THE UNIVERSITY of EDINBURGH

Edinburgh Research Explorer

Alternating Maximization

Citation for published version:
Richtárik, P, Taká, M & Damla Ahipaaolu, S 2012 'Alternating Maximization: Unifying Framework for 8 Sparse PCA Formulations and Efficient Parallel Codes' ArXiv. <http://uk.arxiv.org/abs/1212.4137>

Link:
Link to publication record in Edinburgh Research Explorer

General rights
Copyright for the publications made accessible via the Edinburgh Research Explorer is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
The University of Edinburgh has made every reasonable effort to ensure that Edinburgh Research Explorer content complies with UK legislation. If you believe that the public display of this file breaches copyright please contact openaccess@ed.ac.uk providing details, and we will remove access to the work immediately and investigate your claim.

OPEN ACCESS

Download date: 28. Aug. 2022

arXiv:1212.4137v1 [stat.ML] 17 Dec 2012

Alternating Maximization: Unifying Framework for 8 Sparse PCA Formulations and Efficient Parallel Codes *

Peter Richtárik† Martin Takáč‡ S. Damla Ahipaşaoğlu§

December 15, 2012

Abstract

Given a multivariate data set, sparse principal component analysis (SPCA) aims to extract several linear combinations of the variables that together explain the variance in the data as much as possible, while controlling the number of nonzero loadings in these combinations. In this paper we consider 8 different optimization formulations for computing a single sparse loading vector; these are obtained by combining the following factors: we employ *two* norms for measuring variance (L2, L1) and *two* sparsity-inducing norms (L0, L1), which are used in *two* different ways (constraint, penalty). Three of our formulations, notably the one with L0 constraint and L1 variance, have not been considered in the literature. We give a unifying reformulation which we propose to solve via a natural alternating maximization (AM) method. We show the the AM method is nontrivially equivalent to GPower (Journée et al; JMLR **11**:517–553, 2010) for all our formulations. Besides this, we provide 24 efficient parallel SPCA implementations: 3 codes (multi-core, GPU and cluster) for each of the 8 problems. Parallelism in the methods is aimed at i) speeding up computations (our GPU code can be 100 times faster than an efficient serial code written in C++), ii) obtaining solutions explaining more variance and iii) dealing with big data problems (our cluster code is able to solve a 357 GB problem in about a minute).

Keywords: sparse principal component analysis, alternating maximization, GPower, high performance computing, big data analytics, unsupervised learning.

1 Introduction

Principal component analysis (PCA) is an indispensable tool used for dimension reduction in virtually all areas of science and engineering, from machine learning, statistics, genetics and finance to computer networks [1]. Let $A \in \mathbf{R}^{n\times p}$ denote a data matrix encoding n samples (observations) of p variables (features). PCA aims to extract a few linear combinations of the columns of A, called principal components (PCs), pointing in mutually orthogonal directions, together explaining as much variance in the data as possible. If the columns of A are centered, the problem of extracting the first PC can be written as

$$\max\{\|Ax\| : \|x\|_2 \leq 1\}, \tag{1.1}$$

*Open source code with efficient implementations of the algorithms developed in this paper is published here: https://code.google.com/p/24am/

†School of Mathematics, University of Edinburgh, Edinburgh, EH93JZ, United Kingdom

‡School of Mathematics, University of Edinburgh, Edinburgh, EH93JZ, United Kingdom

§Engineering Systems and Design, Singapore University of Technology and Design, Singapore, 138682

where $\|\cdot\|$ is a suitable norm for measuring variance. The solution x of this optimization problem is called the loading vector, Ax (normalized) is the first PC. Further PCs can be obtained in the same way with A replaced by a new matrix in a process called deflation [2]. Classical PCA employs the L_2 norm in the objective; using the L_1 norm instead may alleviate problems caused by outliers in the data and hence leads to a robust PCA model [3].

As normally there is no reason for the optimal loading vectors defining the PCs to be sparse, they are usually combinations of all of the variables. In some applications, however, sparse loading vectors enhance the *interpretability* of the components and are easier to store, which leads to the idea to *induce* sparsity in the loading vectors. This problem and approaches to it are known collectively as sparse PCA (SPCA); for some recent work, see [4]-[11]. A popular way of incorporating a sparsity-inducing mechanism into the above optimization formulation is via either a sparsity-inducing constraint or penalty. Two of the most popular functions for this are the L_0 and L_1 norm of the loading vector x (the L_0 "norm" of x, denoted by $\|x\|_0$, is the number of nonzeros in x).

1.1 Eight optimization formulations

In this paper we consider 8 optimization formulations for extracting a single sparse loading vector (i.e., for computing the first PC) arising as combinations of the following three modeling factors: we use two norms for measuring variance (classical L_2 and robust L_1) and two sparsity-inducing (SI) norms (cardinality L_0 and L_1), which are used in two different ways (as a constraint or a penalty). All have the form

$$OPT = \max_{x \in X} f(x), \tag{1.2}$$

with $X \subset \mathbf{R}^p$ and f detailed in Table 1. Note that if we set $s = p$ in the constrained or $\gamma = 0$ in the penalized versions, the sparsity-inducing functions stop having any effect[1] and we recover the classical and robust PCA (1.1). Choosing $1 \leq s < p$, $\gamma > 0$ will have the effect of directly enforcing or indirectly encouraging sparsity in the solution x.

