ADAPTING REGULARIZED LOW-RANK MODELS FOR PARALLEL ARCHITECTURES*

DEREK DRIGGS†, STEPHEN BECKER‡, AND ALEKSANDR ARAVKIN§

Abstract. We introduce a reformulation of regularized low-rank recovery models to take advantage of GPU, multiple CPU, and hybridized architectures. Low-rank recovery often involves nuclear norm minimization through iterative thresholding of singular values. These models are slow to fit and difficult to parallelize because of their dependence on computing a singular value decomposition at each iteration. *Regularized* low-rank recovery models also incorporate nonsmooth terms to separate structured components (e.g., sparse outliers) from the low-rank component, making these problems more difficult. Using Burer–Monteiro splitting and marginalization, we develop a smooth, nonconvex formulation of regularized low-rank recovery models that can be fit with first-order solvers. Replacing convex problems with nonconvex programs that can be solved more quickly is a well-studied approach to low-rank recovery; here we show that handling the regularization terms carefully allows for greater speedup. Using robust principal component analysis (RPCA) as an example, we compare our approach to other convex, nonconvex, and communication-avoiding algorithms on the GPU, and show that ours is an order-of-magnitude faster. We also show that this acceleration allows for new applications of RPCA, including real-time background subtraction and magnetic resonance imaging analysis.

1. Introduction. Low-rank matrix decompositions are an effective tool for large-scale data analytics. Background subtraction, facial recognition, document indexing, and collaborative filtering all use low-rank matrix recovery algorithms [12]. However, a low-rank structure is only a part of the story—signals often exhibit additional structure. *Regularized* low-rank recovery models seek to decompose a matrix $X \in \mathbb{R}^{m \times n}$ into the sum of a low-rank component $L \in \mathbb{R}^{m \times n}$ and another structured matrix $S \in \mathbb{R}^{m \times n}$, where S could be sparse, clustered, nonnegative, or have some other useful attribute (see, e.g., [44]).

As a motivating example, consider finding an approximate decomposition of a data matrix X into a sum $X = L + S$, where S is sparse and L low rank, and we only partially observe X through linear measurements $b = \mathcal{A}(L) + S$. Searching for the sparsest or lowest-rank representation of this data is a combinatorial problem:

$$(1) \qquad \min_{L,S} \quad \text{rank}(L) + \lambda \cdot \text{card}(S) \quad \text{subject to} \quad \frac{1}{2}\|\mathcal{A}(L) + S - b\|_2^2 \leq \epsilon,$$

where the cardinality function $\text{card}(\cdot)$ counts the number of nonzero entries of S, λ is a tuning parameter, and the parameter ϵ accounts for noise or model inconsistencies. Problem (1) is NP-hard [34] and intractable for large problem sizes. However, both terms admit nonsmooth convex relaxations. $\text{card}(\cdot)$ can be replaced with the ℓ_1-norm,

---

†University of Cambridge, Cambridge CB3 0WA, UK (Derek.Driggs@colorado.edu).

‡University of Colorado, Boulder, CO 80309 (stephen.becker@colorado.edu).

§University of Washington, Seattle, WA 98195 (saravkin@uw.edu).

and the rank function can be replaced with the nuclear norm, $\|\cdot\|_*$, which is equal to the sum of the singular values of the argument:

$$(2) \qquad \min_{L,S} \quad \|L\|_* + \lambda_S \|S\|_1 \quad \text{subject to} \quad \frac{1}{2}\|\mathcal{A}(L) + S - b\|_2^2 \le \epsilon$$

When the parameter ϵ is set to zero, (2) is called *robust principal component analysis* (RPCA) and can recover the lowest-rank L and the sparsest S under mild conditions. Setting $\epsilon > 0$ in (2) is more suitable for most applications, since it allows for noise in the measurements; the problem is then called *stable principal component pursuit* (SPCP) [48, 52]. SPCP stably recovers L and S with error bounded by ϵ [52]. Both RPCA and SPCP are convex problems, much easier to solve than the original problem (1). While (2) is an important problem, it is just one example of a regularized low-rank model.

Problem class. We are interested in general regularized low-rank models:

$$(3) \qquad \min_{L,S} \quad \|L\|_* + \mathcal{L}(\mathcal{A}(L) + S - b) + r(S),$$

where $\mathcal{L}$ is a loss function and r is a convex regularizer. $\mathcal{L}$, which can be *infinite valued*, measures how well $\mathcal{A}(L) + S$ agrees with measured values of X. In RPCA, $\mathcal{L}(v) = \delta_b(v)$ is the indicator function ensuring we match observed data b, while in SPCP, $\mathcal{L}$ is the indicator of the closed Euclidean ball centered at b with radius ϵ, and ensures we are close to b. $\mathcal{L}$ can also be finite valued—for example, a modified SPCP is given by solving

$$(4) \qquad \min_{L,S} \quad \lambda_L \|L\|_* + \frac{1}{2}\|\mathcal{A}(L) + S - b\|_2^2 + \lambda_S \|S\|_1,$$

where λ_L, λ_S are tuning parameters, and $\mathcal{L}(\cdot) = \frac{1}{2}\|\cdot\|_2^2$.

The regularizer $r(S)$ is a convex penalty that promotes the desired structure of S. The ℓ_1 norm $r(S) = \|S\|_1$ promotes sparsity in S; other important examples include the ordered weighted ℓ_1 norm [9, 13, 50], and the elastic net [53], which both enforce sparsely correlated observations (clusters) in S. Other convex regularizers promote known group-sparse and hierarchical structures in S; see [5] for a survey.

The above formulations implicitly vectorize the matrix S into an $m \cdot n$-length vector. Following conventional notation, in the case of RPCA and SPCP when $\mathcal{A}$ is the identity operator, we write X instead of b, and let $\mathcal{L}$ be the squared *Frobenius*-norm loss.

Marginalization in S. In addition to assuming the problem structure (3), we also assume that $r(S)$ is simple enough that we can efficiently minimize (3) in S for a given L. In the motivating SPCP example (4), we have

$$\min_S \left\{\lambda_L \|L\|_* + \frac{1}{2}\|\mathcal{A}(L) + S - b\|_2^2 + \lambda_S \|S\|_1\right\} = \lambda_L \|L\|_* + \rho_{\lambda_S}(\mathcal{A}(L) - b),$$

where ρ_{λ_S} is the Huber function (for a detailed derivation, see, e.g., [3]). This motivates the definition

$$(5) \qquad \varphi : L \mapsto \min_S \left\{\mathcal{L}(\mathcal{A}(L) + S - b) + \lambda_S \cdot r(S)\right\},$$

where φ is smooth (see section 3.2) with value and derivative either explicitly available as in the case of SPCP, or efficiently computable. After marginalizing out S, (3)

becomes

$$(6) \qquad \min_{L} \ \lambda_L \|L\|_* + \varphi(L).$$

Nonconvex reformulation. Proximal gradient methods can then solve (6), but require computing the expensive and communication-heavy singular value decomposition (SVD) at each iteration. This problem has inspired the development of communication-avoiding algorithms for computing the SVD on parallel architectures, which we discuss further in section 6.

To avoid the issues of efficiently computing the SVD, we turn to a nonconvex, factorized representation of the nuclear norm that avoids the SVD entirely. This reformulated program offers more speedup through parallelization than communication-avoiding approaches, and the introduced nonconvexity can be easily controlled. The smooth structure of φ allows a computable *optimality certificate* for the nonconvex case, which is developed and explored in section 3.

To avoid the SVD entirely, we use the factorization $L = UV^T$, where $U \in \mathbb{R}^{m \times k}$, $V \in \mathbb{R}^{n \times k}$, and $k \ll m, n$. This factorization imposes the rank constraint $\operatorname{rank}(L) \leq k$. In addition, we can further penalize rank using the following representation of the nuclear norm (see, e.g., [6, 36, 42]):

$$(7) \qquad \|L\|_* \equiv \inf_{L=UV^T} \ \frac{1}{2}(\|U\|_F^2 + \|V\|_F^2).$$

This equivalence only holds if the ranks of U and V are large enough, and makes it possible to maintain a low-rank $L = UV^T$ without requiring SVDs. The nonconvex analogue to (6) is then given by

$$(8) \qquad \underset{U,V}{\arg\min} \ \frac{\lambda_L}{2}(\|U\|_F^2 + \|V\|_F^2) + \varphi(UV^T).$$

Our main contribution. The main contribution of this paper is showing that the transformation given by (8) provides a general framework for accelerating general low-rank recovery models. At a high level, our algorithm consists of (1) marginalizing to get (6) and then (2) using the change-of-variables $L = UV^T$, which turns the nonsmooth and possibly constrained convex problem (3) into a smooth unconstrained nonconvex problem. Once it is in this new form, the low-rank model can be fit using any first-order solver, which is much faster than fitting the original model using proximal methods, especially on the GPU, since calculating gradients (matrix multiplication) is cheap on the GPU, as opposed to communication-heavy algorithms like the SVD that are relatively slow on the GPU. Algorithm 1 provides an overview of the framework. Applying this framework to SPCP, we develop the program "Split-SPCP," which can be solved using any first-order algorithm. We have found using L-BFGS to fit Split-SPCP is particularly fast (see section 5 for more details). Split-SPCP's speed allows us to apply SPCP to new problems, including real-time background subtraction and magnetic resonance imaging (MRI) analysis.

Roadmap. We provide a survey of related work in section 2. We consider theoretical issues related to the smooth reformulation (8) in section 3. These include issues include the influence of nonoptimal stationary points, the implicit rank bound in the factorized problem, and the convexity of the marginalized function, φ. In section 4, we study heuristic initialization methods, which are important for nonconvex programs. We present detailed numerical studies, comparing our nonconvex

---

Algorithm 1: Split-SPCP meta-algorithm to solve (3).

Input : Problem data as in (3): functions $\mathcal{L}$ and r; data $b \in \mathbb{R}^q$; linear operator $\mathcal{A} : \mathbb{R}^{m \times n} \to \mathbb{R}^q$

1 Determine rank bound k, so $U \in \mathbb{R}^{m \times k}$, $V \in \mathbb{R}^{n \times k}$ `/* Choose` $r = \min(m, n)$ `to apply theory of section 3.3, or see section 4.1 */`

2 Define φ as in (5), and its (sub)gradient as in (10)

3 Define $F(U, V) = \frac{\lambda_L}{2}(\|U\|_F^2 + \|V\|_F^2) + \varphi(UV^T)$

4 Define $\nabla_U F(U, V) = \lambda_L U + \nabla \varphi(UV^T) V$

5 Define $\nabla_V F(U, V) = \lambda_L V + \nabla \varphi(UV^T)^T U$

6 Solve $\min_{U,V} F(U, V)$ (8) via an unconstrained optimization algorithm such as L-BFGS

---

method with other convex and nonconvex alternatives in section 5. In this section, we also demonstrate the effectiveness of our nonconvex formulation of low-rank recovery models for real-time video processing. In section 6, we compare our approach to communication-avoiding alternatives. Our numerical experiments illustrate that our nonconvex program is able to solve regularized low-rank recovery problems faster than existing solvers—in many cases, by an order of magnitude—without sacrificing accuracy in the solution. We end with final thoughts in section 7.

2. Prior work and contributions. Replacing convex problems with a faster-to-solve nonconvex problem is a well-studied technique in optimization, particularly in the study of semidefinite programs (SDPs). As nuclear norm minimization can be cast as an SDP, several authors have studied nonconvex, factorized matrix completion models, often solving them using alternating minimization [18, 21, 23, 24, 25, 26]. Matrix completion models are a special class of (8) when no regularizer is present (i.e., $r(S) \equiv 0$). This line of research is supported by the existence of conditions that guarantee fast convergence to a globally optimal solution [18, 21, 25, 26], sometimes at a linear rate [23, 44].

