1 Overview

This section discusses some of the algorithms that have recently been proposed in the literature to solve the Robust PCA problem

$$\begin{aligned} p^* = \min_{L,S} \ & \|L\|_* + \lambda \|S\|_1 \\ \text{s.t.} \quad & M = L + S \end{aligned} \tag{1}$$

There are various methods that can be used to solve (1). A straightforward way described in section 2.1 is to use general purpose interior point solvers [19, 21] to solve an SDP formulation of the dual of (1), an approach that works well for low-dimensional data M but unfortunately does not scale well. Another approach is to use iterative thresholding techniques as described in section 2.2, which result in a very simple algorithm that can be applied also to high-dimensional data. Unfortunately the convergence of this algorithm is extremely slow. More sophisticated approaches include an accelerated proximal gradient algorithm (section 2.3) and a gradient ascent algorithm applied to the dual problem of (1) (section 2.4). The current state of the art seems to be a adaptation of the Augmented Lagrangian Method to the non-smooth problem (1), an approach that is discussed in section 2.5.

2 Main algorithms for Robust PCA

2.1 Interior Point Methods

As will be shown in the following section, the dual problem of (1) can be cast as a Semidefinite Programming (SDP) problem for which a number of efficient polynomial-time interior-point solvers have been developed. In principle, one can then just apply these general purpose off-the-shelf solvers to the dual SDP. This approach will work well for small and medium sized problems, but unfortunately it does not scale with the size of the data matrix M. The following section first develops the dual of the Robust PCA problem, section 2.1.2 then discusses the limitations of using interior point methods on the resulting SDP.

2.1.1 Formulation of the Dual problem of Robust PCA as an SDP

Let us use the equality constraint to eliminate the variable L from (1). Using the facts that $\|X\|_* = \max_{\|Y\|_2 \leq 1} \mathbf{Tr}\, Y^T X$ and $\|X\|_1 = \max_{\|Z\|_\infty \leq 1} \mathbf{Tr}\, Z^T X$ the problem can be written as

$$\begin{aligned} p^* = \min_{S} \max_{Y,Z} \ & \mathbf{Tr}\, Y^T (M - S) + \lambda\, \mathbf{Tr}\, Z^T S \\ \text{s.t.} \quad & \|Y\|_2 \leq 1 \\ & \|Z\|_\infty \leq 1 \end{aligned} \tag{2}$$

The dual function is

$$\begin{aligned} g(Y,Z) &= \min_{S} \ \mathbf{Tr}\, Y^T M + \mathbf{Tr}(\lambda Z - Y)^T S \\ &= \begin{cases} \mathbf{Tr}\, Y^T M & \text{if } \lambda Z = Y \\ -\infty & \text{otherwise} \end{cases} \end{aligned} \tag{3}$$

We obtain the dual problem

$$\begin{aligned} p^* \geq d^* = \max_{Y,Z} \quad & \mathbf{Tr}\, Y^T M \\ \text{s.t.} \quad & \lambda Z = Y \\ & \|Y\|_2 \leq 1 \\ & \|Z\|_\infty \leq 1 \end{aligned} \tag{4}$$

which after eliminating the variable Z and using homogeneity of the norm becomes

$$\begin{aligned} d^* = \max_{Y} \quad & \mathbf{Tr}\, Y^T M \\ \text{s.t.} \quad & \|Y\|_2 \leq 1 \\ & \|Y\|_\infty \leq \lambda \end{aligned} \tag{5}$$

Noting that $\|Y\|_2 \leq 1 \iff I - X^T(I)^{-1}X \succeq 0$ and using a Schur Complement lemma, the dual problem can be transformed into the following SDP standard form:

$$\begin{aligned} d^* = \max_{Y} \quad & \mathbf{Tr}\, M^T Y \\ \text{s.t.} \quad & \begin{bmatrix} I & Y^T \\ Y & I \end{bmatrix} \succeq 0 \\ & \mathbf{Tr}\, \Delta_{ij}^T Y \leq \lambda, \quad i = 1, \ldots, m,\ j = 1, \ldots, n \\ & \mathbf{Tr}\, \Delta_{ij}^T Y \geq -\lambda, \quad i = 1, \ldots, m,\ j = 1, \ldots, n \end{aligned} \tag{6}$$

Here $\Delta_{ij} \in \mathbf{R}^{m \times n}$ is such that $\Delta_{ij}(k,l) = \delta_{ik}\delta_{jl}$. Note that both primal and dual problem are (trivially) strictly feasible. In fact, the primal is unconstrained and an obvious strictly feasible dual variable is $Y = 0$. Hence strong duality holds ($p^* = d^*$) and both primal and dual problem are attained.

Recovering the primal solution

To recover the primal solution we can distinguish the two cases $\lambda \leq 1$ and $\lambda > 1$. For the latter one, first note that $\|Y\|_\infty \leq \|Y\|_2$ for all Y. To show this first recall that $c := \|Y\|_2 = \max_{\|x\|_2=1} \|Yx\|_2$. This means that $c - x^T Y^T Y x \geq 0$, $\forall\, x$ or, equivalently, $cI - Y^T Y \succeq 0$. Using a Schur Complement this can be written as

$$\begin{bmatrix} I & Y \\ Y^T & cI \end{bmatrix} \succeq 0$$

It is easy to see that this implies that

$$\begin{bmatrix} 1 & Y_{ij} \\ Y_{ij} & c \end{bmatrix} \succeq 0$$

for all i, j, which is equivalent to $|Y_{ij}| \leq c$, $\forall\, i, j$. This shows that $\|Y\|_\infty \leq \|Y\|_2$.

Hence we observe that if $\lambda > 1$ the constraint $\|Y\|_2 \leq \lambda$ will be inactive, that is $|Y_{ij}| < \lambda$, $\forall\, i, j$. Therefore, for $\lambda > 1$, removing this constraint from (5) does not change the problem. We have $d^* = \max_{\|Y\|_2 \leq 1} \mathbf{Tr}\, Y^T M = \|M\|_*$, and this lower bound is achieved by the primal (feasible) solution $L = M$ and $S = 0$.

If $\lambda \leq 1$, then ....

2.1.2 Using Interior Point Methods to Solve the Dual

Problem (5) is an SDP and can therefore in principle be solved using off-the-shelf solver packages such as `sedum` [19] and `sdpt3` [21]. These solvers are based on interior point methods which offer superior convergence rates [6] but unfortunately do not scale ell at all with the size of the matrix M. Figure 1 shows the average running time over 10 instances of the dual problem for the solvers `sedumi` and `sdpt3` for randomly generated data of very low dimensions. There is a surprisingly big

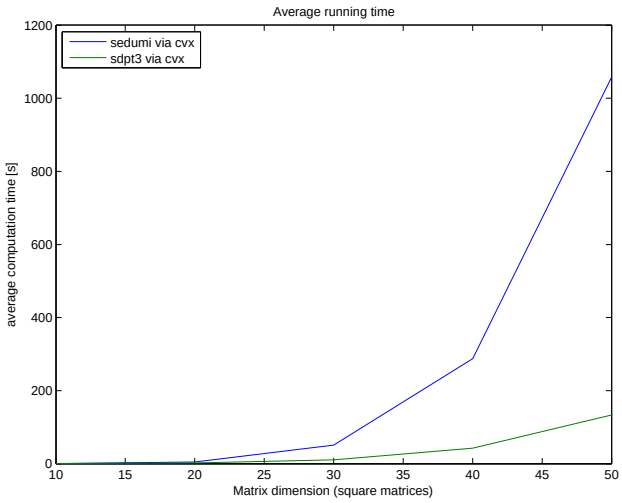

Figure 1: Average running time of interior point methods for the dual problem

difference in terms of the running times: `sdpt3` is consistently faster than `sedumi`, with the spread growing significantly with the problem size. For the largest dimension `sdpt3` on average is 8 times faster than `sedumi` for this particular problem. To be able to give a realistic assessment of why this is the case would however require detailed knowledge of the solver implementation.

In any case it is obvious that, with an average running time of the faster of the two solvers of more than 2 minutes even on toy-sized problems with only 2500 matrix entries, interior point solvers are unsuitable for most problem of interest. The reason for this extremely bad scaling with the matrix dimension is that the computation of the Newton step direction relies on second-order information of the (barrier-augmented) objective function, the computation of which is infeasible when the number of variables is very high. For applications that involve large-scale data matrices (with mn in the millions or even billions), using interior point methods therefore is essentially impossible. In order to overcome this scalability issue, a variety of first-order methods exploiting the particular structure of the Robust PCA problem have been proposed. Some of these methods will be described in the following.