#	Variance	SI norm	SI norm usage	X	$f(x)$
1	L_2	L_0	constraint	$\{x \in \mathbf{R}^p \ : \ \|x\|_2 \leq 1, \ \|x\|_0 \leq s\}$	$\|Ax\|_2$
2	L_1	L_0	constraint	$\{x \in \mathbf{R}^p \ : \ \|x\|_2 \leq 1, \ \|x\|_0 \leq s\}$	$\|Ax\|_1$
3	L_2	L_1	constraint	$\{x \in \mathbf{R}^p \ : \ \|x\|_2 \leq 1, \ \|x\|_1 \leq \sqrt{s}\}$	$\|Ax\|_2$
4	L_1	L_1	constraint	$\{x \in \mathbf{R}^p \ : \ \|x\|_2 \leq 1, \ \|x\|_1 \leq \sqrt{s}\}$	$\|Ax\|_1$
5	L_2	L_0	penalty	$\{x \in \mathbf{R}^p \ : \ \|x\|_2 \leq 1\}$	$\|Ax\|_2^2 - \gamma\|x\|_0$
6	L_1	L_0	penalty	$\{x \in \mathbf{R}^p \ : \ \|x\|_2 \leq 1\}$	$\|Ax\|_1^2 - \gamma\|x\|_0$
7	L_2	L_1	penalty	$\{x \in \mathbf{R}^p \ : \ \|x\|_2 \leq 1\}$	$\|Ax\|_2 - \gamma\|x\|_1$
8	L_1	L_1	penalty	$\{x \in \mathbf{R}^p \ : \ \|x\|_2 \leq 1\}$	$\|Ax\|_1 - \gamma\|x\|_1$

Table 1: Eight sparse PCA optimization formulations; see 1.2.

All 4 SPCA formulations of Table 1 involving L_2 variance were previously studied in the literature and are very popular. For instance, [6] solve a series of convex relaxations of the L_0 constrained

[1] In the L_1 penalized formulations this can be seen from the inequality $\|x\|_1 \leq \sqrt{\|x\|_0}\|x\|_2$.

L_2 variance problem, [7] considered the L_0 penalized and constrained formulations, [9] studied the L_0 and L_1 penalized versions, [10] looked at all four. The L_1 constrained L_1 variance formulation was first proposed only recently, in [11]. To the best of our knowledge, the remaining three L_1 variance formulations were not considered in the literature before. In particular, the L_0 constrained L_1 variance formulation is new—and is perhaps preferable as it directly constraints the cardinality of the loading vector x without using any proxies.

1.2 Reformulation and alternating maximization (AM) method

In all 8 formulations we introduce an additional (dummy) variable y, which allows us to propose a generic *alternating maximization* method for solving them: i) for a fixed loading vector, find the best dummy variable (one maximizing the objective), then ii) fix the dummy variable and find the best loading vector; repeat steps i) and ii). This and the resulting algorithms are described in detail in Section 2. The generic AM method is not limited to our choice of SPCA formulations. Indeed, it is applicable, for instance, if instead of measuring the variance using either the L_1 or the L_2 norm, we use any other norm. One critical feature shared by the formulations in Table 1 is that steps i) and ii) of the AM method can be performed efficiently, in closed form, with the main computational burden in each step being a matrix-vector multiplication (Ax in step i) and $A^T y$ in step ii)). Our method produces a sequence of loading vectors $x^{(k)},\ k \geq 0$, with monotonically increasing values $f(x^{(k)})$.

Our approach of introducing a dummy variable and using AM is similar to that of [9], where it is done *implicitly*, but mainly to [12], where it is fully *explicit*, albeit used for different purposes.

Besides providing a conceptual unification for solving all 8 formulations using a single algorithm (AM), the main theoretical result of this paper is establishing that, surprisingly, in all 8 cases, *the AM method is equivalent to the GPower method* [9] applied to a certain derived objective function, with iterates being either the loading vectors or the dummy variables, depending on the formulation. This result is stated and proved in Section 3.

1.3 Parallelism

Besides giving a new unifying framework and a generic algorithm for solving a number of SPCA formulations, 5 of which were previously proposed in the literature and 3 not, our further contribution is in providing efficient strategies for parallelizing AM at two different levels: i) running AM in parallel from multiple starting points in order to obtain a solution explaining more variance and ii) speeding up the linear algebra involved. This is described in detail in Section 4.

Moreover, we provide parallel open-source codes implementing these parallelization strategies, for each of our 8 formulations, on 3 computing architectures: i) *multi-core machine*, ii) *GPU-enabled computer*, and iii) *computer cluster*. We also provide a serial code; however, as nearly all modern computers are multi-core, the serial implementation only serves the purpose of a benchmark against which once can measure parallelization speedup. Hence, we provide a total of $8 \times 3 = 24$ parallel sparse PCA codes based on AM. Numerical experiments with our multi-core, GPU and cluster codes are performed in Section 5.

Parallelism in our codes serves several purposes:

1. *Speeding up computations.* As described above, the AM method computes a matrix-vector multiplication at every iteration; this can be parallelized. We find that our GPU implementa-

tions are faster than our multi-core implementations, which are, in turn, considerably faster than the benchmark single-core codes.

2. *Obtaining solutions explaining more variance.* In some applications, such as in the computation of RIP constants for compressed sensing [13], it is critical that a PC is computed with as high explained variance as possible. The output of our 8 subroutines depends on the starting point used; it only finds local solutions. Running them repeatedly from different starting points and keeping the solution with the largest objective value results in a PC explaining more variance. There are several ways in which this can be done, we implement 4 (NAI = "naive", SFA = "start-from-all", BAT = "batches" and OTF = "on-the-fly"); details are given in Section 4. A naive (NAI) approach is to do this sequentially; a different possibility is to run the method from several or all starting points in parallel (BAT, SFA), possibly asynchronously (OTF). This way at each iteration we need to perform a matrix-matrix multiplication which, when computed in parallel, is performed significantly faster compared to doing the corresponding number of parallel matrix-vector multiplications, one after another.

3. *Dealing with big data problems.* If speed matters, for problems of small enough size we recommend using a GPU, if available. Since GPUs have stricter memory limitations than multi-core workstations (a typical GPU has 6GB RAM, a multi-core machine could have 20GB RAM), one may need to use a high-memory multi-core workstation if the problem size exceeds the GPU limit. However, for large enough (=big data) problems, one will need to use a cluster. Our cluster codes partition A, store parts of it on different nodes, and do the computations in a distributed way.

Notation. By x and y we denote column vectors in $\mathbf{R}^p$ and $\mathbf{R}^n$, respectively. The coordinates of a vector are denoted by subscripts (eg., $x_1, x_2, \dots$) while iterates are denoted by superscripts in brackets (eg., $x^{(0)}$, $x^{(1)}$, $\dots$). We reserve the letter k for the iteration counter. By $\|x\|_0$ we refer to the cardinality (number of nonzero loadings) of vector x. The L_1, L_2 and L_∞ norms are defined by $\|z\|_1 = \sum_i |z_i|$, $\|z\|_2 = (\sum_i z_i^2)^{1/2}$ and $\|z\|_\infty = \max_i |z_i|$, respectively. For a scalar t, we let $[t]_+ = \max\{0, t\}$ and by $\mathbf{sgn}(t)$ we denote the sign of t.