In [44], the authors review the use of alternating minimization to solve regularized principal components analysis (PCA) problems of the following form:

$$(9) \qquad \min_{U,V} \quad \mathcal{L}\left(\text{vec}(UV^T) - b\right) + r(U) + \tilde{r}(V).$$

Matrix completion models and RPCA can adopt the split-form in (9), but SPCP with a general sampling operator cannot. Our analysis applies to a more general class of models and offers an approach using first-order solvers rather than alternating minimization.

The authors of [40] develop a split RPCA program and solve it using alternating minimization as well. While their technique can be an order of magnitude faster than competitive convex solvers, its performance suffers when the magnitude of the sparse component is large compared to the magnitude of the low-rank component.

The algorithm developed in [49] is the approach most closely related to ours. In [49], the authors investigate a nonconvex formulation of RPCA that can be solved with gradient descent. When the rank of the low-rank component is small, their algorithm achieves provable convergence with a runtime that is nearly linear in the dimension of the problem. While this is the fastest known runtime for any RPCA algorithm, these

results are specific to RPCA, and cannot be easily extended to the general models we consider. Also, as we demonstrate in section 5, our approach can be significantly faster in practice. The paper [8] employs ideas similar to [49] to develop a first-order method for specific SDPs, but the problems we consider are still more general.

A different nonconvex approach to RPCA is considered in [35]. In contrast to the research discussed previously, their method does not use a low-rank factorization of L. Instead, these authors develop an algorithm of alternating projections, where each iteration sequentially projects L onto a set of low-rank matrices and thresholds the entries of S. Their technique also comes with recovery and performance guarantees.

Compared to existing work, our approach to accelerating low-rank recovery algorithms is more general and more effective. We develop a nonconvex formulation of regularized low-rank recovery models that can be solved using first-order optimization algorithms, and show that marginalizing the regularizers offers further performance guarantees. We address problems associated with local minima and spurious stationary points that accompany nonconvexity by showing that all local minima of our model are global minima provided the dimensions of U, V in (8) are large enough, and by providing a certificate to prove that a given point is a local minimum (and not a spurious stationary point). We also show that our method is particularly well suited for the GPU.

3. Theoretical considerations. We discuss three issues related to the approach (8). First, we study the convexity and differentiability of the marginal function (6) in sections 3.1 and 3.2. Next, moving from (6) to (8) by factoring L, we transform a convex problem into a nonconvex one. This creates two issues: an implicit rank constraint $\text{rank}(L) \leq k$ that is not present in (8), and the potential for local minima and spurious stationary points in the nonconvex variant that do not correspond to a solution of the convex problem. We address these issues in section 3.3.

Section 3.4 introduces a computable certificate that can be used to check whether the nonconvex model has converged to a global solution. Using the smoothness of φ, we derive a simple optimality certificate that can be tracked while optimizing (8) to identify either when the implicit rank of the factors is too small or when we have converged to a minimizer of (6).

3.1. Convexity of φ. The convexity of the marginal function (6) is a well-known result in convex analysis; see, e.g., [37, Prop. 2.22]. We include a short proof of this fact in Appendix B for completeness.

3.2. Smoothness of φ. In the special case where $\mathcal{L}$ is the least-squares error, Lipschitz differentiability of φ is immediate from the strong convexity of $\mathcal{L}$. Support for this claim is given, for example, in [7], where the relevant proposition is as follows.

PROPOSITION (see [7, Prop. 12.29]). *Let r be a proper lower semicontinuous convex function, and let $^{\mu}r(\mathcal{A}(L) + S - b) = \inf_S \ r(S) + \frac{1}{2\mu}\|\mathcal{A}(L) + S - b\|_2^2$ be the Moreau envelope of f with parameter μ. Then $^{\mu}r$ is differentiable and its gradient is μ^{-1}-Lipschitz continuous.*

In the general case, $\varphi(L)$ is differentiable as long as the objective in (4) has a unique minimizer (e.g., if it is strictly convex in S) and $\mathcal{L}$ is convex and smooth. By [37, Thm. 10.58], the subdifferential of φ is

(10)
$$\partial\varphi(L) = \text{conv}\left\{\mathcal{A}^*\nabla\mathcal{L}\left(\mathcal{A}(L) + S^\star - b\right) \mid S^\star \in \arg\min_S \frac{1}{2}\|\mathcal{A}(L) + S - b\|_2^2 + r(S)\right\}.$$

The subdifferential is a singleton if and only if the minimizing $S^\star$ is unique.

Using SPCP as an example ($r(S) = \lambda_S \|S\|_1$), we have

$$\varphi(L) = \min_S \left\{ \frac{1}{2} \|\mathcal{A}(L) + S - b\|_2^2 + \lambda_S \|S\|_1 \right\}.$$

The objective is strongly convex in S, and has the unique minimizer

$$S^\star = \text{prox}_{\lambda_S \|\cdot\|_1}(\mathcal{A}(L) - b).$$

Therefore, φ is differentiable and $\nabla \varphi$ is firmly nonexpansive, with

$$\nabla \varphi(L) = \mathcal{A}^* \left(\mathcal{A}(L) + S^\star - b \right).$$

We used (10) rather than the parametric form of $\varphi(L)$ to obtain its value and gradient. Using (7) to replace the nuclear norm and φ to eliminate the nonsmooth regularizer, we now have a smooth optimization problem amenable to first-order methods.

For minimizing smooth objectives, quasi-Newton methods are often faster than algorithms that require only first-order smoothness, such as gradient descent. The convergence rates of quasi-Newton methods depend on second-order smoothness, which does not hold in problems such as SPCP, since, e.g., the Huber function is only $\mathcal{C}^1$. Although convergence of this quasi-Newton method cannot be expected in general, there are special cases for which convergence results could be established [28, 29], and more recently [19], and empirically justified good numerical performance has been observed for decades [27].

3.3. Rank and local minima. The factorized problem (8) is not equivalent to (6) because of the implicit rank constraint. However, we can show that (8) is equivalent to the rank-constrained problem

$$\min_L \quad \lambda_L \|L\|_* + \varphi(L) \quad \text{subject to} \quad \text{rank}(L) \le k.$$

This follows immediately from (7). In order for the split problem to recover the solution to (1), it must be initialized with a k larger than the rank of the minimizing L. However, a larger k slows computation, so it must be chosen with care. This issue is considered in more depth in section 4.

We want to be sure that any local minimum of (8) corresponds to a (global) minimum of (6). The following theorem combines ideas from [11] and [4] to show this holds provided that k is larger than the rank of the minimizer for (6).

THEOREM 1. *Consider an optimization problem of the following form:*

$$(11) \qquad \min_{X \succeq 0} \quad f(X) \quad \text{such that} \quad \text{rank}(X) \le k,$$

where $X \in \mathbb{R}^{n \times n}$ is a positive semidefinite real matrix, and f is a lower semicontinuous function mapping to $[-\infty, \infty]$ and has a nonempty domain over the set of positive semidefinite matrices. Using the change of variable $X = PP^T$, take $P \in \mathbb{R}^{n \times k}$, and consider the problem

$$(12) \qquad \min_P \quad g(P) \overset{def}{=} f(PP^T).$$

Let $\bar{X} = \bar{P}\bar{P}^T$, where $\bar{X}$ is feasible for (11). Then $\bar{X}$ is a local minimizer of (11) if and only if $\bar{P}$ is a local minimizer of (12).

The proof of Theorem 1 is deferred to Appendix A. Using the SDP formulation of the nuclear norm presented in [15], our problem can be recast as an SDP so that we can apply Theorem 1. Define

$$Z = \left[\begin{array}{c} U \\ V \end{array} \right] \left[\begin{array}{c} U \\ V \end{array} \right]^T = \left[\begin{array}{cc} UU^T & L \\ L^T & VV^T \end{array} \right].$$

The matrix Z is positive semidefinite, and has form $Z = PP^T$ with $P = \left[\begin{smallmatrix} U \\ V \end{smallmatrix} \right]$. Let $\mathcal{R}(\cdot)$ be the function that extracts the upper-right block of a matrix (so that $\mathcal{R}(Z) = L$), and let

$$f(Z) = \frac{\lambda_L}{2} \operatorname{trace}(Z) + \varphi(\mathcal{R}(Z)).$$

We now see that the rank-constrained problem is equivalent to

$$\min_{Z \succeq 0} \quad f(Z) \quad \text{such that} \quad \operatorname{rank}(Z) \leq k.$$

Applying Theorem 1, we can be assured that $\overline{P}$ is a local minimizer to the split program if and only if $\overline{Z}$ is a local minimizer of the original problem. This is equivalent to the statement that the point $(\overline{U}, \overline{V})$ is a local minimizer of the split problem if and only if $\overline{L} = \overline{U} \, \overline{V}^T$ is a local minimizer of the original, rank-constrained program.

3.4. Certificate of convergence. Although Theorem 1 asserts that the split problem and the rank-constrained problem have the same local minima, the rank-constrained problem is itself nonconvex, so we have not guaranteed that every stationary point of the split problem solves the convex program. Using an approach that builds on [44], we develop a certificate to check whether a recovered solution to (8) corresponds to a spurious stationary point when φ is convex (in particular, when $\mathcal{L}$ and r are convex). This technique can also be used on the convex formulation as a method to check the distance to optimality.

We base our certificate on the following analysis. Let F be any proper convex function, then Fermat's rule states that the set of minimizers of F are points L such that $0 \in \partial F(L)$, where $\partial(\cdot)$ denotes the subdifferential. One could take a candidate point L, construct $\partial F(L)$, and if it contains 0, conclude that L is the minimizer, but this is not realistic for large-scale problems. Instead, we find

$$\mathcal{E} \in \partial F(L),$$

where $\mathcal{E}$ is small. Let $L^\star$ denote any minimizer of F, then from the definition of a subgradient, $F(L^\star) \geq F(L) + \langle \mathcal{E}, L^\star - L \rangle$. Then

$$(13) \qquad F(L) - F(L^\star) \leq \langle \mathcal{E}, L - L^\star \rangle \leq \|\mathcal{E}\|_p \cdot \|L - L^\star\|_d$$

for any pair of primal and dual norms $\| \cdot \|_p$ and $\| \cdot \|_d$. We discuss bounding $\|\mathcal{E}\|_p$ and $\|L - L^\star\|_d$ in the next two subsections.

Finding an approximate zero $\mathcal{E}$. The key to the certificate is analyzing the marginalized problem (6) in terms of L, rather than the problem (3) in terms of (L, S). The point $L = UV^T$ is the optimal solution of (6) if and only if

$$0 \in \partial\big(\|L\|_* + \varphi(L)\big).$$

We have set $\lambda_L = 1$ for convenience or, alternatively, one can absorb λ_L^{-1} into φ. Since

both the nuclear norm and φ are proper lower semicontinuous and the intersection of the interior of their domains is nonempty (in particular, they are finite valued), then by [7, Cor. 16.38], we have that

$$0 \in \partial\big(\|L\|_* + \varphi(L)\big) \quad \Leftrightarrow \quad 0 \in \big(\partial\|L\|_*\big) + \big(\partial\varphi(L)\big).$$

Both of these subdifferentials are computable. Let

$$L = \widetilde{U}\Sigma\widetilde{V}^T = \begin{bmatrix} U_1 & U_2 \end{bmatrix} \begin{bmatrix} \Sigma_1 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} V_1^T \\ V_2^T \end{bmatrix}$$

be the (full) SVD of L, where $U_1 \in \mathbb{R}^{m \times r}$ and $V_1 \in \mathbb{R}^{n \times r}$. The subdifferentials of the nuclear norm at L comprise matrices of the form $X = U_1 V_1^T + W$, where U_1, V_1 contain the left and right singular vectors of L that correspond to nonzero singular values,[1] and W satisfies the conditions $U_1^T W = 0$, $W V_1 = 0$ and $\|W\|_2 \leq 1$. Equivalently,
(14)

$$X \in \partial\|L\|_* \quad \text{if and only if} \quad \exists W' \in \mathbb{R}^{m-r \times n-r} \text{ s.t. } \widetilde{U}^T X \widetilde{V} = \begin{bmatrix} I & 0 \\ 0 & W' \end{bmatrix}, \quad \|W'\| \leq 1.$$

We can find an explicit matrix $D \in \partial\varphi(L)$ using (10), and it is unique under the smoothness conditions described in section 3.2.