2.2 Iterative Thresholding Method

Among the first techniques used to solve (1) for high-dimensional data was an adaptation [22][1] of an iterative thresholding method originally proposed for the matrix completion problem in [7]. For

[1] note that the iterative thresholding algorithm seems to never have appeared in the published version of [22], which instead contains the accelerated proximal gradient method described in section 2.3

this method the authors of consider a relaxation of the original problem (1) of the form

$$\begin{aligned} &\min_{L,S} \ \|L\|_* + \lambda\|S\|_1 + \frac{1}{2\tau}\|L\|_F^2 + \frac{1}{2\tau}\|S\|_F^2 \\ &\text{s.t.} \quad M = L + S \end{aligned} \tag{7}$$

where $\tau \gg 1$ is a scalar parameter. Generalizing the result from [7] (which considers the matrix completion problem, i.e. $S = 0$), the authors of [22] argue that for large values of τ the solution of (7) will be very close to that of (1) (but interestingly enough they do not prove it!). One can now use iterative thresholding techniques to solve (7). Although the resulting algorithm has little relevance in practice because of its extremely slow convergence, we will see in the following sections that its main ideas are the basis for a number of other similar algorithms. Therefore we briefly review the iterative thresholding algorithm here.

The Lagrangian of the problem (7) is given by

$$\mathcal{L}(L, S, Y) = \|L\|_* + \lambda\|S\|_1 + \frac{1}{2\tau}\|L\|_F^2 + \frac{1}{2\tau}\|S\|_F^2 + \frac{1}{\tau}\langle Y, M - L - S\rangle \tag{8}$$

The idea of the iterative thresholding algorithm is, as the name suggests, to update the variables L, S and Y iteratively. More specifically, the Lagrangian $\mathcal{L}(L, S, Y)$ is minimized w.r.t L and S for some fixed dual variable Y, and the violation of the constraint is then used to update Y using the gradient step $Y^+ = Y + t(M - L - S)$, where $0 < t < 1$ is the step size.

Note that for fixed Y we have

$$\begin{aligned} (\hat{L}, \hat{S}) :&= \arg\min_{L,S} \ \mathcal{L}(L, S, Y) \\ &= \arg\min_{L,S} \ \|L\|_* + \lambda\|S\|_1 + \frac{1}{2\tau}\|L\|_F^2 + \frac{1}{2\tau}\|S\|_F^2 + \frac{1}{\tau}\langle Y, M - L - S\rangle \\ &= \arg\min_{L,S} \ \|L\|_* + \lambda\|S\|_1 + \frac{1}{2\tau}\|L - Y\|_F^2 + \frac{1}{2\tau}\|S - Y\|_F^2 \end{aligned} \tag{9}$$

We note that the above problem is completely separable hence the minimizers $\hat{L}$ and $\hat{S}$ can be determined independently. Using optimality conditions based on the subgradient it can be shown [7, 22] that the minimizers $\hat{L}$ and $\hat{S}$ are both of a simple form. To this end, consider the following extension to the matrix case of the well known soft-thresholding operator (e.g. from the proximal mapping of the l_1-norm in the vector case):

$$(\mathcal{S}_\varepsilon[X])_{ij} := \begin{cases} x_{ij} - \varepsilon & \text{if } \ x_{ij} > \varepsilon \\ x_{ij} + \varepsilon & \text{if } \ x_{ij} < -\varepsilon \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

Theorem 1 (Singular Value Thresholding [7])**.** *For each $\tau \geq 0$ and $Y \in \mathbf{R}^{m\times n}$, we have that*

$$\mathcal{D}_\tau(Y) := U\mathcal{S}_\tau[\Sigma]V^T = \arg\min_X \ \|X\|_* + \frac{1}{2\tau}\|X - Y\|_F^2 \tag{11}$$

where $Y = U\Sigma V^*$ is the SVD of Y.

Since Theorem 1 is central to all algorithms discussed in this chapter, we give its proof here.

Proof of Theorem 1. Since the objective function $f_0(X) := \arg\min_X \ \|X\|_* + \frac{1}{2\tau}\|X - Y\|_F^2$ is strictly convex its minimizer is unique. Recall that Z is a subgradient of a convex function $f : \mathbf{R}^{m\times n} \mapsto \mathbf{R}$ at X_0 if $f(X) \geq f(X_0) + \langle Z, X - X_0\rangle$ for all X. The set of all sugradients Z at X_0 is called the subdifferential and denoted $\partial f(X_0)$. By definition it follows that $\hat{X}$ minimizes f_0 if and only if 0 is as subgradient of f_0 at $\hat{X}$. Using standard subgradient calculus this translates to $0 \in \hat{X} - Y + \tau\partial||\hat{X}||_*$, where $\partial||\hat{X}||_*$ is the subdifferential of the nuclear norm at $\hat{X}$. To show that $\mathcal{D}_\tau(Y)$ satisfies (11) we therefore need to show that $0 \in \mathcal{D}_\tau(Y) - Y + \tau\partial||\mathcal{D}_\tau(Y)||_*$.

Let $X \in \mathbf{R}^{m\times n}$ be arbitrary Let $U\Sigma V^*$ be its SVD. It can be shown that

$$\partial||X||_* = \left\{UV^* + W \ \mid \ W \in \mathbf{R}^{m\times n},\ U^*W = 0,\ WV = 0,\ \|W\|_2 \leq 1\right\}$$

To use the above result decompose the SVD of Y as $Y = U_0\Sigma_0 V_0^* + U_1\Sigma_1 V_1^*$, where U_0, V_0 (U_1, V_1) are the singular vectors associated with singular values greater (less or equal) than τ. With this notation we have $\mathcal{D}_\tau(Y) = U_0(\Sigma_0 - \tau I)V_0^*$, so $Y - \mathcal{D}_\tau(Y) = \tau(U_0V_0^* + W)$ with $W = \tau^{-1}U_1\Sigma_1V_1^*$. By definition of the SVD $U_0^*W = 0$ and $WV_0 = 0$. Furthermore, since $\max(\Sigma_1) \leq \tau$ it also holds that $\|W\|_2 \leq 1$. Together this shows that $Y - \mathcal{D}_\tau(Y) \in \tau\partial||\mathcal{D}_\tau(Y)||_*$. □

It is also not hard to find the minimizer $\hat{S}$ of (8), as the following proposition shows:
Proposition 1 (Matrix Value Thresholding)**.** *For each $\tau, \lambda \geq 0$ and $Y \in \mathbf{R}^{m\times n}$, we have that*

$$\mathcal{S}_{\tau\lambda}[Y] = \arg\min_X \ \lambda\|X\|_1 + \frac{1}{2\tau}\|X - Y\|_F^2 \tag{12}$$

Proof of Proposition 1. Note that the objective function $h_0(X) := \lambda\|X\|_1 + \frac{1}{2\tau}\|X-Y\|_F^2$ in (12) is completely decomposable and can be written in the form $h_0(X) = \sum_{i,j}\lambda|X_{ij}| + \frac{1}{2\tau}(X_{ij}-Y_{ij})^2$, so the minimization over X can be carried out element-wise. Using the same reasoning as in the proof of Theorem 1, we have that the element X_{ij} is a minimizer if and only if $0 \in \partial h_0(X)$. This condition can be written as $Y_{ij} - X_{ij} \in \tau\lambda\,\partial|X_{ij}|$. We consider the cases $X_{ij} = 0$ and $X_{ij} \neq 0$. If $X_{ij} = 0$ then the condition reads $Y_{ij} \in \tau\gamma[-1, 1]$. In other words, when $|Y_{ij}| \leq \tau\gamma$ then $X_{ij} = 0$. On the other hand, if $X_{ij} \neq 0$, then $\partial\,|X_{ij}| = \mathrm{sign}(X_{ij})$ and we have $X_{ij} = Y_{ij} - \tau\lambda\,\mathrm{sign}(Y_{ij})$, which is in fact the soft-thresholding operator $\mathcal{S}_{\tau\lambda}[X_{ij}]$. □

Applying Theorem 1 and Proposition 1 to (9) we find that the minimizers $\hat{L}$ and $\hat{S}$ of (8) are given, respectively, by

$$\hat{L} = U\mathcal{S}_\tau[\Sigma]V^T \tag{13}$$

$$\hat{S} = \mathcal{S}_{\tau\lambda}[Y] \tag{14}$$

The overall iterative thresholding algorithm for Robust PCA is given in Algorithm 1.

Algorithm 1 is extremely simple to implement, as each iteration only requires the computation of an SVD of the current dual variable Y_k and a few elementary matrix operations. Note that, in fact, since the shrinking operator sets all singular values less than the threshold parameter to zero one really only needs to compute the singular values that lie above this threshold. In section 3.2.2 we will discuss how this can be exploited to achieve potential speedups in all of the algorithms considered in this chapter.