2 Alternating Maximization (AM) Method

As outlined in the previous section, we will solve (1.2) by introducing a dummy variable y into each of the 8 formulations and apply an AM method to the reformulation. First, notice that for any pair of conjugate norms $\|\cdot\|$ and $\|\cdot\|^*$, we have, by definition,

$$\|z\| = \max_{\|y\|^* \leq 1} y^T z. \tag{2.3}$$

In particular, $\|\cdot\|_2^* = \|\cdot\|_2$ and $\|\cdot\|_1^* = \|\cdot\|_\infty$.

Now, let $Y := \{y \in \mathbf{R}^n \ : \ \|y\|_2 \leq 1\}$ for the L_2 variance formulations and $Y := \{y \in \mathbf{R}^n \ : \ \|y\|_\infty \leq 1\}$ for the L_1 variance formulations. Further, let $F(x, y)$ be the function obtained from $f(x)$ after replacing $\|Ax\|$ with $y^T Ax$ (resp. $\|Ax\|^2$ with $(y^T Ax)^2$). Then, in view of the above, (1.2) takes on the equivalent form

$$OPT = \max_{x \in X} \max_{y \in Y} F(x, y). \tag{2.4}$$

That is, the 8 problems from Table 1 can be reformulated into the form (2.4); the details can be found in Table 2.

#	X	Y	$F(x,y)$
1	$\{x \in \mathbf{R}^p \ : \ \|x\|_2 \leq 1, \ \|x\|_0 \leq s\}$	$\{y \in \mathbf{R}^n \ : \ \|y\|_2 \leq 1\}$	$y^T Ax$
2	$\{x \in \mathbf{R}^p \ : \ \|x\|_2 \leq 1, \ \|x\|_0 \leq s\}$	$\{y \in \mathbf{R}^n \ : \ \|y\|_\infty \leq 1\}$	$y^T Ax$
3	$\{x \in \mathbf{R}^p \ : \ \|x\|_2 \leq 1, \ \|x\|_1 \leq \sqrt{s}\}$	$\{y \in \mathbf{R}^n \ : \ \|y\|_2 \leq 1\}$	$y^T Ax$
4	$\{x \in \mathbf{R}^p \ : \ \|x\|_2 \leq 1, \ \|x\|_1 \leq \sqrt{s}\}$	$\{y \in \mathbf{R}^n \ : \ \|y\|_\infty \leq 1\}$	$y^T Ax$
5	$\{x \in \mathbf{R}^p \ : \ \|x\|_2 \leq 1\}$	$\{y \in \mathbf{R}^n \ : \ \|y\|_2 \leq 1\}$	$(y^T Ax)^2 - \gamma\|x\|_0$
6	$\{x \in \mathbf{R}^p \ : \ \|x\|_2 \leq 1\}$	$\{y \in \mathbf{R}^n \ : \ \|y\|_\infty \leq 1\}$	$(y^T Ax)^2 - \gamma\|x\|_0$
7	$\{x \in \mathbf{R}^p \ : \ \|x\|_2 \leq 1\}$	$\{y \in \mathbf{R}^n \ : \ \|y\|_2 \leq 1\}$	$y^T Ax - \gamma\|x\|_1$
8	$\{x \in \mathbf{R}^p \ : \ \|x\|_2 \leq 1\}$	$\{y \in \mathbf{R}^n \ : \ \|y\|_\infty \leq 1\}$	$y^T Ax - \gamma\|x\|_1$

Table 2: Reformulations of the problems from Table 1.

We propose to solve (2.4) via Algorithm 1.

Algorithm 1 Alternating Maximization (AM) Method.

Select initial point $x^{(0)} \in \mathbf{R}^p$; $k \leftarrow 0$
Repeat
$y^{(k)} \leftarrow y(x^{(k)}) := \arg\max_{y \in Y} F(x^{(k)}, y)$
$x^{(k+1)} \leftarrow x(y^{(k)}) := \arg\max_{x \in X} F(x, y^{(k)})$
Until a stopping criterion is satisfied

2.1 Solving the subproblems

All 8 problems of Table 2 enjoy the property that both of the steps (subproblems) of Algorithm 1 can be computed in closed form. In particular, each of these 8×2 subproblems is of one of the 6 forms listed in Table 3.

The proofs of these elementary results, many of which are of folklore nature, can be found, for instance, in [10] (and partially in [9]). The columns of Table 3, from left to right, correspond to the objective function, feasible region, maximizer (optimal solution) and maximum (optimal objective value). The first result will be used both with $z = x$ and $z = y$, the second result with $z = y$ and the remaining four results with $z = x$.

Table 3 is brief at the cost of referring to a number of operators ($T_s, U_\gamma, V_\gamma : \mathbf{R}^m \mapsto \mathbf{R}^m$ and $\lambda_s : \mathbf{R}^m \mapsto \mathbf{R}$), which we will now define. For a given vector $a \in \mathbf{R}^m$ and integer $s \in \{0, 1, \ldots, m\}$, by $T_s(a) \in \mathbf{R}^m$ we denote the vector obtained from a by retaining only the s largest components of a in absolute value, with the remaining ones replaced by zero. For instance, for $a = (1, -4, 2, 5, 3)^T$ and $s = 2$ we have $T_s(a) = (0, -4, 0, 5, 0)^T$. For $\gamma \geq 0$, we define operators U_γ and V_γ element-wise for $i = 1, \ldots, m$ as follows:

$$(U_\gamma(a))_i := a_i[\mathbf{sgn}(a_i^2 - \gamma)]_+, \tag{2.5}$$

Subproblem #	$\phi(z)$	Z	z^*	$\phi(z^*)$
S1	$a^T z$ or $(a^T z)^2$	$\|z\|_2 \leq 1$	$\frac{a}{\|a\|_2}$	$\|a\|_2$
S2	$a^T z$	$\|z\|_\infty \leq 1$	$\mathbf{sgn}(a)$	$\|a\|_1$
S3	$a^T z$	$\|z\|_2 \leq 1,\ \|z\|_0 \leq s$	$\frac{T_s(a)}{\|T_s(a)\|_2}$	$\|T_s(a)\|_2$
S4	$a^T z$	$\|z\|_2 \leq 1,\ \|z\|_1 \leq \sqrt{s}$	$\frac{V_{\lambda_s(a)}(a)}{\|V_{\lambda_s(a)}(a)\|_2}$	$\lambda_s(a)\sqrt{s} + \|V_{\lambda_s(a)}(a)\|_2$
S5	$(a^T z)^2 - \gamma\|z\|_0$	$\|z\|_2 \leq 1$	$\frac{U_\gamma(a)}{\|U_\gamma(a)\|_2}$	$\|U_\gamma(a)\|_2^2 - \gamma\|U_\gamma(a)\|_0$
S6	$a^T z - \gamma\|z\|_1$	$\|z\|_2 \leq 1$	$\frac{V_\gamma(a)}{\|V_\gamma(a)\|_2}$	$\|V_\gamma(a)\|_2$

Table 3: Closed-form solutions of AM subproblems; $z^* := \arg\max_{z\in Z} \phi(z)$.

$$(V_\gamma(a))_i := \mathbf{sgn}(a_i)(|a_i| - \gamma)_+. \tag{2.6}$$

Furthermore, we let

$$\lambda_s(a) := \arg\min_{\lambda\geq 0} \lambda\sqrt{s} + \|V_\lambda(a)\|_2,$$

which is the solution of the one-dimensional dual of the optimization problem in line 4 of Table 3.

2.2 The AM method for all 8 SPCA formulations

Combining Algorithm 1 with the subproblem solutions given in Table 3, the AM method for all our 8 SPCA formulations can be written down concisely; see Algorithm 2.

Algorithm 2 AM method for solving the 8 SPCA formulations of Table 2.

Select initial point $x^{(0)} \in \mathbf{R}^p$; $k \leftarrow 0$
Repeat
  $u = Ax^{(k)}$
    If L_1 variance **then** $y^{(k)} \leftarrow \mathbf{sgn}(u)$
    If L_2 variance **then** $y^{(k)} \leftarrow u/\|u\|_2$
  $v = A^T y^{(k)}$
    If L_0 penalty **then** $x^{(k+1)} \leftarrow U_\gamma(v)/\|U_\gamma(v)\|_2$
    If L_1 penalty **then** $x^{(k+1)} \leftarrow V_\gamma(v)/\|V_\gamma(v)\|_2$
    If L_0 constraint **then** $x^{(k+1)} \leftarrow T_s(v)/\|T_s(v)\|_2$
    If L_1 constraint **then** $x^{(k+1)} \leftarrow V_{\lambda_s(v)}(v)/\|V_{\lambda_s(v)}(v)\|_2$
  $k \leftarrow k + 1$
Until a stopping criterion is satisfied

Note that in the methods described in Algorithm 2 it is not necessary to normalize the vector $U_\gamma(v)$ (resp. $V_\gamma(v)$, $T_s(v)$, and $V_{\lambda_s(a)}(v)$) when computing $x^{(k+1)}$ since clearly the iterate $y^{(k+1)}$, which depends on $x^{(k+1)}$, is invariant under positive scalings of $x^{(k+1)}$. We have to remember, however, to normalize the output. The method is terminated when a maximum number of iterations $maxIt$ is reached or when

$$\frac{F(x^{(k+1)}, y^{(k)})}{F(x^{(k)}, y^{(k-1)})} \leq 1 + tol,$$

whichever happens sooner.

3 Equivalence of AM and GPower

GPower (generalized power method) [9] is a simple algorithm for maximizing a convex function Ψ on a compact set Ω, which works via a "linearize and maximize" strategy. If by $\Psi'(z^{(k)})$ we denote an arbitrary subgradient of Ψ at $z^{(k)}$, then GPower performs the following iteration:

$$z^{(k+1)} = \arg\max_{z\in\Omega}\{\Psi(z^{(k)}) + \langle\Psi'(z^{(k)}), z - z^{(k)}\rangle\} = \arg\max_{z\in\Omega}\langle\Psi'(z^{(k)}), z\rangle. \tag{3.7}$$

The following theorem, our main result, gives a nontrivial insight into the relationship of AM and GPower, when the former is applied to solving any of the 8 SPCA formulations considered, and GPower is applied to a derived problem, as described by the theorem.

Theorem 1 (AM = GPower)**.** *The AM and GPower methods are equivalent in the following sense:*

1. *For the 4 constrained sparse PCA formulations of Table 1, the x iterates of the AM method applied to the corresponding reformulation of Table 2 are identical to the iterates of the GPower method as applied to the problem of maximizing the convex function*

$$F_Y(x) \stackrel{def}{=} \max_{y\in Y} F(x,y)$$

   on X, started from $x^{(0)}$.

2. *For the 4 penalized sparse PCA formulations of Table 1, the y iterates of the AM method applied to the corresponding reformulation of Table 2 are identical to the iterates of the GPower method as applied to the problem of maximizing the convex function*

$$F_X(y) \stackrel{def}{=} \max_{x\in X} F(x,y)$$

   on Y, started from $y^{(0)}$.

Proof. Recall that we wish to solve the problem

$$OPT = \max_{x\in X} f(x) = \max_{x\in X} \underbrace{\max_{y\in Y} F(x,y)}_{F_Y(x)} = \max_{y\in Y} \underbrace{\max_{x\in X} F(x,y)}_{F_X(y)}.$$

We will now prove the equivalence for all 8 choices of (f, X, Y, F) given in Tables 1 and 2. In the proofs we will also refer to the closed form solutions of the subproblem (S1)–(S6), as detailed in Table 3.

Consider first the constrained formulations: $1, 2, 3$ and 4. By induction assume that the k-th x-iterate ($x^{(k)}$) of AM is identical to the k-th iterate of GPower (for $k = 0$ this is enforced by the assumption that GPower is started from $x^{(0)}$). By considering all 4 formulations individually, we will show that $x^{(k+1)}$ produced by AM and GPower are also identical.