Altogether, we can guarantee a point $\mathcal{E} \in \partial\|L\|_* + \partial\varphi(L)$ such that

$$\begin{aligned}
\|\mathcal{E}\|_F^2 &= \min_{X \in \partial\|L\|_*} \|X + D\|_F^2 \\
&= \min_{X \in \partial\|L\|_*} \|\widetilde{U}^T (X + D) \widetilde{V}\|_F^2 \\
&= \min_{\|W'\| \leq 1} \left\| \begin{bmatrix} I & 0 \\ 0 & W' \end{bmatrix} + \widetilde{U}^T D \widetilde{V} \right\|_F^2 \\
&= \|I - U_1^T D V_1\|_F^2 + \|U_1^T D V_2\|_F^2 + \|U_2^T D V_1\|_F^2 + \min_{\|W'\| \leq 1} \|U_2^T D V_2 - W'\|_F^2.
\end{aligned}$$

In the first equation, the minimum is achieved since the squared norm is continuous and the set is compact, and (14) is used in the third equation. Each term of the fourth equation can be calculated explicitly, with the last term obtained via projection onto the unit spectral-norm ball, which requires the SVD of the small matrix $U_2^T D V_2$. This bound on $\|\mathcal{E}\|_F$ can then be used in (13).

Most terms above can be computed efficiently, in the sense that U_2 and V_2 never need to be explicitly computed (which is most important when r is small, since then U_2 and V_2 have $m-r$ and $n-r$ columns, respectively), and therefore the computation consists only of matrix products, thin-QR factorizations, and an $r \times r$ SVD, leading to a complexity of $\mathcal{O}(rmn + r^2 \cdot (m+n) + r^3)$. For example, we compute

$$\|U_1^T D V_2\|_F^2 = \|U_1^T D\|_F^2 - \|U_1^T D V_1\|_F^2$$

---

[1]The orthogonal matrices U_1 and V_1 should not be confused with the variables U and V that are used as a factorization of L, as U and V are not necessarily orthogonal. In fact, U_1 and V_1 can be efficiently computed from the factorization $L = UV^T$ by taking the QR-decompositions $U = Q_U R_U$ and $V = Q_V R_V$ so $L = Q_U (R_U R_V^T) Q_V^T$. Perform an SVD on the small inner matrix to write $(R_U R_V^T) = U_R \Sigma V_R^T$ and, hence, the Σ are the nonzero singular values of L and $U_1 = Q_U U_R$ and $V_1 = Q_V V_R$ are the corresponding left and right singular vectors.
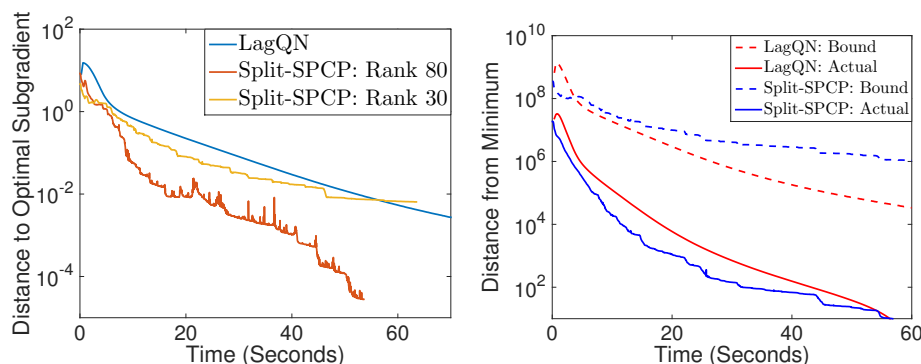
FIG. 1. *(Left): Distance from the optimal subgradient for the convex solver LagQN (defined in section 5) and the nonconvex Split-SPCP with different rank bounds. The rank of the optimal L is 58. (Right): Bounded distance from minimal objective value, as given by $\|\mathcal{E}\|_F(\|L\|_F + F_{\text{bound}})$, with F_{bound} given by the objective value at the current iterate. The actual distance from the minimal objective value is shown for comparison. The rank bound for the Split-SPCP test is 80, and we fit Split-SPCP using L-BFGS.*

and $\|U_2^T D V_1\|_F^2$ is computed analogously. The final term requires an unavoidable $(m-r) \times (n-r)$ SVD factorization, so for large m, n the certificate may be computed as a final check rather than at each iterate.

Bounding the distance to the feasible set. We seek a bound on $\|L - L^\star\|_d$ in an appropriate norm, which together with (13) gives us a bound on the objective function difference $F(L) - F(L^\star)$. Since the bound on $\mathcal{E}$ is in the Frobenius norm, ideally we set $\|\cdot\|_d = \|\cdot\|_F$, but bounds in any norm will work using $\|L\|_F \leq \sqrt{\text{rank}(L)}\|L\|$ and $\|L\|_F \leq \|L\|_*$.

Letting $F(L) = \|L\|_* + \varphi(L)$, we first bound $F(L^\star)$ by computing $F(L)$ for explicit choices of L. To be concrete, in this section we assume $r(S) = \|S\|_1$ with $\lambda_S = 1$, and $\mathcal{L}(\cdot) = \frac{1}{2}\|\cdot\|^2$. Choosing L such that $\mathcal{A}(L) = b$ means that $S = 0$ in the definition of φ (5) and, hence, $F(L) = \|L\|_*$. Choosing $L = 0$ and $S = 0$ gives the bound $F(L) \leq \frac{1}{2}\|b\|^2$. A third choice is to explicitly compute $F(L)$ at all iterates in the algorithm and record the best.

Denoting $F^\star = \min_L F(L)$ and using $F^\star \leq F_{\text{bound}}$, nonnegativity of φ immediately implies $\|L^\star\|_* \leq F_{\text{bound}}$. Hence $\|L - L^\star\|_F \leq \|L\|_F + \|L^\star\|_F \leq \|L\|_F + \|L^\star\|_* \leq \|L\|_F + F_{\text{bound}}$, and $\|L\|_F$ is explicitly computable.

Results. Figure 1 shows how the distance to the optimal subgradient decreases over time for the convex solver LagQN (see section 5 for a description), the nonconvex Split-SPCP with a rank bound that is too strict, and Split-SPCP with a rank bound large enough to reach the global optimum. When the rank bound is too restrictive, Split-SPCP cannot recover the optimal L. Measuring the distance to the optimal L reveals this. Figure 1 shows that the distance plateaus far from zero when the rank of L is bounded above by 30, and the rank of the optimizing L is 58. In practice, this measure can be used to indicate that the rank bound should be increased.

4. Initialization. The nonconvex formulation (8) is sensitive to the initial point, and requires a bound on the rank of L. In this section, we present heuristics for choosing a good initial point and determining an appropriate rank bound.

4.1. Dynamically increasing k. Figure 2 demonstrates the sensitivity of the Split-SPCP program to the factor rank k. These tests were performed on the surveil-
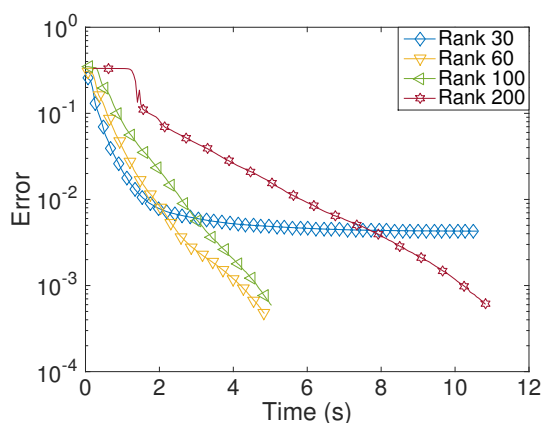
FIG. 2. *Varying the rank bound k in Split-SPCP affects computation time and the accuracy of the final solution. Error is measured as the normalized distance from the objective value at the reference solution. One marker represents* 10 *iterations.*

lance video data described in section 5, where the rank of the low-rank component in our reference solution is 58. We see that when $k \leq \mathrm{rank}(L)$, Split-SPCP does not converge to the correct solution, but if k is much larger than the rank of the minimizing L, then the computation is slowed significantly.

Because our solver is oblivious to the individual dimensions of U and V, columns can be added to both matrices on the fly. Dynamically updating the rank bound can help when (8) is initialized with a k that is too small. Suppose a solver reaches some convergence criterion at iteration i. To see if this is a solution to (6), we add a single column to U and V:

$$\begin{bmatrix} U_i & u \end{bmatrix} \begin{bmatrix} V_i & v \end{bmatrix}^T = L_i + uv^T,$$

and observe whether this rank-one allows for a lower objective value or certificate value. Dynamically increasing k, even aggressively, is more efficient than overestimating k from the start.

4.2. Choosing U_0 and V_0. Choosing an initialization point (U_0, V_0) has received considerable attention in factorized low-rank recovery problems. Most of this work has centered around matrix completion using alternating minimization; see, for example, [21, 23, 25, 26, 44]. Provably good choices for U_0 and V_0 are $U_0 = U\Sigma^{\frac{1}{2}}$ and $V_0 = V\Sigma^{\frac{1}{2}}$, where U and V come from the SVD of $\mathcal{A}^*(X)$ (where $\mathcal{A}^*$ is the adjoint of $\mathcal{A}$). [21] showed that these initial points lie in the basin of attraction of globally optimal minima, yielding fast convergence to an optimal solution.

As mentioned in [44], it is sometimes not necessary to perform a full SVD to form U_0, V_0. Once k is chosen, only a partial SVD is necessary to calculate the first k singular values and vectors, which can be done efficiently using the randomized SVD (rSVD) [20]. Although using the rSVD is significantly faster than a full SVD, the values and vectors it returns are not always accurate, especially when singular values of the underlying data do not decay rapidly. As a general rule, the quality of the initialization is directly proportional to how well the product UV^T approximates $\mathcal{A}^*(X)$.

Figure 3 shows the performance of Split-SPCP using various initial points. The tests were performed on a $5,000 \times 5,000$ matrix $X = L + S$, where $\mathrm{rank}(L) = 1000$ and $\mathrm{sparsity}(S) = 9.43\%$. The rank bound was $k = 1,050$. All 5,000 singular value
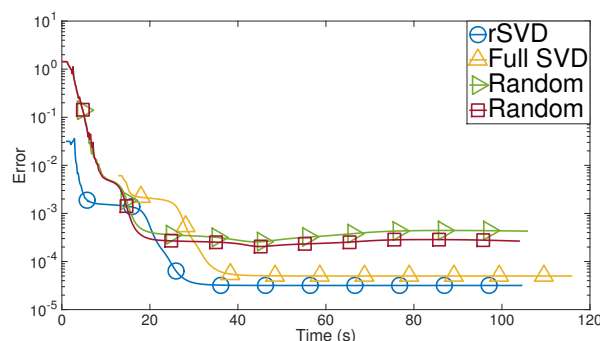
Fig. 3. *The performance of Split-SPCP with various initial points. All tests were run on a $5{,}000 \times 5{,}000$ matrix $X = L + S$, where $\operatorname{rank}(L) = 1000$ and $\operatorname{sparsity}(S) = 9.43\%$. Error is measured as the normalized distance from the objective value of the reference solution. The rank bound was $k = 1{,}050$. One marker represents 50 iterations.*

and vector triples of X were calculated for the SVD test, and U_0 and V_0 were formed from the first $k = 1{,}050$ triples. For the rSVD test, only the first $1{,}050$ triples were approximated, U_0 and V_0 were formed from these.