Algorithm 1: Iterative Thresholding Algorithm

Input: Observation matrix M, parameters λ, τ
initialization: $k = 0$, $Y_0 = 0$;
while *not converged* **do**
$k = k + 1$;
$(U, \Sigma, V) = \text{svd}(Y_{k-1})$;
$L_k = U\mathcal{S}_\tau[\Sigma]V^T$;
$S_k = \mathcal{S}_{\lambda\tau}[Y_{k-1}]$;
$Y_k = Y_{k-1} + t_k(M - L_k - S_k)$;
Output: $L = L_k$, $S = S_k$

While this iterative thresholding scheme is very simple and has been proved to converge, its convergence unfortunately is extremely slow. The authors of [22] found that it typically requires about 10^4 iterations to converge for problems of reasonable size. Furthermore it is hard to find good general schemes to optimize the choice of the step size t_k [12]. As it turns out the other algorithms discussed in the following sections have a comparable complexity but converge much faster, both in theory and practice. Therefore the practical applicability of this approach is limited.

2.3 Accelerated Proximal Gradient Method

An accelerated proximal gradient (APG) method for solving (1) was proposed in [13]. The method is essentially an application of the FISTA algorithm [3] to a relaxation of the original RPCA problem in combination with a continuation technique. This FISTA algorithm algorithm is reminiscent of Nesterov's "optimal" first-oder method for smooth objective functions [16], whose $O(1/k^2)$ convergence rate result was extended to non-smooth objectives in [17]. Besides being theoretically appealing, the APG method is also in practice much faster than the iterative thresholding algorithm discussed in section 2.2.

The following sections describe the APG algorithm. As in [13] we first give a general formulation and then show how it can be applied to the Robust PCA problem, using ideas from section 2.2.

2.3.1 A General Form of the Accelerated Proximal Gradient Method

Proximal gradient algorithms in general can be used to solve unconstrained problems of the form

$$\min_x \; f(x) := g(x) + h(x) \tag{15}$$

where g is convex and differentiable and h is closed[2], convex but need not be differentiable. The proximal mapping of a convex function h is defined as

$$\text{prox}_h(x) := \arg\min_u \left(h(u) + \frac{1}{2}\|u - x\|^2 \right) \tag{16}$$

[2]that is, its epigraph is closed

Here $\|\cdot\|$ denotes the inner product norm[3]. Note that since h is convex and the norm is strictly convex, the optimizer in (16) is unique. The step of each iteration of the classic (non-accelerated) proximal gradient algorithm is

$$x^+ = \text{prox}_{\delta h}(x - \delta \nabla g(x)) \tag{17}$$

where x^+ denotes the next iterate, based on the current iterate x, and δ is the step size (which can be fixed or determined via backtracking line search). Since the proximal mapping (17) needs to be computed at each iteration (possibly multiple times because of the line search), the algorithm is most effective when this can be done cheaply. Depending on the function h, the proximal gradient algorithm can be seen as a generalization of different gradient-based algorithms. In particular, for $h(x) \equiv 0$ the standard gradient algorithm is recovered and for $h(x) = I_C(x)$, where I_C denotes the indicator function for the convex set C, the projected gradient algorithm is recovered.

Note that (17) can be written as

$$\begin{aligned} x^+ &= \arg\min_u \left(h(u) + \frac{1}{2\delta}\|u - x + \delta \nabla g(x)\|^2 \right) = \arg\min_u \left(h(u) + \frac{1}{2\delta}\|u - G(x)\|^2 \right) \\ &= \arg\min_u \left(h(u) + g(x) + \langle \nabla g(x), u - x \rangle + \frac{1}{2\delta}\|u - x\|^2 \right) \end{aligned} \tag{18}$$

where $G(x) := x - \delta\, \nabla g(x)$. Therefore, each step (17) can be interpreted as minimizing the function $h(u)$ plus a quadratic local model of $g(u)$ around the current iterate x. This classic form of the proximal gradient algorithm is well known in the literature and, under Lipschitz continuity of the gradient of g and some technical conditions on the step size t, has been shown to have a convergence rate no worse than $O(1/k)$ (PUT REFERENCE). In particular, a popular choice for δ is the fixed step size $\delta = 1/L_g$, where L_g is a Lipschitz constant of the gradient of g, that is we have $\|\nabla g(x) - \nabla g(y)\| \leq L_g \|x - y\|$ for all x, y. Note that in practice for a general problem the value of L_g might be unknown. This is not a big concern for the type of problem arising in Robust PCA, hence we will assume this choice of $\delta = 1/L_g$ in the following.

It turns out that the choice for the point around which the quadratic local model of $g(u)$ is constructed has an important effect on the convergence rate, and that less obvious choices than just the previous iterate x are actually better (in the sense that they yield higher convergence rates). Specifically, consider a generalized version of (18), where the approximation of g is constructed around some y that may depend on all prior iterates $x_0, \ldots, x_k$:

$$x^+ = \arg\min_u \left(h(u) + \frac{L_g}{2}\|u - G(y)\|^2 \right) \tag{19}$$

Nesterov in [16] showed that in the smooth case the standard gradient algorithm (i.e. $h(x) \equiv 0$) can be accelerated by choosing $y_k = x_k + \frac{t_{k-1}-1}{t_k}(x_k - x_{k-1})$, leading to a theoretical convergence rate of $O(1/k^2)$. This algorithm is optimal in the sense that its convergence rate achieves (in terms of order) the theoretical complexity bound on all first order algorithms. The seminal work [16] has been extended to the non-smooth case in [3, 17], yielding the generic accelerated proximal gradient method given in Algorithm 2.

[3]Note that the derivation holds for real inner product spaces, not just $\mathbf{R}^n$. We are particularly interested in the case $\mathbf{R}^{m \times n}$

Algorithm 2: Accelerated Proximal Gradient Algorithm

initialization: $k = 0$, $t_0 = t_{-1} = 1$;
while *not converged* **do**
$y_k = x_k + \frac{t_{k-1}-1}{t_k}(x_k - x_{k-1})$;
$G_k = y_k - \frac{1}{L_g}\nabla g(y_k)$;
$x_{k+1} = \arg\min_x \left(h(x) + \frac{L_g}{2}\|x - G(y_k)\|^2\right)$;
$t_{k+1} = \frac{1+\sqrt{4t_k^2+1}}{2}$;
$k = k + 1$;

2.3.2 Accelerated Proximal Gradient Algorithm Applied to Robust PCA

Using some the ideas from the iterative thresholding method in section 2.2, it is possible to construct an accelerated proximal gradient algorithm for a relaxation of the original problem (1). In particular, it is easy to see that in the common case $h(x) = \|x\|_1$ the update for x in Algorithm 2 is simply given by $x_{k+1} = \mathcal{S}_{1/L_g}[G(y_k)]$, where $\mathcal{S}_\varepsilon$ is the soft-thresholding operator defined in (10). Similar ideas can be used to develop an APG algorithm for the Robust PCA problem.

The authors of [13] consider a relaxation of (1) of the form

$$\min_{L,S} \ \mu\|L\|_* + \mu\lambda\|S\|_1 + \frac{1}{2}\|L + S - M\|_F^2 \tag{20}$$

It can be shown that for $\mu \to 0$, any solution of (20) approaches the solution set of (1). In practice, rather than fixing μ to some small value, one can achieve superior convergence of the algorithm by using a continuation technique on μ, that is, by solving (20) by repeatedly decreasing the value of μ in the steps of the accelerated proximal gradient algorithm. This can be interpreted as a particular homotopy method.

With $X = (L, S)$ we identify $h(X) = \mu\|L\|_* + \mu\lambda\|S\|_1$ and $g(X) = \frac{1}{2}\|L + S - M\|_F^2$ in (20). The gradient of g is $\nabla g(X) = (\nabla_L g(X), \nabla_S g(X))$, where $\nabla_L g(X) = \nabla_S g(X) = L + S - M$. Thus

$$\begin{aligned}\|\nabla g(X_1) - \nabla g(X_2)\| &= \left\|\begin{bmatrix}L_1 + S_1 - M\\ L_1 + S_1 - M\end{bmatrix} - \begin{bmatrix}L_2 + S_2 - M\\ L_2 + S_2 - M\end{bmatrix}\right\| = \left\|\begin{bmatrix}L_1 - L_2 + S_1 - S_2\\ L_1 - L_2 + S_1 - S_2\end{bmatrix}\right\| \\ &\le \left\|\begin{bmatrix}L_1 - L_2\\ S_1 - S_2\end{bmatrix}\right\| + \left\|\begin{bmatrix}S_1 - S_2\\ L_1 - L_2\end{bmatrix}\right\| = 2\,\|X_1 - X_2\|\end{aligned}$$

which means that a Lipschitz constant is given by $L_g = 2$. It turns out that because of the separability of both the function h and the Frobenius norm the update can be be decomposed as follows:

$$\begin{aligned}(L_{k+1}, S_{k+1}) = X_{k+1} &= \arg\min_X \left(h(X) + \frac{L_g}{2}\|X - G(Y_k)\|^2\right) \\ &= \arg\min_{L,S} \left(\mu\|L\|_* + \mu\lambda\|S\|_1 + \left\|\begin{bmatrix}L\\ S\end{bmatrix} - \begin{bmatrix}G_k^L\\ G_k^S\end{bmatrix}\right\|^2\right) \\ &= \arg\min_{L,S} \left(\|L\|_* + \frac{1}{2\frac{\mu}{2}}\left\|L - G_k^L\right\|^2 + \lambda\|S\|_1 + \frac{1}{2\frac{\mu}{2}}\left\|S - G_k^S\right\|^2\right)\end{aligned}$$

The problem is therefore completely separable and clearly the sum of two problems of the form (15). In particular, if $G_k^L = U\Sigma V^T$ is the SVD of G_k^L then, using the results from section 2.2, the iterates L_{k+1} and S_{k+1} are given by

$$L_{k+1} = U\mathcal{S}_{\frac{\mu}{2}}[\Sigma]V^T \tag{21}$$

$$S_{k+1} = \mathcal{S}_{\frac{\lambda\mu}{2}}[G_k^S] \tag{22}$$

Therefore it is rather straightforward to formulate a version Algorithm 2 for the Robust PCA problem. One extension to Algorithm 2 is the use of a continuation technique (or homotopy method) for the parameter μ. In [13], the authors propose to start from some large value μ_0 and then choose $\mu_{k+1} = \max(\eta\mu_k, \bar{\mu})$, where $0 < \eta < 1$ and $\bar{\mu}$ is a lower limit on μ_k. This continuation technique has been observed to improve convergence. The Accelerated Proximal Gradient Algorithm applied to the Robust PCA problem is given in Algorithm 3.