Formulation 1: Here we have

$$f(x) = \|Ax\|_2, \qquad F(x,y) = y^T Ax,$$

$$X = \{x \in \mathbf{R}^p \ : \ \|x\|_2 \leq 1, \ \|x\|_0 \leq s\}, \qquad Y = \{y \in \mathbf{R}^n \ : \ \|y\|_2 \leq 1\}.$$

First, note that

$$F_Y(x) = \max_{y \in Y} F(x,y) \stackrel{(S1)}{=} \|Ax\|_2,$$

the gradient of which is given by

$$F_Y'(x) = \frac{A^T Ax}{\|Ax\|_2}. \tag{3.8}$$

Given $x^{(k)}$, in the AM method we have

$$y^{(k)} = \arg\max_{y \in Y} F(x^{(k)}, y) \stackrel{(S1)}{=} \frac{Ax^{(k)}}{\|Ax^{(k)}\|_2}. \tag{3.9}$$

One iteration of GPower started from $x^{(k)}$ will thus produce the iterate

$$\begin{aligned} x^{(k+1)} \stackrel{(3.7)}{=} \arg\max_{x \in X} \langle F_Y'(x^{(k)}), x \rangle &\stackrel{(3.8)}{=} \arg\max_{x \in X} \left\langle \frac{A^T Ax^{(k)}}{\|Ax^{(k)}\|_2}, x \right\rangle \\ &\stackrel{(3.9)}{=} \arg\max_{x \in X} \langle A^T y^{(k)}, x \rangle \\ &\stackrel{(S3)}{=} \frac{T_s(A^T y^{(k)})}{\|T_s(A^T y^{(k)})\|_2}. \end{aligned}$$

Observe that this is precisely how $x^{(k+1)}$ is computed in the AM method.

Formulation 2: Here we have

$$f(x) = \|Ax\|_1, \qquad F(x,y) = y^T Ax,$$

$$X = \{x \in \mathbf{R}^p \ : \ \|x\|_2 \leq 1, \ \|x\|_0 \leq s\}, \qquad Y = \{y \in \mathbf{R}^n \ : \ \|y\|_\infty \leq 1\}.$$

First, note that

$$F_Y(x) = \max_{y \in Y} F(x,y) \stackrel{(S2)}{=} \|Ax\|_1,$$

the gradient of which is given by

$$F_Y'(x) = A^T \mathbf{sgn}(Ax). \tag{3.10}$$

Given $x^{(k)}$, in the AM method we have

$$y^{(k)} = \arg\max_{y \in Y} F(x^{(k)}, y) \stackrel{(S2)}{=} \mathbf{sgn}(Ax^{(k)}). \tag{3.11}$$

One iteration of GPower started from $x^{(k)}$ will thus produce the iterate

$$\begin{aligned} x^{(k+1)} \overset{(3.7)}{=} \arg\max_{x\in X}\langle F_Y'(x^{(k)}), x\rangle &\overset{(3.10)}{=} \arg\max_{x\in X}\left\langle A^T \mathbf{sgn}(Ax^{(k)}), x\right\rangle \\ &\overset{(3.11)}{=} \arg\max_{x\in X}\langle A^T y^{(k)}, x\rangle \\ &\overset{(S3)}{=} \frac{T_s(A^T y^{(k)})}{\|T_s(A^T y^{(k)})\|_2}. \end{aligned}$$

Observe that this is precisely how $x^{(k+1)}$ is computed in the AM method.

Formulation 3: Here we have

$$f(x) = \|Ax\|_2, \qquad F(x,y) = y^T Ax,$$
$$X = \{x \in \mathbf{R}^p \ : \ \|x\|_2 \le 1, \ \|x\|_1 \le \sqrt{s}\}, \qquad Y = \{y \in \mathbf{R}^n \ : \ \|y\|_2 \le 1\}.$$

First, note that

$$F_Y(x) = \max_{y\in Y} F(x,y) \overset{(S1)}{=} \|Ax\|_2,$$

the gradient of which is given by

$$F_Y'(x) = \frac{A^T Ax}{\|Ax\|_2}. \tag{3.12}$$

Given $x^{(k)}$, in the AM method we have

$$y^{(k)} = \arg\max_{y\in Y} F(x^{(k)}, y) \overset{(S1)}{=} \frac{Ax^{(k)}}{\|Ax^{(k)}\|_2}. \tag{3.13}$$

One iteration of GPower started from $x^{(k)}$ will thus produce the iterate

$$\begin{aligned} x^{(k+1)} \overset{(3.7)}{=} \arg\max_{x\in X}\langle F_Y'(x^{(k)}), x\rangle &\overset{(3.14)}{=} \arg\max_{x\in X}\left\langle \frac{A^T Ax^{(k)}}{\|Ax^{(k)}\|_2}, x\right\rangle \\ &\overset{(3.15)}{=} \arg\max_{x\in X}\langle A^T y^{(k)}, x\rangle \\ &\overset{(S4)}{=} \frac{V_{\lambda_s(A^T y^{(k)})}(A^T y^{(k)})}{\|V_{\lambda_s(A^T y^{(k)})}(A^T y^{(k)})\|_2}. \end{aligned}$$

Observe that this is precisely how $x^{(k+1)}$ is computed in the AM method.

Formulation 4: Here we have

$$f(x) = \|Ax\|_1, \qquad F(x,y) = y^T Ax,$$
$$X = \{x \in \mathbf{R}^p \ : \ \|x\|_2 \le 1, \ \|x\|_1 \le \sqrt{s}\}, \qquad Y = \{y \in \mathbf{R}^n \ : \ \|y\|_\infty \le 1\}.$$

First, note that

$$F_Y(x) = \max_{y \in Y} F(x,y) \overset{(S1)}{=} \|Ax\|_1,$$

the gradient of which is given by

$$F_Y'(x) = \frac{A^T A x}{\|Ax\|_2}. \tag{3.14}$$

Given $x^{(k)}$, in the AM method we have

$$y^{(k)} = \arg\max_{y \in Y} F(x^{(k)}, y) \overset{(S1)}{=} \frac{Ax^{(k)}}{\|Ax^{(k)}\|_2}. \tag{3.15}$$

One iteration of GPower started from $x^{(k)}$ will thus produce the iterate

$$\begin{aligned} x^{(k+1)} \overset{(3.7)}{=} \arg\max_{x \in X} \langle F_Y'(x^{(k)}), x \rangle &\overset{(3.14)}{=} \arg\max_{x \in X} \left\langle \frac{A^T A x^{(k)}}{\|Ax^{(k)}\|_2}, x \right\rangle \\ &\overset{(3.15)}{=} \arg\max_{x \in X} \langle A^T y^{(k)}, x \rangle \\ &\overset{(S4)}{=} \frac{V_{\lambda_s(A^T y^{(k)})}(A^T y^{(k)})}{\|V_{\lambda_s(A^T y^{(k)})}(A^T y^{(k)})\|_2}. \end{aligned}$$

Observe that this is precisely how $x^{(k+1)}$ is computed in the AM method.