Figure 3 shows that initializing U_0 and V_0 with the first $1{,}050$ singular value and vector triples returned by the full SVD yields the smallest initial error, but the extra computational cost is not worth the better start. The rSVD allows quicker convergence to a similar solution. The two random initializations converge to a stationary point that is not globally optimal.

5. Numerical experiments. In this section, we present numerical experiments that illustrate the speed of Split-SPCP. Recall that the Split-SPCP program is given by

$$\min_{U,V} \quad \frac{\lambda_L}{2}(\|U\|_F^2 + \|V\|_F^2) + \varphi(UV^T),$$

where

$$\varphi(UV^T) = \min_S \frac{1}{2}\|UV^T + S - X\|_F^2 + \lambda_S\|S\|_1.$$

Since the objective of Split-SPCP is smooth, any first-order method can be used to solve it. Also, because the communication-heavy SVD step is no longer a limitation, we are motivated to choose a solver and an implementation that are most suited for the GPU.

5.1. Implementation details. To find a solver that fits our Split-SPCP program well, we compare the performance of several first-order methods as they solve a logistic regression problem on the GPU. Each of the solvers we use in this test is modified from [39], using the MATLAB Parallel Computing Toolbox, which in turn uses MAGMA libraries [43] to run on the GPU. For all GPU computations, we use a Tesla K40c GPU card with thread block sizes $[1024, 1024, 64]$, grid sizes $[2.15 \times 10^9, 6.6 \times 10^4, 6.6 \times 10^4]$, a clock rate of 745 MHz, and 12.0 GB of total memory. For all of the tests run on the CPU in this section, we compute on an Intel Xeon CPU E5-2630 version 3, using a 64-bit architecture and a clock rate of 2.40GHz. Our version of MATLAB is R2016b.

The time it takes for various solvers to minimize a logistic loss with a Tikhonov regularization term for 10^4 labels is shown in Table 1 below. We measure performance

TABLE 1

First-order methods solving logistic loss program with 10^4 labels on the GPU. All times are reported as a ratio with respect to the fastest solver (L-BFGS with 10 iterations in memory). Time is recorded when the solution is within 10^{-8} of the optimal value.

Solver	Normalized computation time
Cyclic steepest descent	3.46
Barzilai–Borwein	1.24
Conjugate gradient	1.44
Scaled conjugate gradient	1.53
Preconditioned conjugate gradient	3.20
L-BFGS (10 iterations in memory)	**1**
L-BFGS (50 iterations in memory)	1.82
BFGS	2.39
Hessian-Free Newton's method	3.05

TABLE 2

References for the algorithms used in testing. Among these are the fastest models for background subtraction, as determined in [10] and [41].

Algorithm	Reference
Split-SPCP	**This Work**
RPCA-Gradient Descent (RPCA-GD)	[49]
Lagrangian Quasi-Newton (LagQN)	[2]
Max Quasi-Newton (MaxQN)	[2]
Fast PCP (FPCP)	[38]
Go Decomposition (GoDec)	[51]
Inexact ALM (IALM)	[31]
LMaFit (LaMaFit)	[46]

in "computation time," the time it takes for each solver to come within 10^{-8} of the optimal objective value.

We see from our analysis that the L-BFGS solver, storing a small number of iterations in memory, yields the best performance. Of course, the relative performance of different solvers on the GPU depends on the stopping tolerance as well as the specific problem, but there is further evidence in the literature supporting L-BFGS as a competitively efficient solver [47].

5.2. SPCP for background subtraction. In [10] and [41], the authors provide a review of the fastest algorithms in the family of robust PCA and principal component pursuit for background subtraction. We choose some of the fastest of these to compare to our Split-SPCP algorithm (see Table 2). Each of these methods runs faster on the GPU than the CPU, so we run all of them on the GPU for our comparisons (using the MATLAB Parallel Computing Toolbox). We find that our nonconvex solver is particularly well suited for the GPU, and *outperforms all other solvers in almost every case.*

We have included convex and nonconvex algorithms in our comparisons. The two nonconvex algorithms, RPCA-gradient descent (RPCA-GD) and LMaFit in Figure 4, are of particular interest because they both employ a factorization of the low-rank component $L = UV^\top$, which is similar to our model. Also, as discussed in section 2, RPCA-GD has the fastest known runtime for any RPCA algorithm.

For our background subtraction tests, we use the escalator surveillance video provided by [30]. We want to identify the people in the video while ignoring the moving
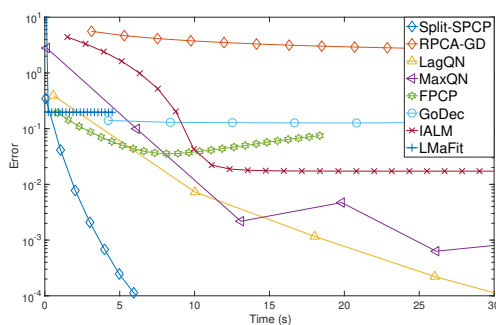
FIG. 4. *Comparing SPCP solvers for background subtraction. One marker corresponds to* 40 *iterations, except for RPCA-GD, where every fifth iterate is labeled.*

escalators and the stationary background. This problem is particularly difficult for many low-rank recovery algorithms, such as PCA, because the motion of the escalator is a confounder. SPCP is less sensitive to outliers, so it can overcome this challenge.

To find a reference solution, we hand tune the parameters λ_L and λ_S in the (convex) quasi-Newton Lagrangian SPCP algorithm (LagQN) until we find a qualitatively accurate decomposition. This process produces a reference low-rank component L_{ref} and sparse component S_{ref}. The L_{ref} we choose has rank 58, and S_{ref} is 58.03% sparse. The optimal parameters for both Split-SPCP and LagQN are $\lambda_L = 115$ and $\lambda_S = 0.825$. We tune the parameters in the other solvers to recover this solution as closely as possible.

Throughout this section, we measure error as the normalized difference between the objective and the objective at the reference solution. Since the solvers minimize different objectives, we calculate the Split-SPCP objective value at each iteration for every algorithm, and we do not include these calculations in our time measurements. To initialize Split-SPCP, we use the first 100 singular values and vectors from the randomized SVD of the data matrix.

Several of the algorithms in Figure 4 do not converge to the same solution, despite considerable effort in parameter tuning. These algorithms were designed to quickly find approximate solutions to the SPCP problem, and might suffer from the large amount of noise present in the data. The approximate solutions recovered by these algorithms are qualitatively different from the Split-SPCP solution; see Figure 5. We also find that a lower objective value generally corresponds to a qualitatively superior solution.

To quantitatively measure the quality of the solutions, we use the *corrected Akaike information criterion*, or AIC_C. The AIC_C measures the fit of a statistical model to a certain data set. Given a model with p parameters and a log-likelihood function ℓ, the value of the AIC_C is

$$\text{AIC}_\text{C} = 2(p - \log(\ell_{\max})) + \frac{2p(p+1)}{m \cdot n - p - 1},$$

where $m \cdot n$ is the size of the data set and $\ell_{\max}$ is the maximum of ℓ. The preferred statistical model is the one that minimizes the AIC_C. The AIC_C favors models that maximize the likelihood function and penalize complex models with many parameters, guarding against overfitting. Using AIC_C as a measure of quality avoids both overemphasis of objective values and the inherent ambiguity of visual comparisons.

(a) X,L, and S matrices found by (from top to bottom) Split-SPCP, LagQN, and GoDec.

(b) X,L, and S matrices found by (from top to bottom) FPCP, IALM, and LMaFit.
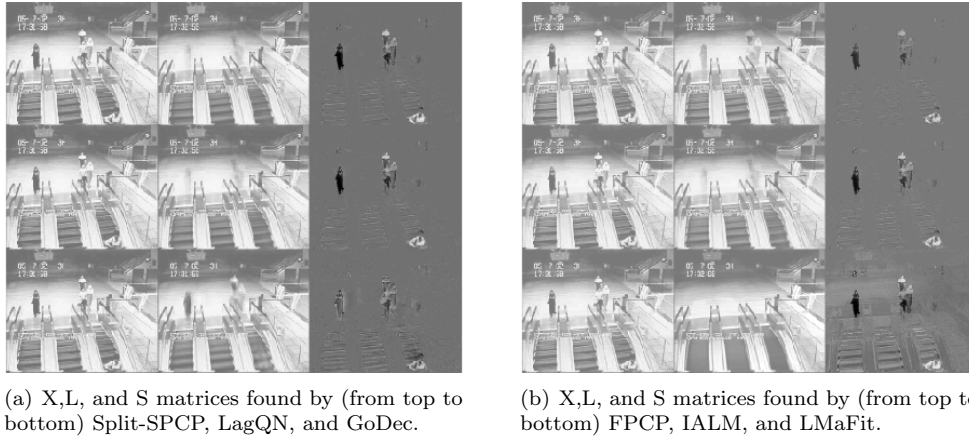
FIG. 5. *Background subtraction using various SPCP solvers on surveillance video data from* [30] *(frame* 10 *is shown). We see that Split-SPCP, LagQN, and IALM best locate the people while ignoring the escalators.*

To compute the AIC_C value, we must formulate SPCP as a statistical model. It is well known that the least-squares loss term is the log-likelihood of a Gaussian distributed random variable, i.e., we assume that the data are of the form $X = L+S+Z$, where the entries of Z are independent and identically distributed (i.i.d.) Gaussian random variables with $\mu = 0$ and variance estimated using the sample variance, $\hat{\sigma}^2 = \frac{\|X\|_F^2}{m \cdot n}$. Similarly, the ℓ_1-regularizer assumes that the entries of S are drawn i.i.d. from the Laplace distribution with mean 0 and variance estimated as $\hat{b} = \frac{\|S\|_1}{m \cdot n}$. The nuclear norm of L is the ℓ_1-norm of its singular values, so its corresponding prior assumes that the singular values of L follow the Laplace distribution. The log-likelihood function is then

$$\ell(L, S, X) = -\left(\frac{m \cdot n}{2}\right)\log(2\pi\sigma^2) - \frac{\|L + S - X\|_F^2}{2\sigma^2}$$
$$- (m \cdot n)\log(2b) - \frac{\|S\|_1}{2b} - \text{rank}(L)\log(2b_*) - \frac{\|L\|_*}{2b_*},$$

where σ^2, b, and b_* are computed according to their respective estimator. We must also define the number of parameters of this model, which is equal to its degrees of freedom. Each term in SPCP provides the following degrees of freedom:

$$\text{rank}(L) = k \quad \rightarrow \quad k(m + n - k) \text{ degrees of freedom},$$
$$\|S\|_1 \quad \rightarrow \quad nnz(S) \text{ degrees of freedom},$$
$$\tfrac{1}{2}\|L + S - X\|_F^2 \quad \rightarrow \quad \left(\frac{\|L+S-X\|_F^2}{\|X\|_F^2}\right)(m \cdot n) \text{ degrees of freedom}.$$

Here, $nnz(S)$ counts the number of nonzero entries in S. Finding the degrees of freedom in a rank-k matrix or a sparse matrix are both standard calculations. For the loss term, we use the residual effective degrees of freedom as an estimate of the flexibility that it introduces. This is detailed further in [32]. The number of parameters p is equal to the total degrees of freedom.

The AIC_C values for the solutions shown in Figure 5 are listed in Table 3. The AIC_C value for the reference solution is listed under "oracle." The reference solution

TABLE 3
Degrees of freedom in the solutions shown in Figure 5.