Algorithm 3: Accelerated Proximal Gradient Algorithm for Robust PCA

Input: Observation matrix M, parameter λ
initialization: $k = 0$, $L_0 = L_{-1} = 0$, $S_0 = S_{-1} = 0$, $t_0 = t_{-1} = 1$, $\bar{\mu} = \delta\mu_0$;
while *not converged* **do**
$Y_k^L = L_k + \frac{t_{k-1}-1}{t_k}(L_k - L_{k-1})$, $Y_k^S = S_k + \frac{t_{k-1}-1}{t_k}(S_k - S_{k-1})$;
$G_k^L = Y_k^L - \frac{1}{2}(Y_k^L + Y_k^S - M)$;
$(U, \Sigma, V) = \text{svd}(G_k^L)$, $L_{k+1} = U\mathcal{S}_{\mu_k/2}[\Sigma]V^T$;
$G_k^S = Y_k^S - \frac{1}{2}(Y_k^L + Y_k^S - M)$;
$S_{k+1} = \mathcal{S}_{\lambda\mu_k/2}[G_k^S]$;
$t_{k+1} = \frac{1+\sqrt{4t_k^2+1}}{2}$;
$\mu_{k+1} = \max(\eta\mu_k, \bar{\mu})$;
$k = k + 1$;
Output: $L = L_k$, $S = S_k$

We will restate the main convergence theorem for Algorithm 3 without proof.

Theorem 2 ([12])**.** *Let* $F(X) = F(L, S) := \bar{\mu}\|L\|_* + \bar{\mu}\lambda\|S\|_1 + \frac{1}{2}\|L + S - M\|_F^2$. *Then for all* $k > k_0 := -C_1 \log(\eta)$ *it holds that*

$$F(X_k) - F(X^*) \leq \frac{4\|C_{k_0} - X^*\|_F^2}{(k - k_0 + 1)^2}$$

where $C_1 = \log(\mu_0/\bar{\mu})$ *and* X^* *is any solution to* (20).

Remark 2.1 (Accuracy of the APG method). Note that for any finite $\bar{\mu}$ Algorithm 3 provides only an approximate solution of the Robust PCA problem due to the replacement of the equality constraint with the penalty term $\frac{1}{2\bar{\mu}}\|L + S - M\|_F^2$. In practice one therefore has to tradeoff accuracy ($\bar{\mu}$ extremely small) and computational efficiency.

2.4 Gradient Ascent on the Dual

In section 2.1.2 we discussed why solving the Robust PCA dual problem (5) via interior point techniques quickly becomes infeasible with growing size of the data matrix M. Another way of

solving the dual given in [13] is based on a steepest ascent algorithm. Note that the dual problem (5) derived in section 2.1.1 can be written as

$$d^* = \max_Y \ \mathbf{Tr}\, M^T Y \quad \text{subject to} \quad J(Y) \leq 1 \tag{23}$$

where $J(Y) := \max\left(\|Y\|_2, \lambda^{-1}\|Y\|_\infty\right)$. Denote by $F := \{Y \mid J(Y) \leq 1\}$ the feasible set. The maximum operator in the function J implies that $F = \{Y \mid \|Y\|_2 \leq 1\} \cap \{Y \mid \lambda^{-1}\|Y\|_\infty \leq 1\}$, hence F is clearly convex, being the intersection of two convex sets. Further, the function $J(Y)$ is positive and homogenous in Y, hence the maximum over a linear of the objective function is achieved on the boundary ∂F of the feasible set, that is, the optimum is achieved when $J(Y) = 1$.

Top solve (23), the authors in [13] propose to use a projected gradient algorithm. The gradient of the objective function $\mathbf{Tr}\, M^T Y$ is simply M. The projection onto the constraint set is more involved but can be treated using standard methods from convex optimization. In particular, the algorithm at each iteration k involves projecting the gradient M onto the tangent cone $T_F(Y_k)$ of the feasible set F at the point Y_k. If W_k is the steepest ascent direction obtained from this projection, the iterate can be updated using the (projected) gradient step

$$Y_{k+1} = \frac{Y_k + t_k W_k}{J(Y_k + t_k W_k)} \tag{24}$$

where t_k is the step size that can be determined by a simple line search of the form

$$t_k = \arg\max_{t \geq 0} \left\langle M, \frac{Y_k + t W_k}{J(Y_k + t W_k)} \right\rangle \tag{25}$$

Note that the division by $J(Y_k + t_k W_k)$ ensures that the next iterate Y_{k+1} lies on ∂F. It is shown in [13] that if the maximizing step size t_k is equal to zero at some point Y_k, then Y_k is the optimal solution to the dual problem (23).

In order to perform the projection on the tangent cone T_F of F at the point Y_k, the authors of [13] make use the following two facts:

1. For any convex subset $K \subseteq V$ of a real vector space V the normal cone $N_K(x)$ at the point $x \in K$ is the polar cone to the tangent cone $T_K(x)$.
2. If two cones C_1 and C_2 in V are are polar cones to each other and $\mathcal{P}_{C_1}$ and $\mathcal{P}_{C_2}$ are the projection operators onto C_1 and C_2, respectively, then $\mathcal{P}_{C_1}(X) + \mathcal{P}_{C_1}(X) = X$ for any point $X \in V$.

From these facts, it follows immediately that $\mathcal{P}_{T_F(Y_k)}(M)$, the projection of M onto the dual cone of F at Y_k, can be computed as $\mathcal{P}_{T_F(Y_k)}(M) = M - M_k$, where $M_k := \mathcal{P}_{N_F(Y_k)}(M)$. It can be shown that the normal cone is characterized by the subgradient of the function J via $N(Y_k) = \{\alpha X \mid \alpha \geq 0, X \in \partial J(Y_k)\}$. Note that J is in fact the pointwise maximum over two functions, hence from strong subgradient calculus it follows that

$$\partial J(Y_k) = \begin{cases} \partial \|Y_k\|_2 & \text{if} \quad \|Y_k\|_2 > \lambda^{-1}\|Y_k\|_\infty \\ \lambda^{-1} \partial \|Y_k\|_\infty & \text{if} \quad \|Y_k\|_2 < \lambda^{-1}\|Y_k\|_\infty \\ \mathrm{Conv}\left\{\partial \|Y_k\|_2 \,,\, \lambda^{-1}\partial \|Y_k\|_\infty\right\} & \text{if} \quad \|Y_k\|_2 = \lambda^{-1}\|Y_k\|_\infty \end{cases} \tag{26}$$

where $\|\cdot\|_\infty$ is the maximum absolute value of the matrix entries (not the induced ∞-norm).

The first two cases are rather simple and the associated projections $\mathcal{P}_2(\cdot)$ and $\mathcal{P}_\infty(\cdot)$ onto the normal cones generated by the sub gradients of $\|\cdot\|_2$ and $\|\cdot\|_\infty$ at Y_k, respectively, are efficiently computable [13]. In particular, at each step the projection $\mathcal{P}_2(\cdot)$ requires the computation of the largest singular value[4] (and associated singular vectors) of the matrix Y_k, and the projection $\mathcal{P}_\infty(\cdot)$ requires only a simple element-wise comparison of the matrices M and Y_k. The projection in the third case, i.e. when $N(Y_k) = N_2(Y_k) + N_\infty(Y_k)$, is more involved and can be performed using an alternating projections algorithm. It can be shown [13] that by initializing $S_0 = 0$ and $j = 0$, and then repeatedly setting $L_{j+1} = \mathcal{P}_2(M - S_j)$, $S_{j+1} = \mathcal{P}_\infty(M - L_{j+1})$ will yield the projection $M_k = \mathcal{P}_{N(Y_k)}(M)$ in the sense that $\lim_{j\to\infty} L_j + S_j = M_k$. The projected gradient ascent algorithm for the dual problem is given in Algortihm (4).

