ruhe and P. Å. Wedin. Algorithms for separable nonlinear least squares problems. *SIAM Review (SIREV)*, 22(3):318–337, 1980. 3

[24] N. Srebro, J. Rennie, and T. S. Jaakkola. Maximum-margin matrix factorization. In *Advances in Neural Information Processing Systems 17 (NIPS 2004)*, pages 1329–1336. 2005. 1

[25] D. Strelow. General and nested Wiberg minimization. In *Proceedings of the 2012 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1584–1591, 2012. 2

[26] D. Strelow. General and nested Wiberg minimization: L2 and maximum likelihood. In *Proceedings of the 12th European Conference on Computer Vision (ECCV)*, pages 195–207. 2012. 2

[27] C. Tomasi and T. Kanade. Shape and motion from image streams under orthography: a factorization method. *International Journal of Computer Vision (IJCV)*, 9(2):137–154, 1992. 3

[28] K. Usevich and I. Markovsky. Optimization on a Grassmann manifold with application to system identification. *Automatica*, 50(6):1656–1662, Jun 2014. 5

[29] R. Vidal and R. Hartley. Motion segmentation with missing data using PowerFactorization and GPCA. In *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2, pages 310–316, Jun 2004. 5

[30] T. Wiberg. Computation of principal components when data are missing. In *Proceedings of the 2nd Symposium of Computational Statistics*, pages 229–326, 1976. 3

[31] C. Zach. Robust bundle adjustment revisited. In *In Proceedings of the 13th European Conference on Computer Vision (ECCV)*, pages 772–787, 2014. 8