Solver	AIC$_\text{C}$ ($\times 10^6$)
Oracle	7.37
Split-SPCP	**7.64**
GoDec	10.95
LagQN	11.30
MaxQN	11.64
FPCP	15.70
IALM	2.43×10^7
LMaFit	3.99×10^9

was found using the LagQN solver with a tight tolerance. All the other values in the table correspond to solutions that meet comparable tolerances. Also, since the IALM model does not use an ℓ_1-norm regularizer, many of the values in the returned S matrix were small but not exactly zero. To make accurate comparisons, we applied the shrinkage operator to the S matrix returned by IALM to set small values equal to zero before calculating the degrees of freedom. The values for IALM and LMaFit are high because these solvers are not generally robust to large amounts of noise. With the AIC$_\text{C}$ metric as well, we see that Split-SPCP discovers the best solution, and in a much shorter time compared to the other algorithms.

5.3. Synthetic data. Split-SPCP demonstrates similar performance on synthetic data. For our synthetic-data test, we create two $1{,}000 \times 150$ random matrices with each entry drawn independently from the univariate normal distribution, and define our low-rank reference matrix, L_ref, as the product of these two matrices. To form the sparse reference matrix, S_ref, we fill 50% of a $1{,}000 \times 1{,}000$ matrix with numbers drawn independently from the univariate normal distribution, and the other entries we set equal to zero. X is then the sum $L_\text{ref} + S_\text{ref}$ with added white Gaussian noise, so that $\frac{\|L_\text{ref}+S_\text{ref}-X\|_F}{\|X\|_F} = 8.12 \times 10^{-5}$. Matrices with these characteristics are often encountered in video processing applications.

We initialize Split-SPCP with the first 200 singular value and vector triples of X, so that k is larger than the rank of L. These triples are found using the rSVD. The tuning parameters are set to $\lambda_L = 2.95$ and $\lambda_S = 0.1$. As before, we measure performance based on the normalized difference from the true Split-SPCP objective value. The results are shown in Figure 6.

We see that FPCP and GoDec discover approximate solutions quickly, but the approximations are not within 10^{-2} of the true objective. LMaFit exhibits poor performance. This could be due to the fact that S is about 50% sparse, and LMaFit struggles with problems that have a large sparse component [46]. Split-SPCP significantly outperforms all solvers in speed, and outperforms all solvers except for LagQN in accuracy. LagQN and Split-SPCP converge to solutions with the same objective value, albeit Split-SPCP converges much more quickly.

Split-SPCP has an advantage because it parallelizes well on GPU architectures. This is due to the relative simplicity of the L-BFGS algorithm, which avoids complex linear algebraic decompositions that require heavy communication, such as the QR and SVD (see also [16]). Figure 7 shows that although Split-SPCP on the CPU is slower than LagQN on the CPU, Split-SPCP enjoys enormous speedup when it is implemented on the GPU.
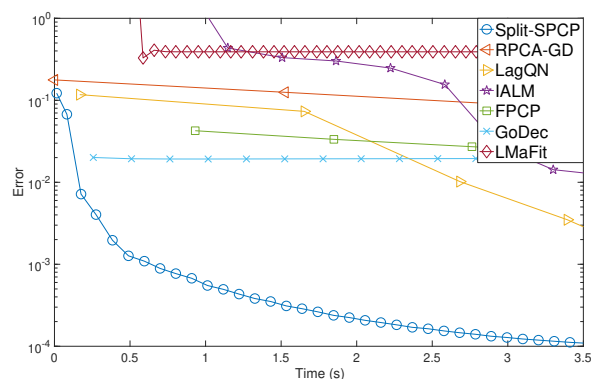
Fig. 6. *Performance of SPCP solvers on the synthetic data described in section 5.3 on the GPU. Error is measured as normalized difference from the solution's objective value. One marker represents six iterations.*
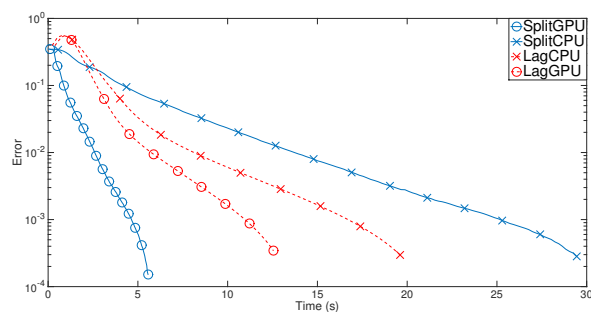

Fig. 7. *Comparing the performance of Split-SPCP and LagQN on the CPU and the GPU. Although Split-SPCP is slower than the similar convex solver on the CPU, it sees more acceleration on the GPU because it avoids QR and SVD computations. Tests were performed on the synthetic data described in section 5.3.*

5.4. Data imputation. To demonstrate the flexibility of our framework, we also test Split-SPCP on a synthetic data imputation task. We consider the problem of imputing entries of a low-rank matrix from partial observations in the presence of noise and sparse corruptions. Our model takes the form

$$\min_{L,S} \quad \lambda_L \|L\|_* + \frac{1}{2}\|\mathcal{P}_\Omega(L + S - b)\|_F^2 + \lambda_S\|S\|_1, \tag{15}$$

where Ω is a support set containing observed indices, and $\mathcal{P}_\Omega$ is a "masking operator." Ignoring vectorization, the action of $\mathcal{P}_\Omega$ is as follows:

$$(\mathcal{P}_\Omega X)_{i,j} = \begin{cases} X_{i,j}, & (i,j) \in \Omega, \\ 0, & (i,j) \notin \Omega. \end{cases}$$

This is a regularized low-rank model since by observation, minimizing

$$\frac{1}{2}\|\mathcal{P}_\Omega(L + S - b)\|_F^2 + \lambda_S\|S\|_1$$

in S gives the same set of solutions as minimizing

$$\frac{1}{2}\|\mathcal{P}_\Omega(L) + S - \mathcal{P}_\Omega(b)\|_F^2 + \lambda_S\|S\|_1$$
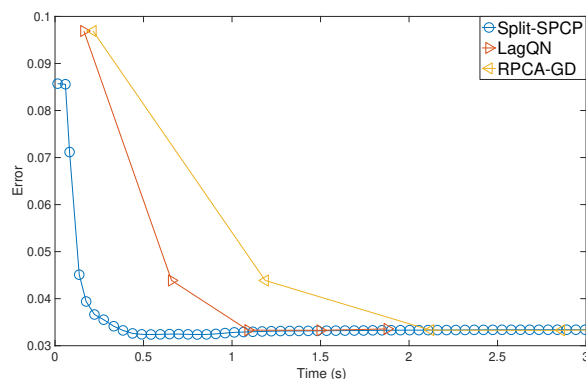
in S.

FIG. 8. *Performance of SPCP solvers on the synthetic data imputation task described in section 5.4. Error is measured as the normalized distance from the reference low-rank component: $\frac{\|L-L_{ref}\|_F}{\|L_{ref}\|_F}$. One marker represents six iterations.*

To create our test, we form L_{ref}, S_{ref}, and X as described in the previous section, but with $\text{rank}(L_{\text{ref}}) = 100$, S_{ref} with 10% sparsity, and $\frac{\|L_{\text{ref}}+S_{\text{ref}}-X\|_F}{\|X\|_F} = 4.92 \times 10^{-3}$. Our masking operator hides 50% of the entries of X, chosen uniformly at random. We use the parameter values $\lambda_L = 1.23$ and $\lambda_S = 0.01$.

Not all of the algorithms we have tested previously can be easily extended to solve (15); we perform tests with all of the algorithms that can: Split-SPCP, LagQN, and RPCA-GD. Because we are interested specifically in the recovery of the low-rank component, our error is measured as the normalized distance from the reference low-rank component: $\frac{\|L-L_{\text{ref}}\|_F}{\|L_{\text{ref}}\|_F}$. As before, all algorithms were run on the GPU. The results are shown in Figure 8. Split-SPCP finds a solution of similar quality to LagQN and RPCA-GD, but does so much faster: in under 0.5 seconds, compared to 1.1 seconds for LagQN and 2.1 seconds for RPCA-GD.

5.5. Dimensional scaling and real-time video processing. Background subtraction in video processing is a standard application for low-rank recovery algorithms, but these algorithms are generally unable to perform real-time background subtraction due to their slow implementation. The experiments in section 5.2 show that Split-SPCP is faster and more accurate than the methods reviewed in [10] and [41] for background subtraction and, in this section, we use Split-SPCP for the real-time processing of high-resolution videos with standard frame rates. We also explore how Split-SPCP scales as the resolution and frame rate increase.

To measure the performance of Split-SPCP on more diverse video datasets, we use videos from the Background Model Challenge [45]. The experiments shown in Figure 9 consider a video with 240×320 pixels per frame and 32,964 frames, which is substantially larger than the low-resolution escalator video of section 5.2 with 130×160 pixels per frame and only 200 frames. We unfold the three-dimensional video data into a matrix with each column containing one frame, and we partition the matrix columnwise into blocks comprising a fixed number of frames. We find that a rank bound near 60 is large enough to capture the low-rank background. With the rank bound fixed at 60, we record the computation time required to perform 200 iterations of Split-SPCP on one block as a function of the block size. Larger block sizes yield better solutions, but they also require more time for SPCP to converge. We find that after 200 iterations, Split-SPCP converges to within 10^{-3} of the optimal objective value for all block sizes, and the solution is qualitatively similar to the optimal solution.
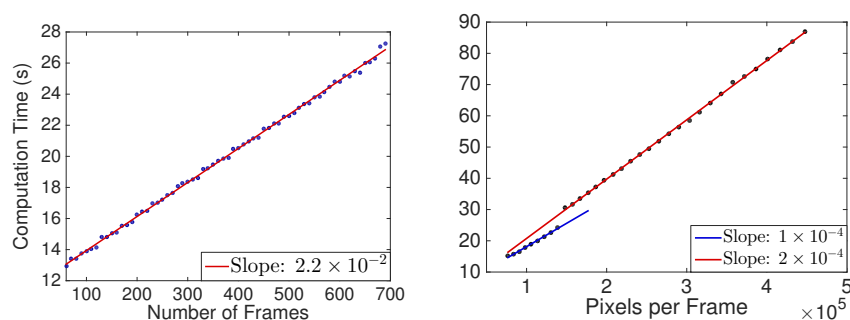
FIG. 9. *The scaling of Split-SPCP as the number of columns (or frames) increases (left) and as the number of rows (or pixels per frame) increases (right). Each point is the average of twenty trials. For the column-scaling tests, the resolution is fixed at 240×320 pixels per frame, and for the row-scaling tests, the number of frames in a block is fixed at 100. The tuning parameters are set to $\lambda_L = 200, \lambda_S = 5$.*

Figure 9 shows how Split-SPCP scales as the video resolution and the number of frames per block increase. These two quantities correspond to the number of rows and the number of columns of each block. We increase the resolution by rescaling each frame so that the aspect ratio stays close to constant, and we use interpolation to impute the missing pixel values. It is easy to discern the $\mathcal{O}(n)$ scaling as the number of frames (columns) is increased. Increasing the resolution affects the computation time linearly, but beyond a "critical resolution" of 332×446 pixels per frame, the slope of the linear scaling undergoes a shift in regime. This linear scaling is expected because the bottleneck of Split-SPCP is the matrix-matrix multiply UV^T, which requires $\mathcal{O}(mnk)$ operations, so with k fixed, the computation time of Split-SPCP grows linearly with the number of data points. The results of the background subtraction are shown in Figure 10.