Algorithm 4: Projected Gradient Ascent for the Dual Problem

Input: Observation matrix M, parameter λ
initialization: $k = 0$, $Y_0 = \text{sign}(M)/J(M)$;
while *not converged* **do**
  Compute the projection M_k of M onto $N_F(Y_k)$:
  if $\|Y_k\|_2 > \lambda^{-1}\|Y_k\|_\infty$ **then**
    $M_k = \mathcal{P}_2(M)$, $L = M$, $S = 0$
  else if $\|Y_k\|_2 < \lambda^{-1}\|Y_k\|_\infty$ **then**
    $M_k = \mathcal{P}_\infty(M)$, $L = 0$, $S = M$
  else
    $L = 0$, $S = 0$;
    while *not converged* **do**
      $L = \mathcal{P}_2(M - S)$, $S = \mathcal{P}_\infty(M - L)$
    $M_k = L + S$
  Perform a line search as in (25) to determine step size t_k;
  $Y_{k+1} = \frac{Y_k + t_k(M - M_k)}{J(Y_k + t_k(M - M_k))}$;
  $k = k + 1$;
Output: L, S

Using the properties of dual norms, it is possible to show [13] that the primal solution $(\hat{L}, \hat{S})$ can easily be recovered from the dual optimal $\hat{Y}$. Specifically, it can be shown that if $\|Y_k\|_2 < 1$ or $\lambda^{-1}\|Y_k\|_\infty < 1$ the solution is degenerate and given by $\hat{L} = 0, \hat{S} = M$ (in case $\|Y_k\|_2 < 1$) or $\hat{L} = M, \hat{S} = 0$ (in case $\lambda^{-1}\|Y_k\|_\infty < 1$). If $\|Y_k\|_2 = \lambda^{-1}\|Y_k\|_\infty = 1$, then $(\hat{L}, \hat{S})$ is any pair of points that achieves convergence of the alternating projections method described above.

The algorithm at heart is a simple projected gradient ascent algorithm, the convergence rate of which is known to be $O(1/k)$ (in terms of outer iterations, not counting the alternating projection sub-algorithm). The most costly operations are the projections $\mathcal{P}_2(\cdot)$ that require the computation of the largest singular value of matrices of the same size as M.

[4] Note that the largest singular value of Y_k needs to be computed anyway at each step in order to distinguish the three cases in (26)

2.5 Augmented Lagrangian Method

2.5.1 The General Case

The classic Augmented Lagrangian Method (ALM), described in [5], is a method for solving equality-constrained convex optimization problems. Consider a generic equality-constrained optimization problem of the form

$$p^* = \min_{x \in \mathcal{H}} f_o(x) \quad \text{s.t.} \quad f_c(x) = 0 \tag{27}$$

where $\mathcal{H}$ and $\mathcal{H}'$ are Hilbert spaces and $f_o : \mathcal{H} \mapsto \mathbf{R}$ and $f_c : \mathcal{H} \mapsto \mathcal{H}'$ are both convex functions. The conventional Lagrangian of the problem is $\mathcal{L}(x, \lambda) = f_o(x) + \langle \lambda, f_c(x) \rangle$, with $\lambda \in \mathcal{H}'$ being the dual variable. The augmented Lagrangian method, as its name suggests, uses an augmented Lagrangian of the form

$$\mathcal{L}(x, \lambda, \mu) = f_o(x) + \langle \lambda, f_c(x) \rangle + \frac{\mu}{2} \|f_c(x)\|^2 \tag{28}$$

where $\|\cdot\|$ is the inner product norm. Here λ can now be interpreted as an estimate of a dual variable. The general algorithm is quite simple and given in Algorithm 5.

Algorithm 5: Generic Augmented Lagrangian Method

while *not converged* **do**
  $x_{k+1} = \arg\min_x \mathcal{L}(x, \lambda_k, \mu_k)$;
  $\lambda_{k+1} = \lambda_k + \mu_k f_c(x_{k+1})$;
  update μ_k to μ_{k+1};
  $k = k + 1$;
Output: $X = X_k$

2.5.2 Alternating Directions ALM for Robust PCA

For the Robust PCA problem (1) the augmented Lagrangian is (associating $x = (L, S)$)

$$\mathcal{L}(L, S, Y, \mu) = \|L\|_* + \lambda \|S\|_1 + \langle Y, M - L - S \rangle + \frac{\mu}{2} \|M - L - S\|_F^2 \tag{29}$$

Note that one of the usual assumptions made when using an augmented Lagrangian method is that the objective function f_o is differentiable. This is clearly not the case for the Robust PCA problem. However, in [12] the authors show that the same convergence properties as for the differentiable case can be retained also for the Robust PCA problem. The main step in Algorithm 5 is solving the problem $x_{k+1} = \arg\min_x \mathcal{L}(x, \lambda_k, \mu_k)$, which in the case of the Robust PCA problem reads

$$(L_{k+1}, S_{k+1}) = \arg\min_{L,S} \ \|L\|_* + \lambda \|S\|_1 + \langle Y_k, M - L - S \rangle + \frac{\mu_k}{2} \|M - L - S\|_F^2 \tag{30}$$

and can be solved using an alternating directions approach based on the ideas in section 2.2. Specifically, consider (30) for S fixed. The corresponding "directional" subproblem in the variable L

can be formulated as

$$\begin{aligned} L_{k+1} &= \arg\min_L \ \|L\|_* + \lambda\|S\|_1 + \langle Y_k, M - L - S\rangle + \frac{\mu_k}{2}\|M - L - S\|_F^2 \\ &= \arg\min_L \ \|L\|_* + \langle Y_k, M - L - S\rangle + \frac{1}{2\mu_k^{-1}}\|M - L - S\|_F^2 \\ &= \arg\min_L \ \|L\|_* + \frac{1}{2\mu_k^{-1}}\left\|L - (M - S + \mu_k^{-1}Y_k)\right\|_F^2 \end{aligned} \tag{31}$$

Suppose $U\Sigma V^T = M - S + \mu_k^{-1}Y_k$ is the SVD of $M - S + \mu_k^{-1}Y_k$. Using the same arguments as in section 2.2, the minimizer in (31) is given by $L_{k+1} = U\mathcal{S}_{\mu_k^{-1}}[\Sigma]V^T$, where $\mathcal{S}_\varepsilon$ is the soft-thresholding operator defined in (10).

Conversely, suppose that L in (30) is fixed. The "directional" subproblem in the variable S is

$$\begin{aligned} S_{k+1} &= \arg\min_S \ \|L\|_* + \lambda\|S\|_1 + \langle Y_k, M - L - S\rangle + \frac{\mu_k}{2}\|M - L - S\|_F^2 \\ &= \arg\min_L \ \lambda\|S\|_1 + \langle Y_k, M - L - S\rangle + \frac{1}{2\mu_k^{-1}}\|M - L - S\|_F^2 \\ &= \arg\min_L \ \lambda\|S\|_1 + \frac{1}{2\mu_k^{-1}}\left\|S - (M - L + \mu_k^{-1}Y_k)\right\|_F^2 \end{aligned} \tag{32}$$

Again using the results from section 2.2 we find that $S_{k+1} = \mathcal{S}_{\lambda\mu_k^{-1}}[M - L + \mu_k^{-1}Y_k]$.

The alternating directions method for solving (30) is based on solving (31) and (32) iteratively, alternating between using a fixed S to update L and using the newly computed L to update S until convergence. Note that one can use prior iterates L_k and S_k for hot-starting the alternative directions based solution of (30) in order to significantly reduce the necessary number of directional steps within each iteration. These are all the ingredients needed for the alternating directions ALM method, which is given is Algorithm 6.

Algorithm 6: Alternating Directions Augmented Lagrangian Method

Input: Observation matrix M, parameter λ
initialization: $k = 0$, $\mu_0 > 0$, $\hat{Y}_0 = \text{sign}(M)/J(\text{sign}(M))$;
while *not converged* **do**
  $j = 0$, $L_{k+1}^0 = \hat{L}_k$, $S_{k+1}^0 = \hat{S}_k$;
  while *not converged* **do**
    $(U, \Sigma, V) = \text{svd}(M - S_{k+1}^j + \mu_k^{-1}\hat{Y}_k)$;
    $L_{k+1}^{j+1} = U\mathcal{S}_{\mu_k^{-1}}[\Sigma]V^T$;
    $S_{k+1}^{j+1} = \mathcal{S}_{\lambda\mu_k^{-1}}[M - L_{k+1}^{j+1} + \mu_k^{-1}\hat{Y}_k]$;
    $j = j + 1$;
  $\hat{Y}_{k+1} = \hat{Y}_k + \mu_k(M - \hat{L}_{k+1} - \hat{S}_{k+1})$;
  Update μ_k to μ_{k+1};
  $k = k + 1$;
Output: $L = \hat{L}_k$, $S = \hat{S}_k$

The following theorem gives the main convergence result for Algorithm 6.