Consider now the penalized formulations: $5, 6, 7$ and 8. By induction assume that the k-th y-iterate ($y^{(k)}$) of AM is identical to the k-th iterate of GPower (for $k = 0$ this is enforced by the assumption that GPower is started from $y^{(0)}$). By considering all 4 formulations individually, we will show that $y^{(k+1)}$ produced by AM and GPower are also identical. Let $A = [a_1, \ldots, a_p]$, i.e., the i-th column of A is a_i.

Formulation 5: Here we have

$$f(x) = \|Ax\|_2^2 - \gamma \|x\|_0, \qquad F(x,y) = (y^T A x)^2 - \gamma \|x\|_0,$$
$$X = \{x \in \mathbf{R}^p \ : \ \|x\|_2 \leq 1\}, \qquad Y = \{y \in \mathbf{R}^n \ : \ \|y\|_2 \leq 1\}.$$

First, note that

$$F_X(y) = \max_{x \in X} F(x,y) \overset{(S5)}{=} \|U_\gamma(A^T y)\|_2^2 - \gamma \|U_\gamma(A^T y)\|_0 = \sum_{i=1}^p [(a_i^T y)^2 - \gamma]_+,$$

the subgradient of which is given by

$$F_X'(y) = 2 \sum_{i=1}^p [\mathbf{sgn}((a_i^T y) - \gamma)]_+ (a_i^T y) a_i \overset{(2.5)}{=} 2 A U_\gamma(A^T y). \tag{3.16}$$

Given $y^{(k)}$, in the AM method we have

$$x^{(k+1)} = \arg\max_{x\in X} F(x, y^{(k)}) \overset{(S5)}{=} \frac{U_\gamma(A^T y^{(k)})}{\|U_\gamma(A^T y^{(k)})\|_2}. \tag{3.17}$$

One iteration of GPower started from $y^{(k)}$ will thus produce the iterate

$$\begin{aligned} y^{(k+1)} \overset{(3.7)}{=} \arg\max_{y\in Y} \langle F_X'(y^{(k)}), y\rangle &\overset{(3.16)}{=} \arg\max_{\|y\|_\infty\le 1} \langle 2AU_\gamma(A^T y), y\rangle \\ &\overset{(3.17)}{=} \arg\max_{\|y\|_2\le 1} \langle Ax^{(k+1)}, y\rangle \\ &\overset{(S1)}{=} \frac{Ax^{(k+1)}}{\|Ax^{(k+1)}\|_2}. \end{aligned}$$

Observe that this is precisely how $y^{(k+1)}$ is computed in the AM method.

Formulation 6: Here we have

$$f(x) = \|Ax\|_1^2 - \gamma\|x\|_0, \qquad F(x,y) = (y^T Ax)^2 - \gamma\|x\|_0,$$
$$X = \{x \in \mathbf{R}^p \ : \ \|x\|_2 \le 1\}, \qquad Y = \{y \in \mathbf{R}^n \ : \ \|y\|_\infty \le 1\}.$$

First, note that

$$F_X(y) = \max_{x\in X} F(x,y) \overset{(S5)}{=} \|U_\gamma(A^T y)\|_2^2 - \gamma\|U_\gamma(A^T y)\|_0 = \sum_{i=1}^p [(a_i^T y)^2 - \gamma]_+,$$

the subgradient of which is given by

$$F_X'(y) = 2\sum_{i=1}^p [\mathbf{sgn}((a_i^T y) - \gamma)]_+ (a_i^T y) a_i \overset{(2.5)}{=} 2AU_\gamma(A^T y). \tag{3.18}$$

Given $y^{(k)}$, in the AM method we have

$$x^{(k+1)} = \arg\max_{x\in X} F(x, y^{(k)}) \overset{(S5)}{=} \frac{U_\gamma(A^T y^{(k)})}{\|U_\gamma(A^T y^{(k)})\|_2}. \tag{3.19}$$

One iteration of GPower started from $y^{(k)}$ will thus produce the iterate

$$\begin{aligned} y^{(k+1)} \overset{(3.7)}{=} \arg\max_{y\in Y} \langle F_X'(y^{(k)}), y\rangle &\overset{(3.18)}{=} \arg\max_{\|y\|_\infty\le 1} \langle 2AU_\gamma(A^T y), y\rangle \\ &\overset{(3.19)}{=} \arg\max_{\|y\|_\infty\le 1} \langle Ax^{(k+1)}, y\rangle \\ &\overset{(S2)}{=} \mathbf{sgn}(Ax^{(k+1)}). \end{aligned}$$

Observe that this is precisely how $y^{(k+1)}$ is computed in the AM method.