These experiments suggest conditions under which Split-SPCP can decompose a video in real time. Assuming a resolution of 240×320 pixels per frame and a frame rate of 24 frames per second, which is standard in many applications of video processing, the video must be partitioned in blocks of about 600 frames or more. With this partitioning, the algorithm will have finished decomposing the given block before the next block is recorded. This number increases linearly with resolution, following the trend line given in the rightmost plot of Figure 9. Following these guidelines, Split-SPCP can decompose the videos from [45] in real time without sacrificing quality. An example is shown in Figure 10.

6. A comparison with accelerated-SVD algorithms. There are several algorithms for low-rank matrix recovery that preserve convexity and focus on accelerating the SVD step. In this section, we explore two such approaches: a communication-avoiding rSVD and the Frank–Wolfe method with marginalization. We compare both to our SVD-free approach. Both of these methods preserve convexity, but numerical experiments suggest that our nonconvex formulation is superior.

6.1. Communication-avoiding rSVD. Many problems in data analysis involve matrices that are "tall-skinny." For example, the surveillance video we consider in section 5 has dimensions $20{,}800 \times 200$, and the psychometric data shown in Figure 12 produces a matrix with 230 times more rows than columns. The prevalence of this tall-skinny structure is due in part to the fact that multidimensional data sets must be "unwrapped" into a two-dimensional matrix before matrix-based algorithms

Fig. 10. *Real-time background subtraction using Split-SPCP. Split-SPCP correctly identifies (left) a car in the middle-ground, (center) a person in the foreground, and (right) a person in the background. The resolution is* 240×320 *pixels per frame,* 600 *frames were decomposed at one time, and only* 200 *iterations were allowed.*

can analyze the data, and the canonical unwrapping of a square data set with more than two dimensions leads to a highly rectangular matrix. Tall-skinny matrices pose a challenge for matrix decompositions, especially on computational architectures where communication is costly (such as a GPU) [14].

The rSVD is often used to accelerate low-rank recovery models by projecting a data matrix of size $m \times n$ onto a low-dimensional subspace, and performing computations with the resulting smaller matrix of size $m \times k$ with $k \ll n$.[2] The rSVD offers improved performance on the CPU, but it exacerbates the tall-skinny problem, making parallelization difficult even when the input matrix is square.

The effect of this increased communication is notable: on the CPU, the rSVD step in LagQN accounts for about 55% of the algorithm's total running time, but when this solver is run on the GPU, the rSVD accounts for about 90% of SPCP's total running time, as shown in Figure 11. The tall-skinny QR (TSQR) algorithm of [14] can provide a convex option for accelerating low-rank recovery models. While TSQR has been used to accelerate the SVD step in RPCA for tall-skinny matrices [1], we extend this idea to the rSVD algorithm on general matrices, and we compare this communication-avoiding approach to Split-SPCP for analyzing functional MRI (fMRI) brain scan data.

For the experiments in this section, we implement the TSQR algorithm on a single GPU card as well as multiple CPUs. The GPU card and CPUs are the same models described in section 5.1. We use 32 CPU cores: two sockets, eight cores per socket, and two threads per core. To run QR decompositions in parallel on the GPU, we use the batched QR algorithm of MAGMA version 1.7.0 [43], compiled as a MEX file using GCC for Red Hat version 4.8.5-11. We wrote our own implementation of TSQR on multiple CPUs using the MATLAB Parallel Computing Toolbox. The initial number of blocks in our TSQR implementation is 32, corresponding to the 32 CPU cores we have available. Specific details on the TSQR algorithm can be found in [14].

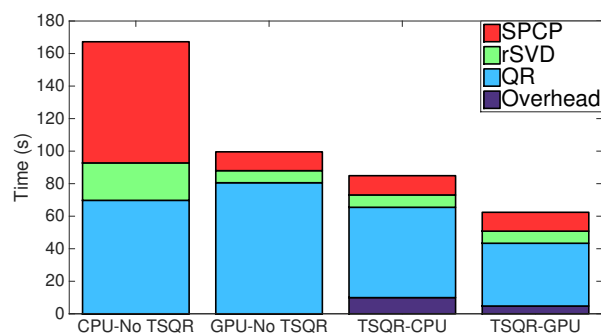[2]Further details of the rSVD algorithms can be found in [20].

FIG. 11. *The time spent performing subprocesses of the SPCP algorithm on fMRI brain scan data. Lower blocks are subprocesses of upper blocks, so the times are cumulative. The test "TSQR-CPU" uses a hybridized architecture, running TSQR on multiple CPUs but running the rest of SPCP on the GPU. Both tests "GPU-No TSQR" and "TSQR-GPU" were run entirely on the GPU. "Overhead" refers to time moving data between the CPU and GPU.*
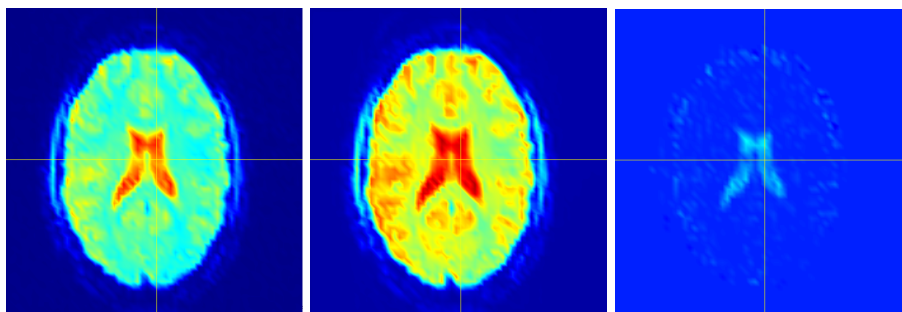

FIG. 12. *The results of SPCP on fMRI brain scan data. Activity is measured in blood-oxygen-level dependent (BOLD) signal. (From left to right: original image, low-rank image recovered by SPCP, and sparse image recovered by SPCP.)*

fMRI brain scan data sets are inherently four-dimensional, so unwrapping the dataset into a two-dimensional array creates a tall-skinny matrix with hundreds of times more rows than columns. The enormity of these data sets makes low-rank recovery models intractable without parallelization. Figure 11 shows the total time spent performing SPCP on one of these data sets, as well as the total time spent performing rSVDs, performing the QR step of the rSVD, and moving data (labeled as the overhead cost). The results of this test are included in Figures 12 and 13.

The datasets used for testing were taken from a study of the human brain's response to uncomfortable stimuli, specifically, the feeling of submerging a hand in hot or cold water. Analyzing these scans to find common neurological responses to these stimuli is difficult due to the enormous amount of error in the data. There is uniformly distributed noise due to constant ancillary physiological activity, and there are also sparsely distributed groups representing neurological structures that should all exhibit the same behavior. The ventricles, for example, are filled with cerebrospinal fluid (CSF), which does not contribute to neurological communication, so they should not be active. All signals observed in the ventricles should be treated as sparsely structured outliers. SPCP removes the uniform noise and, most remarkably, correctly identifies signals in the brain's ventricles as outliers. In Figure 12, the largest
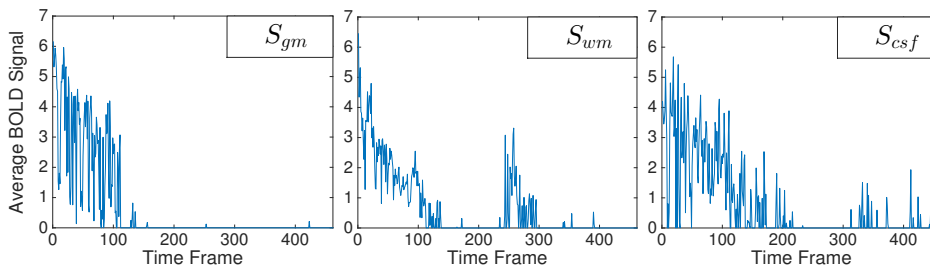
FIG. 13. *The average BOLD signal in different regions of S. The averages were normalized by the average original signal in the corresponding regions.*
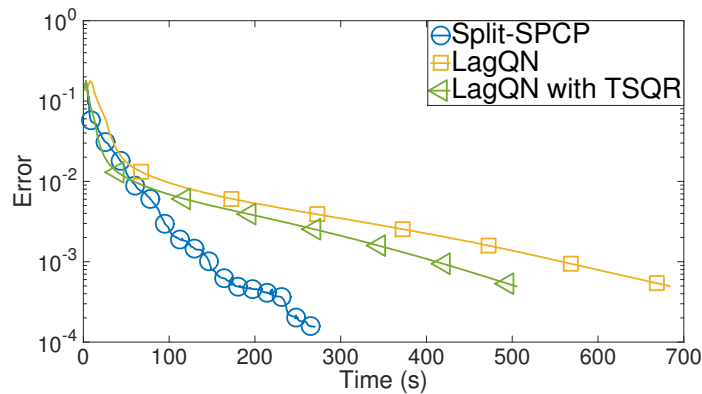

FIG. 14. *Comparing Split-SPCP to LagQN using TSQR on the GPU and without using TSQR. Split-SPCP was initialized with the rank bound $k = 462$, the maximum possible rank of the low-rank component. One marker represents 35 iterations.*

ventricles are the two structures in the center of the brain. The rightmost image shows that the majority of the noise contained in S is from these ventricles.

The other two major components of the brain are white and gray matter. The activity we are hoping to observe takes place in the gray matter, so ideally SPCP would remove most signals from the white matter regions. However, the regions of white matter are more difficult to distinguish than the regions of CSF, and SPCP removes about equal amounts of noise from the white matter as it does from the gray. If we let S_{gm} be the gray-matter component of S, and define S_{wm} and S_{csf} similarly, Figure 13 shows the average BOLD signal in S_{gm}, S_{wm}, and S_{csf} for each frame in time. These data were normalized by the average original signal in the corresponding regions.

It is clear that S_{csf} contains more signal than the other two regions. For S_{gm}, SPCP detects noise in only the first 100 time slices. The removed signal from the white matter is more distributed over time, and the total amount of noise in S_{gm} and S_{wm} is comparable. These results suggest that SPCP correctly identifies outliers in the fMRI data, especially within the regions of CSF.

Although using TSQR to reduce communication costs provides noticeable speedup, the nonconvex approach is faster. In Figure 14, we compare our nonconvex Split-SPCP algorithm to a convex solver using TSQR for the rSVD step. Both solvers were decomposing the same $106{,}496 \times 462$ brain-scan data matrix used in section 6.4, and both were run on the GPU.

Downloaded 01/04/19 to 128.111.121.42. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

As with previous tests, a reference solution $(L_{\mathrm{ref}}, S_{\mathrm{ref}})$ was found by solving LagQN to high accuracy, and the error is measured as the normalized distance from the reference objective value. The rank of L_{ref} was 423, and Split-SPCP was initialized with the rank bound $k = 462$. Since X had 462 columns, this was the largest possible rank bound, and Split-SPCP still greatly outperformed the convex solvers. Both solvers also found solutions of similar quality. At an error of 5×10^{-4}, the low-rank component found by LagQN had rank 427, and the low-rank component found by Split-SPCP had rank 425. Similarly, the sparse component of LagQN had sparsity 58.5%, and the sparse component of Split-SPCP had sparsity 58.3%, while S_{ref} was 58.5% sparse. Split-SPCP recovered nearly the same solution as the parallelized convex algorithm at a much smaller cost.