Theorem 3 ([12]). *For Algorithm 6, any accumulation point* $(\hat{L}, \hat{S})$ *of* $(\hat{L}_k, \hat{S}_k)$ *is an optimal solution to the Robust PCA problem* (1) *with convergence rate of at least* $O(\mu_{k-1}^{-1})$ *in the sense that*

$$\left| ||\hat{L}_k||_* + \lambda ||\hat{S}_k||_1 - f^* \right| = O(\mu_{k-1}^{-1})$$

where f^* *is the optimal value of the Robust PCA problem.*

2.5.3 Inexact ALM for Robust PCA

The authors in [12] point out that it is not really necessary to solve (30) exactly at each iteration k by running the alternating directions subroutine until convergence. It turns out that, under some technical conditions on the sequence $\{\mu_k\}$, one can also prove convergence of the overall algorithm when performing only a single step in either direction at each iteration k. This results in Algorithm 7 which in [12] is referred to as the "Inexact Augmented Lagrangian Method" (IALM).

Algorithm 7: Inexact Augmented Lagrangian Method

Input: Observation matrix M, parameter λ
initialization: $k = 0$, $\mu_0 > 0$, $Y_0 = M/J(M)$;
while *not converged* **do**
  $(U, \Sigma, V) = \text{svd}(M - S_k + \mu_k^{-1} Y_k)$;
  $L_{k+1} = U \mathcal{S}_{\mu_k^{-1}}[\Sigma] V^T$;
  $S_{k+1} = \mathcal{S}_{\lambda \mu_k^{-1}}[M - L_{k+1} + \mu_k^{-1} Y_k]$;
  $\hat{Y}_{k+1} = Y_k + \mu_k (M - L_{k+1} - S_{k+1})$;
  Update μ_k to μ_{k+1};
  $k = k + 1$;
Output: $L = \hat{L}_k$, $S = \hat{S}_k$

While it is possible to show convergence of Algorithm 7, the authors do not provide a bound on the rate of convergence as they do for the "exact ALM" (Algorithm 6). However, empirical results suggest that the IALM algorithm is generally significantly faster that the EALM algorithm with only slightly lower accuracy.

3 Discussion of Algorithms

The previous section discussed a number of algorithms for Robust PCA. The naive application of general-purpose interior point solvers to the dual problem works well for small problems but does not scale with the size of the data matrix M. The simple iterative thresholding algorithm can handle very large problem sizes but its convergence is extremely slow. Both the APG and the dual gradient ascent algorithm are much faster and better suited for large problems. An extension of the APG algorithm that uses a prediction strategy for the dimension of the principal singular space whose singular values are larger than the threshold allows to only compute a partial SVD in each step of the algorithm. With this extension the the APG algorithm turns out to be faster than the dual gradient ascent algorithm.

The current state of the art in terms of complexity, accuracy and convergence seem to be the ALM methods discussed in section 2.5. Though the authors of [12] do not give a complexity analysis for

the inexact ALM algorithm, empirical results suggest that for practical problems the inexact ALM is considerably faster than its exact counterpart.

3.1 Numerical Comparison

In this section we provide some simulation results to illustrate the performance of the different algorithms. As seen in section 2.1.2, using interior point methods to solve the dual problem is computationally infeasible for all but the smallest problems, hence

3.2 The Importance of the SVD

Most of the presented algorithms (in fact, all of them except for the direct use of interior point solvers on the dual) involve repeated computations of the Singular Value Decomposition (or at least a partial SVD) of matrices of considerable size. This is not very surprising, as the nuclear norm in the objective function is the sum of the singular values of the matrix argument. This repeated computation of the SVD is in fact the bottleneck of most current algorithms for Robust PCA. Hence it seems that, at least for the algorithms discussed above, being able to perform SVD on large matrices are the key to developing fast algorithms that can be used in applications with large-scale data.

3.2.1 Comparison of different SVD algorithms

Figure 2 shows a comparison of the average computation time of the SVD of matrices of different sizes using Matlab's internal SVD routine (based on the algorithm in [8]) and PROPACK [10], which is based on a Lanczos bidiagonalization algorithm with partial reorthogonalization. We have compared the average running time for square matrices of different sizes, up to dimension 3000. For each size we generated 20 random matrices, and determined the average running time over this set. From Figure 2 we see that for smaller matrices Matlab's internal SVD routine is significantly faster than PROPACK. On the other hand, for large matrices (of dimension greater than 500×500) PROPACK is much faster than the Matlab routine. In fact, for the largest matrices we tested it is about one order of magnitude faster. This shows that the SVD-based algorithms discussed above can all benefit strongly from fast SVD algorithms.

When the matrix under consideration is sparse, the SVD can be carried out much faster using specialized methods [4]. Unfortunately, while the iterates of the matrix S_k (in case of primal algorithms) are indeed sparse, the matrices for which the SVD actually needs to be computed are not. Therefore it does not seem clear how sparse SVD methods could help in improving performance of SVD-based Robust PCA algorithms.

3.2.2 Partial SVD methods

If we look at the a Singular Value Thresholding operation it is obvious that we really only needs to compute those singular values that lie above the specified threshold (which is known a priori in each step and does not depend on the singular values), since the other singular values will be set to zero anyway. One possibility to speed up the algorithms that involve thresholding of singular values

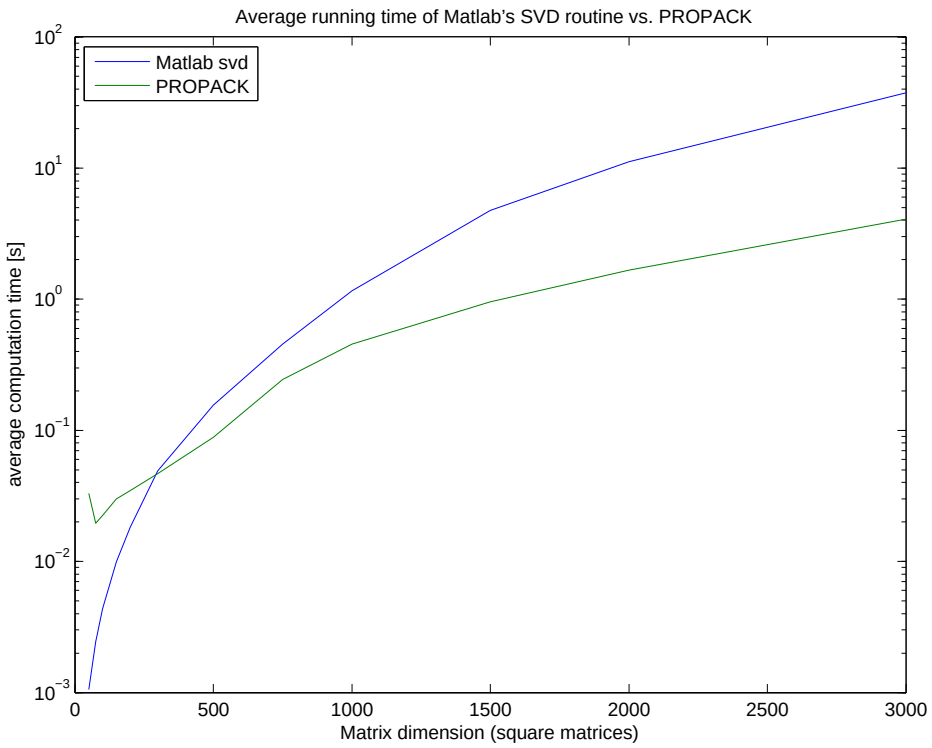

Figure 2: Numerical comparison of Matlab's and PROPACK's SVD routines

is therefore to use a partial SVD to compute only those singular values of interest. For the APG algorithm the authors of [12] use the software PROPACK that allows the computation of such a partial SVD. However, PROPACK by default requires the dimension of the principal singular space whose singular values are larger than the threshold, which is of course unknown a priori. Since it turns out that the rank of the the iterates L_k in the APG algorithm is monotonically increasing, a reasonable prediction of this dimension is not too hard [12]. Doing so then allows to a partial SVD at each step rather than a full SVD, which can potentially speed up the algorithm (if the partial SVD can be computed efficiently).

The PROPACK package has later also been modified to allow the computation of those singular values above a given threshold [11]. Figure 3 shows a comparison of the computation times of PROPACK's standard (full) SVD routine and the algorithm [11] for different thresholds. Contrary to what one would expect, the full SVD in all cases is significantly faster than the partial SVD. At this point it is not clear what the reason for this is. Either the algorithm for performing the "thresholded SVD" itself is very slow, or the implementation that is provided by [11] is extremely inefficient (or both). Either way, it obviously makes no sense to use to the provided implementation to compute partial SVDs. This is not to say that partial SVD is in generally slow. In fact, in [12] it is found that computing the partial SVD for a fixed dimension of the principal singular space using PROPACK indeed results in a speedup of the APG algorithm.