Formulation 7: Here we have

$$f(x) = \|Ax\|_2 - \gamma\|x\|_1, \qquad F(x,y) = y^T Ax - \gamma\|x\|_1,$$
$$X = \{x \in \mathbf{R}^p \ : \ \|x\|_2 \leq 1\}, \qquad Y = \{y \in \mathbf{R}^n \ : \ \|y\|_2 \leq 1\}.$$

Note that the functions $y \mapsto F(x,y)$ are linear and that, by definition, $F_X(y) = \max_{x \in X} F(x,y)$. Moreover, note that the gradient of $y \mapsto F(x,y)$ at y is equal to Ax. Hence, if x is any vector that maximizes $F(x, y^{(k)})$ over X, then Ax is a subgradient of F_X at $y^{(k)}$. Note that this is precisely how $x^{(k+1)}$ is defined in the AM method: $x^{(k+1)} = \arg\max_{x \in X} F(x, y^{(k)})$. Hence, $Ax^{(k+1)}$ is a subgradient of F_X at $y^{(k)}$ and one iteration of GPower started from $y^{(k)}$ will produce the iterate

$$y^{(k+1)} \stackrel{(3.7)}{=} \arg\max_{y \in Y} \langle F_X'(y^{(k)}), y \rangle = \arg\max_{\|y\|_2 \leq 1} \langle Ax^{(k+1)}, y \rangle \stackrel{(S1)}{=} \frac{Ax^{(k+1)}}{\|Ax^{(k+1)}\|_2}.$$

Observe that this is precisely how $y^{(k+1)}$ is computed in the AM method.

Formulation 8: Here we have

$$f(x) = \|Ax\|_1 - \gamma\|x\|_1, \qquad F(x,y) = y^T Ax - \gamma\|x\|_1,$$
$$X = \{x \in \mathbf{R}^p \ : \ \|x\|_2 \leq 1\}, \qquad Y = \{y \in \mathbf{R}^n \ : \ \|y\|_\infty \leq 1\}.$$

Note that the functions $y \mapsto F(x,y)$ are linear and that, by definition, $F_X(y) = \max_{x \in X} F(x,y)$. Moreover, note that the gradient of $y \mapsto F(x,y)$ at y is equal to Ax. Hence, if x is any vector that maximizes $F(x, y^{(k)})$ over X, then Ax is a subgradient of F_X at $y^{(k)}$. Note that this is precisely how $x^{(k+1)}$ is defined in the AM method: $x^{(k+1)} = \arg\max_{x \in X} F(x, y^{(k)})$. Hence, $Ax^{(k+1)}$ is a subgradient of F_X at $y^{(k)}$ and one iteration of GPower started from $y^{(k)}$ will produce the iterate

$$y^{(k+1)} \stackrel{(3.7)}{=} \arg\max_{y \in Y} \langle F_X'(y^{(k)}), y \rangle = \arg\max_{\|y\|_\infty \leq 1} \langle Ax^{(k+1)}, y \rangle \stackrel{(S2)}{=} \mathbf{sgn}(Ax^{(k+1)}).$$

Observe that this is precisely how $y^{(k+1)}$ is computed in the AM method.

□

Having established equivalence between AM and GPower, local convergence of the AM method for all 8 SPCA formulations follows from the theory developed in [9] and [10].

4 Embedding AM within a Parallel Scheme

In this section we describe several approaches for embedding Algorithm 2 (AM) within a parallel scheme for solving l identical SPCA problems, started from a number of starting points, $x^{(0,1)}, \ldots, x^{(0,l)}$. This is done in order to obtain a loading vector explaining more variance and will be discussed in more detail in Section 4.1.

As we will see, it may not necessarily be most efficient to solve *all* l problems simultaneously. Instead, we consider a class of parallelization schemes where we divide the l problems into "batches"

of r problems each, and solve each batch of r problems simultaneously. In this setting at each iteration we need to perform identical operations in parallel, notably matrix-vector multiplications $Ax^{(k,1)}, \ldots, Ax^{(k,r)}$ and $A^T y^{(k,1)}, \ldots, A^T y^{(k,r)}$. It is useful to view the sequence of matrix-vector products as a single matrix-matrix product, e.g., $A[x^{(k,1)}, \ldots, x^{(k,r)}]$ in the first case, and use optimized libraries for parallelization. This simple trick leads to considerable speedups when compared to other approaches. We use similar ideas for the parallel evaluation of the operators. Note that even in the $l = 1$ case, i.e, if we wish to run SPCA from a single starting point only, there is scope for parallelization of the matrix-vector products and function evaluations. Hence, parallelization in our method serves two purposes:

1. to obtain solutions explaining more variance by solving the problem from several starting points (we choose $l > 1$),
2. to speed up computations by parallelizing the linear algebra involved (this applies to both $l = 1$ and $l > 1$ cases).

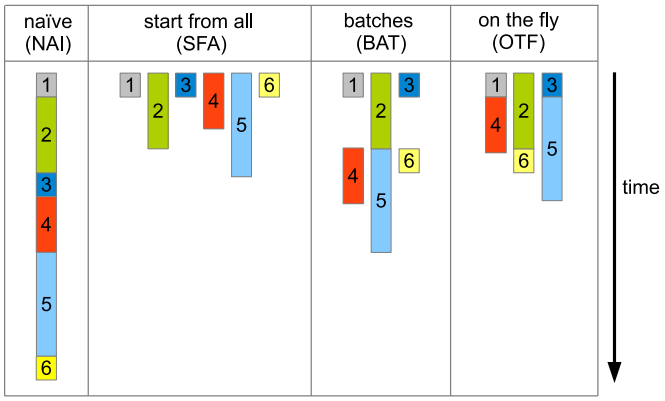

Figure 1: Four ways of embedding Algorithm 2 (AM) in a parallel scheme. In this example we run AM on the same problem $l = 6$ times, using different starting points.

In particular, in this section we describe 4 parallelization approaches:

- NAI = "naive" ($r = 1$),
- SFA = "start-from-all" ($r = l$),
- BAT = "batches" ($l < r < l$)
- OTF = "on-the-fly" (BAT improved by a dynamic replacement strategy to reduce idle time).

The working of these 4 approaches is illustrated in Figure 1 in a situation with $l = 6$. In what follows we describe the methods informally, in a narrative style, with a suitable choice of numerical experiments illustrating the differences between the ideas.