6.2. Frank–Wolfe method with marginalization. Our approach to accelerating low-rank recovery models is closely connected to the one presented in [33] to accelerate the Frank–Wolfe method for RPCA. The Frank–Wolfe method, first proposed in [17], has recently gained attention for its ability to yield scalable algorithms for optimization problems involving structure-encouraging regularizers [22, 33]. Frank–Wolfe methods optimize a Lipschitz differentiable function over a convex set by solving a sequence of linear subproblems. For example, if f is Lipschitz differentiable and $\mathcal{C}$ is a convex set, Frank–Wolfe solves the problem

$$\min_{x \in \mathcal{C}} \quad f(x)$$

by linearizing f about each iteration x_k: $f(y) \approx f(x_k) + \langle \nabla f(x), y - x \rangle$, minimizing the linear problem to find a descent direction $y_k - x_k$, and then stepping in this direction with a step size η: $x_{k+1} = x_k + \eta(y_k - x_k)$. Traditionally, $\eta = \frac{2}{k+2}$ is fixed for convergence guarantees, but better performance can be achieved with adjustable step sizes [33].

For optimization problems involving the nuclear norm, the set $\mathcal{C}$ is a scaled nuclear norm ball, and the linear subproblem becomes a search for the leading singular tuple of L_k. This is much cheaper than computing the full SVD at each iteration, which is required by proximal methods.

In contrast to nuclear norm minimization problems, it is often difficult to scale the Frank–Wolfe method for problems involving ℓ_1 regularization. For these problems, each Frank–Wolfe iteration updates iterate S_k with only a single nonzero element [33]. This makes the Frank–Wolfe method infeasible when the sparse component of the data-matrix has several thousand nonzero elements or more. The authors of [33] adapt the Frank–Wolfe method to solve SPCP, using the traditional Frank–Wolfe update for the low-rank component and using a projected gradient update for the sparse component. Their adaptations allow for significant speedup and better convergence compared to the traditional Frank–Wolfe method.

The techniques used in [33] can be seen as a special case of the marginalization we present in this work. With the sparsely structured component marginalized, the program depends only upon the low-rank component, so the benefits of the Frank–Wolfe scheme are preserved and its drawbacks are negated. Algorithm 2 shows how marginalization can be used to extend the adaptations presented in [33, Algorithms 5 and 6]. Algorithm 2 solves the following program, which is equivalent to SPCP:

$$\min_{\|L\|_* \leq t} \quad \lambda_L t + \min_S \frac{1}{2}\|\mathcal{P}_\Omega(L + S - X)\|_F^2 + \lambda_S \|S\|_1.$$

Algorithm 2: Frank–Wolfe method with marginalization.

Input : $\mathbf{L_0}, \lambda_L, \lambda_S, t_0 = 0, U_0 = \frac{1}{2\lambda_L}\|\mathcal{P}_\Omega(\mathbf{L_0} + \mathbf{S_0} - \mathbf{X})\|_F^2$

1 **while** Not Converged **do**

2 $\mathbf{S_k} \leftarrow \text{shrink}(\mathbf{X} - \mathbf{L_k}, \lambda_S)$

3 $\mathbf{Y_k} \leftarrow \underset{\|\mathbf{Y}\|_* \leq 1}{\arg\min} \left\langle \mathcal{P}_\Omega(\mathbf{L_k} + \mathbf{S_k} - \mathbf{X}), \mathbf{Y} \right\rangle$

4   **if** $\lambda_L \geq -\left\langle \mathcal{P}_\Omega(\mathbf{L_k} + \mathbf{S_k} - \mathbf{X}), \mathbf{Y_k} \right\rangle$ **then**

5 $\mathbf{V_k} \leftarrow \mathbf{0}$

6 $V_{t_k} \leftarrow 0$

7   **else**

8 $\mathbf{V_k} \leftarrow U_k \mathbf{Y_k}$

9 $V_{t_k} \leftarrow U_k$

10 $\eta^\star \leftarrow \underset{\eta \in (0,1)}{\arg\min} \quad \frac{1}{2}\|\mathcal{P}_\Omega((1-\eta)\mathbf{L_k} + \eta\mathbf{V_k} + \mathbf{S_k} - \mathbf{X})\|_F^2 + (1-\eta)\lambda_L t_k + \eta\lambda_L t_k$

11 $\mathbf{L_{k+1}} \leftarrow (1 - \eta^\star)\mathbf{L_k} + \eta^\star \mathbf{V_k}$

12 $t_{k+1} \leftarrow (1 - \eta^\star)t_k + \eta^\star V_{t_k}$

13 $U_{k+1} \leftarrow \frac{1}{2\lambda_L}\|\mathcal{P}_\Omega(\mathbf{L_0} + \mathbf{S_0} - \mathbf{X})\|_F^2 + t_k$
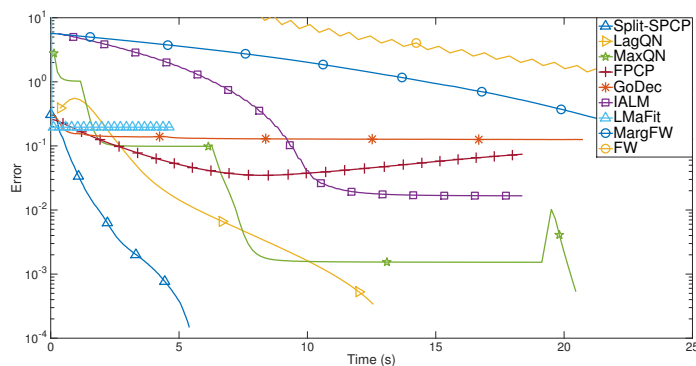

FIG. 15. *Comparing Frank–Wolfe method with marginalization (denoted "MargFW"), as well as another Frank–Wolfe scheme adapted for RPCA* [33] *(denoted "FW"), with other SPCP solvers for background subtraction on the escalator video of* [30]. *One marker corresponds to* 40 *iterations.*

With the S variable marginalized, it does not affect the linear subproblems arising in the Frank–Wolfe scheme, making this approach much more scalable. Each iteration of Algorithm 2 updates the low-rank component and the nuclear norm bound, t, while the sparse component remains only implicitly defined. More details on this approach can be found in [33].

While Algorithm 2 is similar to the methodology in [33], fully marginalizing the sparse component eliminates the linear subproblem required in [33] to update the iterate S_k. We show the performance of this adapted Frank–Wolfe scheme on SPCP in Figure 15. Although its performance is better than the traditional Frank–Wolfe method (see [33]), it is still much slower than solving Split-SPCP with L-BFGS.

7. Conclusion. In this manuscript, we present a general framework for accelerating regularized low-rank recovery models on parallel architectures. Factorizing the low-rank component $L = UV^T$ and optimizing over the factors U and V induces a

low-rank constraint on L while eliminating the iterative SVDs often required for nuclear norm minimization. Although this factorization sacrifices the convexity of the program, we develop a certificate to determine whether our model has converged to a minimum or to a spurious stationary point, showing empirically and theoretically that spurious stationary points do not pose a serious problem for our nonconvex model. We also show that marginalizing the regularized S variable creates an objective that is Lipschitz differentiable when the loss function is strongly convex. The smooth objective allows the use of first-order solvers, which exhibit faster convergence and better parallelization.

We demonstrate the effectiveness of our framework on SPCP, showing that Split-SPCP runs an order of magnitude faster on the GPU than existing solvers. There are other regularized low-rank recovery processes that decrease the cost of the SVD step for improved efficiency, including communication-avoiding and Frank–Wolfe-type algorithms. To compare with Split-SPCP, we propose an improved algorithm in both classes, and show that our SVD-free approach offers greater speedup.

With the acceleration our framework provides, regularized low-rank recovery models can be applied to solve new problems. Split-SPCP can process video streams in real time and decompose extremely large data sets, such as fMRI brain scans, whose analysis with low-rank recovery models would otherwise be infeasible.

Appendix A. Factorization theorem details. Both problems share the same set of local minimizers. This was first observed in the case of SDP in [11] and later generalized in [4].

Our variant of the theorem follows.

THEOREM 2. *Consider an optimization problem of the following form,*

$$(16) \qquad \min_{X \succeq 0} \quad f(X) \quad such \ that \quad \mathrm{rank}(X) \le k,$$

where $X \in \mathbb{R}^{n \times n}$ is a positive semidefinite real matrix, and f is a lower semi-continuous (lsc) function mapping to $[-\infty, \infty]$ and has a nonempty domain over the positive semidefinite matrices. Using the change of variable $X = PP^T$, take $P \in \mathbb{R}^{n \times k}$, and consider the problem

$$(17) \qquad \min_{P} \quad g(P) \overset{def}{=} f(PP^T).$$

Let $\bar{X} = \bar{P}\bar{P}^T$, where $\bar{X}$ is feasible for (16). Then $\bar{X}$ is a local minimizer of (16) if and only if $\bar{P}$ is a local minimizer of (17).

Proof. We follow ideas from both [11] and [4]. From [11, Lem. 2.1], if both P and K are $n \times k$ matrices, then $PP^T = KK^T$ if and only if $P = QK$ for some orthogonal matrix $Q \in \mathbb{R}^{k \times k}$. The objective in (17) depends only on PP^T, so it is clear that P is a local minimizer of (17) if and only if PQ is a local minimizer for all orthogonal Q.

Note that g defined by $P \mapsto f(PP^T)$ is also lsc since it is the composition of f with the continuous function $P \mapsto PP^T$. We require the functions to be lsc so that the notion of "local minimizer" is well-defined.

Suppose $\bar{X}$ is a local minimizer of (16), i.e., for some $\epsilon > 0$ there is no better feasible point, and factor it as $\bar{X} = \bar{P}\bar{P}^T$ for some $n \times k$ matrix $\bar{P}$. Then we claim $\bar{P}$ is a local minimizer of (17). If it is not, then there exists a sequence $P_n \to \bar{P}$ with $g(P_n) < g(\bar{P})$ for all n. By continuity of the map $P \mapsto PP^T$, there is some n large

enough such that $X_n \overset{\text{def}}{=} P_n P_n^T$ is within ϵ of $\bar{X}$ with $f(X_n) < f(\bar{X})$, contradicting the local optimality of $\bar{X}$.

We prove the other direction using contrapositive. Suppose $\bar{X}$ is not a local minimizer of (16), so there is a sequence $X_n \to \bar{X}$ with $f(X_n) < f(\bar{X})$. Factor each X_n as $X_n = P_n P_n^T$, and observe that it is not true that P_n converges. However, (X_n) converges and hence is bounded, thus (P_n) is bounded as well (for the spectral norm, $\|X_n\| = \|P_n\|^2$, and over the space of matrices, all norms are equivalent). Since the P_n are in a finite dimensional space, the Bolzano–Weierstrass theorem guarantees that there is a subsequence of (P_n) that converges. Let the limit of the subsequence be $\bar{P} = \lim_{k\to\infty} P_{n_k}$, and note $\bar{P}\bar{P}^T = \bar{X}$ since $X_{n_k} \to \bar{X}$. Then $g(P_{n_k}) = f(X_{n_k}) < f(\bar{X}) = g(\bar{P})$, so $\bar{P}$ is not a local solution. It also follows from the proof's first paragraph that there cannot be another local solution $\tilde{P}$ that also satisfies $\tilde{P}\tilde{P}^T = \bar{X}$.   □

Remark 1. We recover the settings of both [4, Thm. 4.1] and [11], since allowing f to be extended valued and lsc encompasses constraints.

Remark 2. [4, Cor. 4.2] can be clarified to state that the two problems listed there are "equivalent" in the sense that they share the same local (as well as global) minimizers, using similar continuity arguments on the mapping $\mathcal{R}$ and its adjoint. Furthermore, since the constraints are compact, solutions exist in both formulations.

Appendix B. Convexity of φ.