If we look at the discussed algorithms, we notice another problem with the partial SVD methods: the value of the threshold for the singular value thresholding operation in all cases decreases with the iteration number. Hence the number of singular values that need be computed actually grows with the number of iterations, which means that computing a partial SVD yields the highest benefit

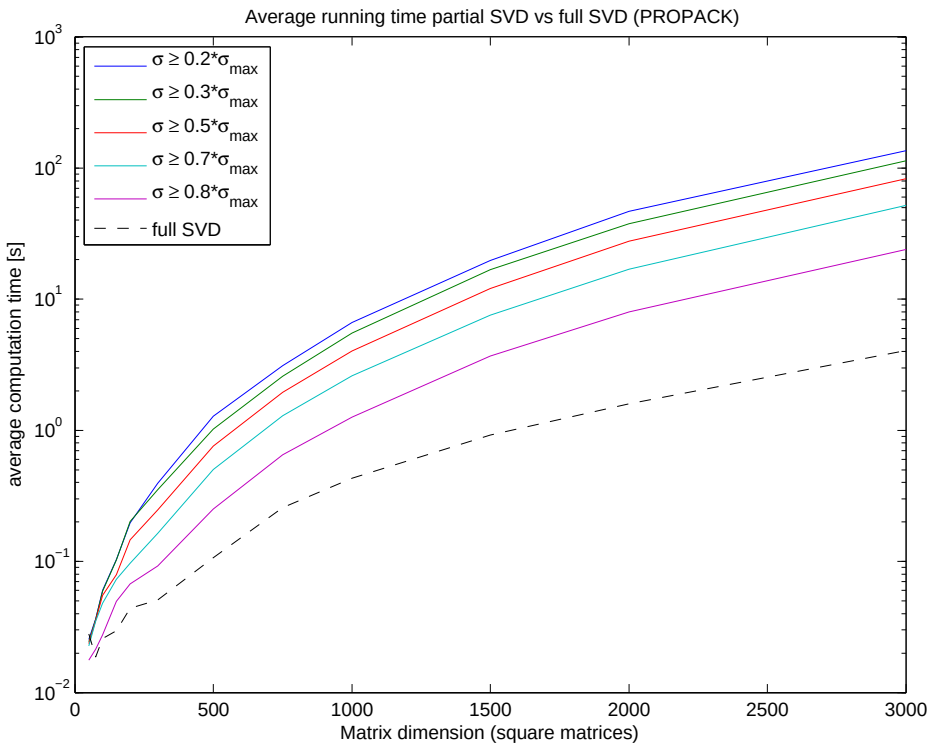

Figure 3: Numerical comparison of partial and full SVD via PROPACK

only in the early stages of the algorithm.

3.2.3 Warm-start methods for SVD

Another potential way of speeding up many SVD-based algorithms is to exploit the fact that the matrix of which the SVD has to be computed does generally not change much between iterations, in particular after a few iterations. To utilize this fact, the authors of [14] propose a warm start technique for the block Lanczos method for SVD computation.

Figure 4 shows a comparison of the performance of the warm-start block Lanczos method [14] on synthetic data of different size. It can be seen that the running time of the warm-start method is roughly two thirds of that of the standard method, over all dimensions considered in the simulation. The relative errors of the solution, given by $\|\hat{L} - L\|_F / \|L\|_F$ and $\|\hat{S} - S\|_F / \|S\|_F$, respectively, are generally worse than those of the standard method. This is due to the fact that only an approximate SVD is computed at each iteration. What is remarkable in Figure 4 is that the number of iterations is essentially independent of the matrix size. Note however that this strong consistency very likely also has to do with the fact that all the considered randomly generated matrices are structurally identical.

We want to emphasize here that the implementation of the warm-start block Lanczos method, as provided by [11], is completely Matlab-based and not fully optimized for performance. A careful implementation of the complete warm-start block Lanczos based inexact ALM algorithm in a performance-oriented language such as C can be expected to yield further speedups. Nevertheless,

even with the current implementation it is possible to solve rather large problems quite fast, with problems involving matrices with tens of million entries being solved in a few minutes.

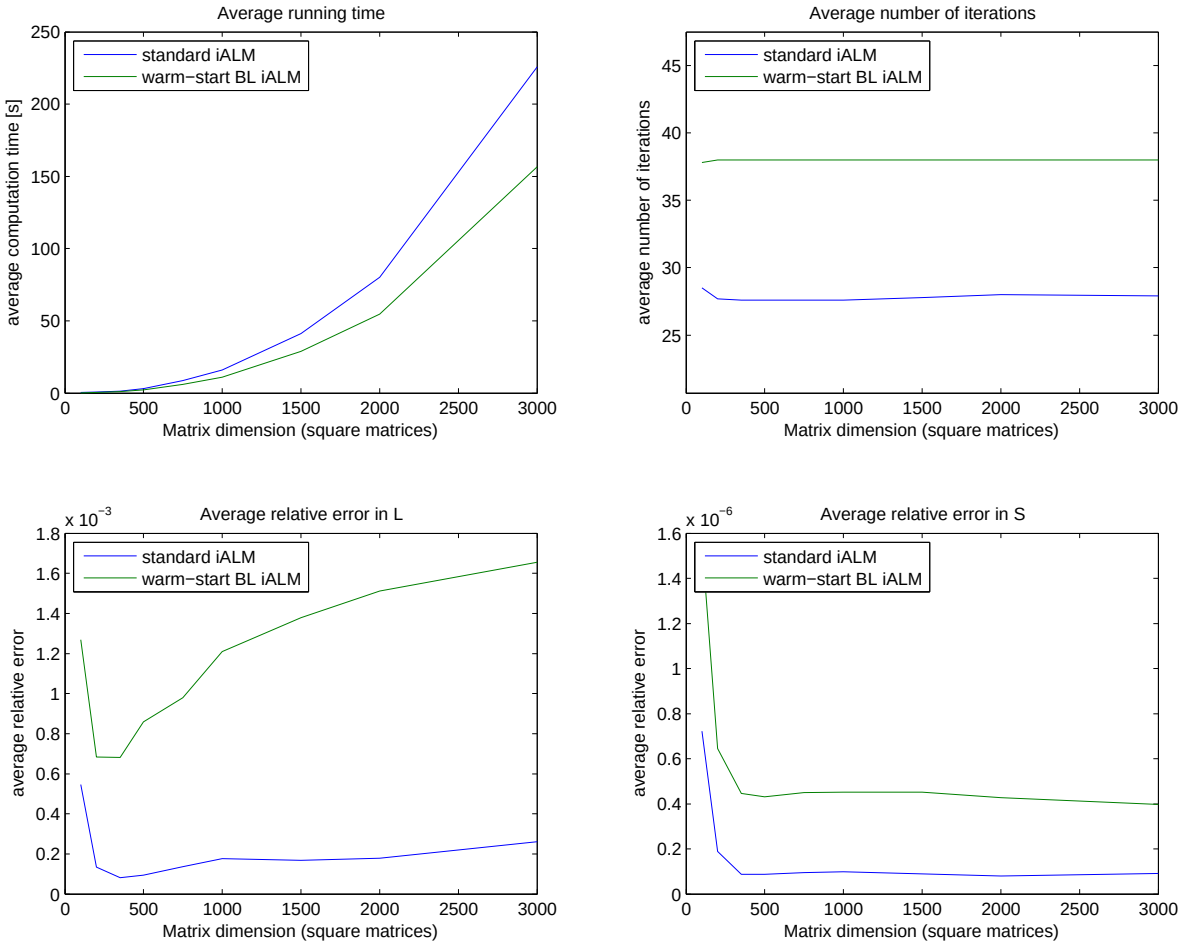

Figure 4: Numerical comparison of warm-start vs. standard Block-Lanczcos method

3.3 Possible Directions for Parallelization

With the advent of highly parallelized computing architectures in modern CPUs and GPUs, a number of projects have been devoted to the development and implementation of algorithms that exploit this massive computing power. Examples for these are MAGMA [18] and CULA [9], which are adapting classic high-performance linear algebra packages such as LAPACK and BLAS to the highly parallelized architecture of modern GPUs.

Some of the steps in the discussed SVD-based algorithms are inherently parallelizable, for example the entry-wise soft-thresholding of a matrix, the elementary matrix operations or the computation of the Frobenius norm for the stopping criterion. The parallelization of the SVD is less obvious, but research in this area is quite active and some methods have been proposed [4]. At this point we do not want to go into the details of how to parallelize the surveyed algorithms (in particular the fast augmented Lagrangian methods), we merely want to point out that we indeed see potential for further speedups and hence the use of the discussed algorithms also on large-scale data.