LEMMA 1. *Let* $f : \mathbb{R}^{m \times n} \times \mathbb{R}^{m \times n} \to \mathbb{R}$ *be a convex function, and define* $g(L) = \min_{S \in \mathbb{R}^{m \times n}} f(L, S)$ *for* $L \in \mathbb{R}^{m \times n}$. *The function* g *is convex.*

Proof. For all $L_1, L_2, S_1, S_2 \in \mathbb{R}^{m \times n}$ and $t \in (0, 1)$,

$$
\begin{aligned}
g(tL_1 + (1-t)L_2) &= \min_{S \in \mathbb{R}^{m \times n}} f(tL_1 + (1-t)L_2, S) \\
&\leq f(tL_1 + (1-t)L_2, tS_1 + (1-t)S_2) \\
&\leq tf(L_1, S_1) + (1-t)f(L_2, S_2).
\end{aligned}
$$

As this inequality holds for arbitrary S_1 and S_2, it holds for any minimizing $S^\star$:

$$
\begin{aligned}
g(tL_1 + (1-t)L_2) &\leq tf(L_1, S^\star) + (1-t)f(L_2, S^\star) \\
&= tg(L_1) + (1-t)g(L_2). \qquad\qquad \square
\end{aligned}
$$

It follows immediately that φ as defined in (5) is convex when $\mathcal{L}$ and r are convex.

REFERENCES

[1] M. ANDERSON, G. BALLARD, J. DEMMEL, AND K. KEUTZER, *Communication-avoiding QR decomposition for GPUs*, in Proceedings of the 2011 IEEE International Parallel and Distributed Processing Symposium, IEEE, IEEE Computer Society, Los Alamitos, CA, 2011, pp. 48–58.

[2] A. ARAVKIN, S. BECKER, V. CEVHER, AND P. OLSEN, *A variational approach to stable principal component pursuit*, in Uncertainty in Artificial Intelligence, Quebec City, Kaufmann, San Mateo, CA, 2014, pp. 32–41.

[3] A. ARAVKIN, J. BURKE, D. DRUSVYATSKIY, M. FRIEDLANDER, AND S. ROY, *Level-Set Methods for Convex Optimization*, Math. Program., (2018) https://link.springer.com/article/10.1007/s10107-018-1351-8.

[4] A. ARAVKIN, R. KUMAR, H. MANSOUR, B. RECHT, AND F. J. HERRMANN, *Fast methods for denoising matrix completion formulations, with applications to robust seismic data interpolation*, SIAM J. Sci. Comput., 36 (2014), pp. S237–S266.

[5] F. BACH, R. JENATTON, J. MAIRAL, AND G. OBOZINSKI, *Optimization with sparsity-inducing penalties*, Found. Trends Mach. Learn., 4 (2012), pp. 1–106.

[6] F. BACH, J. MAIRAL, AND J. PONCE, *Convex Sparse Matrix Factorizations*, preprint, https://arxiv.org/abs/0812.1869, 2008.

[7] H. H. BAUSCHKE AND P. L. COMBETTES, *Convex Analysis and Monotone Operator Theory in Hilbert Spaces*, Springer, New York, 2011.

[8] S. BHOJANAPALLI, A. KYRILLIDIS, AND S. SANGHAVI, *Dropping convexity for faster semi-definite optimization*, in Proceedings of Machine Learning Research, 49 (2016), pp. 530–582.

[9] M. BOGDAN, E. VAN DEN BERG, C. SABATTI, W. SU, AND E. CANDÈS, *SLOPE-adaptive variable selection via convex optimization*, Ann. Appl. Stat., 9 (2015), pp. 1103–1140.

[10] T. BOUWMANS AND E. ZAHZAH, *Robust PCA via principal component pursuit: A review for a comparative evaluation in video surveillance*, Comput Vis. Image Underst., 122 (2014), pp. 22–34.

[11] S. BURER AND R. MONTEIRO, *Local minima and convergence in low-rank semidefinite programming*, Math. Progam., 103 (2005), pp. 427–444.

[12] E. J. CANDÈS, X. LI, Y. MA, AND J. WRIGHT, *Robust principal component analysis?*, J. ACM, 58 (2011), pp. 1–37.

[13] D. DAVIS, *An $O(n \log(n))$ Algorithm for Projecting Onto the Ordered Weighted ℓ_1 Norm Ball*, Technical report CAM 15-32, University of California, Los Angeles, 2015.

[14] J. DEMMEL, L. GRIGORI, M. HOEMMEN, AND J. LANGOU, *Communication-optimal parallel and sequential QR and LU factorizations*, SIAM J. Sci. Comput., 34 (2012), pp. A206–A239.

[15] M. FAZEL, H. HINDI, AND S. P. BOYD, *A rank minimization heuristic with application to minimum order system approximation*, in IEEE Proceedings of the American Control Conference, IEEE, Piscataway, NJ, 2001, pp. 4734–4739.

[16] Y. FEI, G. RONG, B. WANG, AND W. WANG, *Parallel L-BFGS-B algorithm on GPU*, Comput. Graph., 40 (2014), pp. 1–9.

[17] M. FRANK AND P. WOLFE, *An algorithm for quadratic programming*, Naval Res. Logist. Q., 3 (1956), pp. 95–110.

[18] S. GUNASEKAR, A. ACHARYA, N. GAUR, AND J. GHOSH, *Noisy matrix completion using alternating minimization*, in Machine Learning and Knowledge Discovery in Databases, Springer, Berlin, 2013, pp. 194–209.

[19] J. GUO AND A. S. LEWIS, *Nonsmooth variants of Powell's BFGS convergence theorem*, SIAM J. Optim., 28 (2018), pp. 1301–1311.

[20] N. HALKO, P. G. MARTINSSON, AND J. A. TROPP, *Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions*, SIAM Rev., 53 (2011), pp. 217–288.

[21] M. HARDT, *Understanding alternating minimization for matrix completion*, in 2014 IEEE 55th Annual Symposium on Foundations of Computer Science (FOCS), IEEE, Piscataway, NJ, 2014, pp. 651–660.

[22] M. JAGGI, *Revisiting Frank-Wolfe: Projection-free sparse convex optimization*, in International Conference on Machine Learning, Curran, Red Hook, NY, 2013.

[23] P. JAIN, P. NETRAPALLI, AND S. SANGHAVI, *Low-rank matrix completion using alternating minimization*, in Proceedings of the 45th Annual ACM Symposium on the Theory of Computing, ACM, New York, 2013, pp. 665–674.

[24] R. KESHAVAN, *Efficient Algorithms for Collaborative Filtering*, Ph.D. thesis, Stanford University, Palo, Alto, CA, 2012.

[25] R. KESHAVAN AND A. MONTANARI, *Regularization for matrix completion*, in IEEE International Symposium on Information Theory Proceedings (ISIT), IEEE, Piscataway, NJ, 2010, pp. 1503–1507.

[26] R. KESHAVAN, A. MONTANARI, AND S. OH, *Matrix completion from noisy entries*, in Advances in Neural Information Processing Systems, 2009, Curran, Red Hook, NY, pp. 952–960.

[27] C. LEMARECHAL, *Numerical experiments in nonsmooth optimization*, in Progress in Nondifferentiable Optimization, International Institute for Applied Systems Analysis, Luxenburg, Austria, 1982, pp. 61–84.

[28] A. LEWIS AND M. OVERTON, *Nonsmooth optimization via quasi-Newton methods*, Math. Program., 141 (2013), pp. 135–163.

[29] A. LEWIS AND S. ZHANG, *Nonsmoothness and a variable metric method*, J. Optim. Theory Appl., 165 (2015), pp. 151–171.

[30] L. Li, W. Huang, I. Gu, and Q. Trian, *Statistical modeling of complex backgrounds for foreground object detection*, IEEE Trans. Image Process., 13 (2004), pp. 1459–1472.

[31] Z. Lin, R. Liu, and Z. Su, *Linearized alternating direction method with adaptive penalty for low-rank representation*, in Advances in Neural Information Processing Systems, Curran, Red Hook, NY, 2011, pp. 612–620.

[32] C. Loader, *Local Regression and Likelihood*, Springer, New York, 1999.

[33] C. Mu, Y. Zhang, J. Wright, and D. Goldfarb, *Scalable robust matrix recovery: Frank-Wolfe meets proximal methods*, SIAM J. Sci. Comput., 38 (2016), pp. A3291–A3317.

[34] B. K. Natarajan, *Sparse approximate solutions to linear systems*, SIAM J. Comput., 24 (1995), pp. 227–234.

[35] P. Netrapalli, U. N. Niranjan, S. Sanghavi, A. Anandkumar, and P. Jain, *Provable non-convex robust PCA*, in Advances in Neural Information Processing Systems (NIPS), MIT Press, Cambridge, MA, 2014, pp. 1107–1115.

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

[37] R. Rockafellar and R. Wets, *Variational Analysis*, Springer, Berlin, 2009.

[38] P. Rodrígues and B. Wohlberg, *Fast principal component pursuit via alternating minimization*, in Proceedings of IEEE International Conference on Image Processing (ICIP), IEEE, Piscataway, NJ, 2013, pp. 69–73.

[39] M. Schmidt, *minFunc*, https://www.cs.ubc.ca/~schmidtm/Software/minFunc.html (2005).

[40] Y. Shen, Z. Wen, and Y. Zhang, *Augmented Lagrangian alternating direction method for matrix separation based on low-rank factorization*, Optim. Methods Softw., 29 (2012), pp. 239–263.

[41] A. Sobral and A. Vacavant, *A comprehensive review of background subtraction algorithms evaluated with synthetic and real videos*, Comput. Vis. Image Underst., 122 (2014), pp. 4–21.

[42] N. Srebro, J. Rennie, and T. Jaakkola, *Maximum-margin matrix factorization*, in Advances in Neural Information Processing Systems (NIPS), MIT Press, Cambridge, MA, 2005, pp. 1329–1336.

[43] S. Tomov, J. Dongarra, and M. Baboulin, *Towards dense linear algebra for hybrid GPU accelerated manycore systems*, Parallel Comput., 36 (2010), pp. 232–240.

[44] M. Udell, C. Horn, R. Zadeh, and S. Boyd, *Generalized low rank models*, Found. Trends Mach. Learn., 9 (2015), pp. 1–118.

[45] A. Vacavant, T. Chateau, A. Wilhelm, and L. Lequiévre, *A benchmark dataset for outdoor foreground/background extraction*, in ACCV 2012, Springer, Berlin, 2012, pp. 291–300.

[46] Z. Wen, W. Yin, and Y. Zhang, *Solving a low-rank factorization model for matrix completion by a nonlinear successive over-relaxation algorithm*, Math. Program. Comput., 4 (2010), pp. 333–361.

[47] S. J. Wright, *Primal-Dual Interior-Point Methods*, SIAM, Philadelphia, 1997.

[48] Y. Wright, A. Ganesh, S. Rao, and Y. Ma, *Robust principal component analysis: Exact recovery of corrupted low-rank matrices by convex optimization*, in Advances in Neural Information Processing Systems (NIPS), Curran, Red Hook, NY, 2009.

[49] X. Yi, D. Park, Y. Chen, and C. Caramanis, *Fast algorithms for robust PCA via gradient descent*, in Advances in Neural Network Information Processing Systems (NIPS), Curran, Red Hook, NY, 2016, pp. 4159–4167.

[50] X. Zeng and M. A. T. Figueiredo, *The Ordered Weighted ℓ_1 Norm: Atomic Formulation, Dual Norm, and Projections*, arXiv, 2014.

[51] T. Zhou and D. Tao, *Godec: Randomized low-rank & sparse matrix decomposition in noisy case*, in International Conference on Machine Learning (ICML), International Machine Learning Society, Madison, WI, 2011.

[52] Z. Zhou, X. Li, J. Wright, E. J. Candès, and Y. Ma, *Stable principal component pursuit*, in IEEE International Symposium on Information Theory (ISIT), Curran, Red Hook, NY, 2010.

[53] H. Zou and T. Hastie, *Regularization and variable selection via the elastic net*, J. Roy. Statist. Soc. Ser. B, 67 (2005), pp. 301–320.