4 Outlook: Algorithms for Stable Principal Component Pursuit

As discussed in section REFERENCE, the results obtained for the Robust PCA problem have been extended to the case when in addition to the sparse noise, the data is corrupted also by a small non-sparse noise component [23]. This problem is usually referred to as Stable Principal Component Pursuit (Stable PCP). The associated optimization problem is

$$\begin{aligned} p^* = \min_{L,S} \ & \|L\|_* + \lambda \|S\|_1 \\ \text{s.t.} \quad & \|M - L - S\|_F \leq \delta \end{aligned} \tag{33}$$

where $\delta > 0$ is given. In this section we want to give a brief outlook on what algorithms have been proposed for (33). While to date the literature on algorithms for this problem is less extensive than for the classic Robust PCA problem, a number of efficient ALM algorithms have already been proposed.

First of all, it is not hard to reformulate problem (33) as an SDP, which can then in principle be solved using general purpose interior point solvers. However, the same scalability issues as in the standard Robust PCA problem will prohibit the use of these methods for most problems of practical interest.

In the original Stable PCP paper [23], the authors directly use the Accelerated Proximal Gradient (APG) algorithm from section 2.3.2 for solving an approximate version of (33). Using a duality argument it is easy to see that (33) is equivalent to

$$\min_{L,S} ||L||_* + \lambda ||S||_1 + \frac{1}{2\mu} ||M - L - S||_F$$

for some value $\mu(\delta)$. Note that the above problem is just (20). The authors make the argument that it makes sense to choose μ to be the smallest value such that the minimizer of is likely to be $\hat{L} = \hat{S} = 0$ if $L = S = 0$ and $M = Z$. In this way, μ is large enough to threshold away the noise, but not too large to over-shrink the original matrices. Assuming an iid Gaussian distribution of the dense noise component, i.e. $(Z_0)_i, j) \sim \mathcal{N}(0, \sigma^2)$, it turns out[5] that $n^{-1/2}\|Z_0\| \to \sqrt{2}\sigma$ a.s. as $n \to \infty$. As a result of this, the choice for μ becomes $\mu = \sqrt{2n}\sigma$. We note that the assumption of a Gaussian noise matrix Z_0 is often reasonable but not always satisfied. If it is not, then it is not clear whether using the APG algorithm to solve the associated approximate problem is a good idea and different algorithms may be needed.

In [20] an algorithm for (33) based on partial variable splitting is proposed[6]. In fact, the problem an simply be written as

$$\begin{aligned} p^* = \min_{L,S,Z} \ & \|L\|_* + \lambda \|S\|_1 \\ \text{s.t.} \quad & M = L + S + Z \\ & \|Z\|_F \leq \delta \end{aligned} \tag{34}$$

At heart, the proposed ASALM algorithm is the extension of the ALM method from section 2.5.2 to the case with partial variable splitting. It turns out that the additional subproblem appearing

[5] this based on the strong Bai Yin Theorem [2], which implies that for an $n \times n$ real matrix with entries $\xi_{ij} \sim \mathcal{N}(0, 1)$ the it holds that $\limsup_{n\to\infty} \|Z_0\|_2/\sqrt{n} = 2$ almost surely

[6] The authors in fact consider the more general problem where the constraint reads $\|\mathcal{P}_\Omega(M - L - S)\|_F \leq \delta$, where $\mathcal{P}_\Omega$ is the projection on the set Ω of observed data

at each iteration can be solved explicitly at a cost similar to the matrix value thresholding (the cost of which is $O(mn)$). The dominant computation is still the singular value thresholding operation that is based on the computation of the SVD. Hence the proposed method is comparable in computational complexity to the ALM method from section 2.5.2.

In [1] a number of different algorithms for Stable PCP are discussed, most of which are based on a smoothed or partially smoothed objective function. Two of these algorithms essentially apply Nesterov's optimal algorithms [15] to the partially smoothed problem, which yields a theoretical complexity of $O(1/\varepsilon)$. It is further shown that the subproblems appearing in these algorithms either have a closed-form solution or can be solved very efficiently. Still using a partially smoothed objective function, the authors also apply a partial variable splitting technique and propose to use an alternating minimization algorithm for a linearized version of the associated augmented Lagrangian function. This also yields an algorithm with a theoretical complexity of $O(1/\varepsilon)$.

The authors of [1] also propose a first-order algorithm that works directly with the fully non-smooth objective, which they call NSA. Very similar in nature to the ASALM method proposed in [20], this algorithm is also an extension of the ALM method from section 2.5.2 to the setting with partial variable splitting. While the authors were not able to derive theoretical complexity results, empirical evidence suggests that this algorithm is quite efficient in solving the Stable PCP problem. In particular, numerical simulations indicate that NSA consistently outperforms ASALM. The main reason for this seems to be that while ASALM performs an alternating minimization over three directions (L, S and Z), NSA uses the fact that the joint minimization over (S, Z) also has an explicit solution and therefore only alternates over two directions.

References

[1] N. S. Aybat, D. Goldfarb, and G. Iyengar. Fast First-Order Methods for Stable Principal Component Pursuit. *arXiv preprint*, 1105.2126S, May 2011.

[2] Z. D. Bai and Y. Q. Yin. Necessary and sufficient conditions for almost sure convergence of the largest eigenvalue of a wigner matrix. *The Annals of Probability*, 16(4):pp. 1729–1741, 1988.

[3] A. Beck and M. Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM Journal on Imaging Sciences*, 2(1):183–202, 2009.

[4] M. Berry, D. Mezher, B. Philippe, and A. Sameh. *Handbook of parallel computing and statistics*, chapter Parallel Algorithms for the Singular Value Decomposition. Statistics, textbooks and monographs. Chapman & Hall/CRC, 2005.

[5] D. Bertsekas. *Constrained optimization and Lagrange multiplier methods*. Optimization and neural computation series. Athena Scientific, 1996.

[6] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[7] J.-F. Cai, E. J. Candès, and Z. Shen. A singular value thresholding algorithm for matrix completion. *SIAM J. on Optimization*, 20(4):1956–1982, Mar. 2010.

[8] J. Demmel and W. Kahan. Accurate singular values of bidiagonal matrices. *SIAM Journal on Scientific and Statistical Computing*, 11(5):873–912, 1990.

[9] J. R. Humphrey, D. K. Price, K. E. Spagnoli, A. L. Paolini, and E. J. Kelmelis. Cula: hybrid gpu accelerated linear algebra routines. volume 7705, page 770502. SPIE, 2010.

[10] R. M. Larsen. Lanczos bidiagonalization with partial reorthogonalization. Technical Report DAIMI PB-357, Dept. of Computer Science, Aarhus University, Sep 1998.

[11] Z. Lin. Some Software Packages for Partial SVD Computation. *Arxiv preprint arXiv:1108.1548*, Aug. 2011.

[12] Z. Lin, M. Chen, and Y. Ma. The Augmented Lagrange Multiplier Method for Exact Recovery of Corrupted Low-Rank Matrices. *arXiv preprint*, Sept. 2010.

[13] Z. Lin, A. Ganesh, J. Wright, L. Wu, M. Chen, and Y. M. Fast convex optimization algorithms for exact recovery of a corrupted low-rank matrix. Technical Report UILU-ENG-09-2214, UIUC, Jul 2009.

[14] Z. Lin and S. Wei. A Block Lanczos with Warm Start Technique for Accelerating Nuclear Norm Minimization Algorithms. *Arxiv preprint arXiv:1012.0365*, Dec. 2010.

[15] Y. Nesterov. Smooth minimization of non-smooth functions. *Mathematical Programming*, 103:127–152, 2005.

[16] Y. E. Nesterov. A method for solving the convex programming problem with convergence rate $O(1/k^2)$. *Soviet Math. Dokl.*, 27(2):372–376, 1983.

[17] Y. E. Nesterov. Gradient methods for minimizing composite objective function. Technical report, CORE, 2007.

[18] B. J. Smith. R package magma: Matrix algebra on gpu and multicore architectures, version 0.2.2, August 2010. http://icl.cs.utk.edu/magma/index.html.

[19] J. F. Sturm. Using sedumi 1.02, a matlab toolbox for optimization over symmetric cones. *Optimization Methods and Software*, 11(1-4):625–653, 1999.

[20] M. Tao and X. Yuan. Recovering low-rank and sparse components of matrices from incomplete and noisy observations. *SIAM Journal on Optimization*, 21(1):57–81, 2011.

[21] K. C. Toh, M. J. Todd, and R. H. Tütüncü. Sdpt3 — a matlab software package for semidefinite programming, version 1.3. *Optimization Methods and Software*, 11(1-4):545–581, 1999.

[22] J. Wright, A. Ganesh, S. Rao, and Y. Ma. Robust principal component analysis: Exact recovery of corrupted low-rank matrices via convex optimization. *Submitted to Journal of the ACM*, 2009.

[23] Z. Zhou, X. Li, J. Wright, E. Candès, and Y. Ma. Stable principal component pursuit. In *Information Theory Proceedings (ISIT), 2010 IEEE International Symposium on*, pages 1518 –1522, june 2010.