4.1 The hunt for more explained variance

As shown in [9], [10] for GPower, and due to our equivalence theorem (Theorem 1), we know that Algorithm 2 (AM) is only able to converge to a local solution. Moreover, quality of the solution will

depend on the starting point (SP) $x^{(0)}$ used. When the algorithm is run just once, the quality of the obtained solution, in terms of the objective value (or explained variance), can be poor. Hence, if the amount of explained variance is important, it will be useful to run the method repeatedly from a number of different SPs. In this and all subsequent experiments we generated $A \in \mathbf{R}^{n\times p}$ with independent and uniformly distributed entries from $[-1, 1]$. Here chose $n = 512$ and $p = 2,048$ (and renormalized the columns so that their norms are uniformly distributed on $[0, 1]$) and solved the corresponding L_0 constrained L_2 variance SPCA problem with $s = 1, 2, 4, \ldots, 2048$. For each s we run AM from $l = 1,000$ randomly generated SPs with $maxIt = 200$ and $tol = 10^{-6}$. The results are given in Figure 2, where the vertical axis corresponds to the amount of explained variance of a particular solution compared to the best solution found.

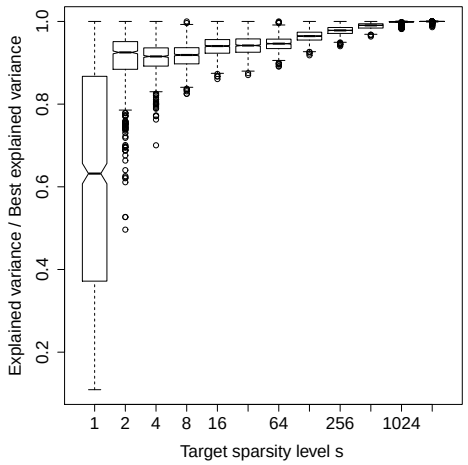

Figure 2: It may be easy to converge to a poor local solution.

Clearly, for small s it is easy to obtain a bad solution if we run the method only a few times; this effect is milder for large s but may be substantial nevertheless in real life problems. Hence, especially when s is small, it is necessary to employ a globalization strategy such as rerunning AM from a number of different starting points. This experiment illustrates that the simple strategy of running the method from a number of randomly generated starting points can be effective in finding solutions with more explained variance. A "naive" (NAI) approach would be to do this sequentially: solve the problem with one starting point first before solving it for another starting point.

4.2 Economies of scale

Running AM in parallel, started from a number of SPs, increases the utilization of computer resources, especially on parallel architectures. In order to demonstrate this, we generated 6 data matrices with $p = 1000, 2000, \ldots, 32000$ and run the AM method for the L_0 penalized L_2 variance SPCA formulation with $l = 256$ SPs (and $maxIt = 10$). By BATr we denote the approach with batches of size r. Hence, SFA = BAT256 and NAI = BAT1. Besides these two basic choices, we look at BAT4, BAT16 and BAT64 as well. The results can be found in Figure 3.

Different problem sizes p appear on the horizontal axis; on the vertical axis we plot the speedup obtained by applying a particular batching strategy compared to NAI. Note that even on a single-

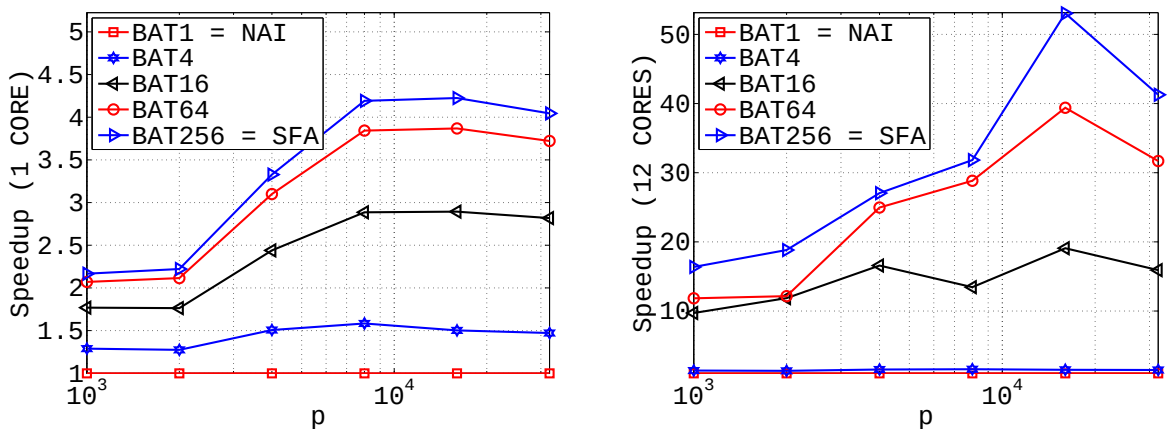

Figure 3: Economies of scale: "Start-from-all" (SFA) is better than any of the batching strategies on a single-core machine (LEFT); even more so on a multi-core machine (RIGHT).

core computer (LEFT plot) we benefit from running the methods in parallel ("economies of scale") rather than running them one after another. Indeed, we can obtain a $2-3\times$ speedup with BAT16 across the whole range of problem sizes, and $4\times$ speedup with SFA for large enough p. With 12 cores (RIGHT plot) the effect is much more dramatic: the speedup for BAT16 is consistently in the $10-20\times$ range, and can even reach $50\times$ for SFA.

4.3 Dynamic replacement

It often happens, especially when batch size is large, that some problems within a batch converge sooner than others. The vanilla BAT approach described above does nothing about it, and continues through matrix-matrix multiplies, updating the already converged iterates, until the last problem in the batch converges. A minor but not negligible speedup is possible by employing an "on-the-fly" (OTF) dynamic replacement technique, where whenever a certain problem converges, it is replaced by a new one. Hence, no predefined batches exist—OTF can be viewed as a greedy list scheduling heuristic. We used $l = 1024$ starting points and compare SFA1024 with BAT64 and OTF64–the dynamic replacement variant of BAT64.

Looking at the LEFT plot in Figure 4, we see that the average number of iterations per starting point is much smaller for OTF. This results in speedup of more than $2\times$ when compared with SFA (RIGHT plot). Notably, SFA is *slower* than both BAT64 and OTF64, which shows that it may not be optimal to choose $r = l$.

5 Multi-core Processors, GPUs and Clusters

Accompanying this paper is the open source software package **"24am"**[2] implementing parallelization strategies described in Section 4, all with Algorithm 2 (AM) used as the underlying solution method, with the option of using any of the 8 optimization formulations of SPCA described in Table 1. The name 24am comes from the fact that we implement the solver for 3 different parallel

[2] https://code.google.com/p/24am/

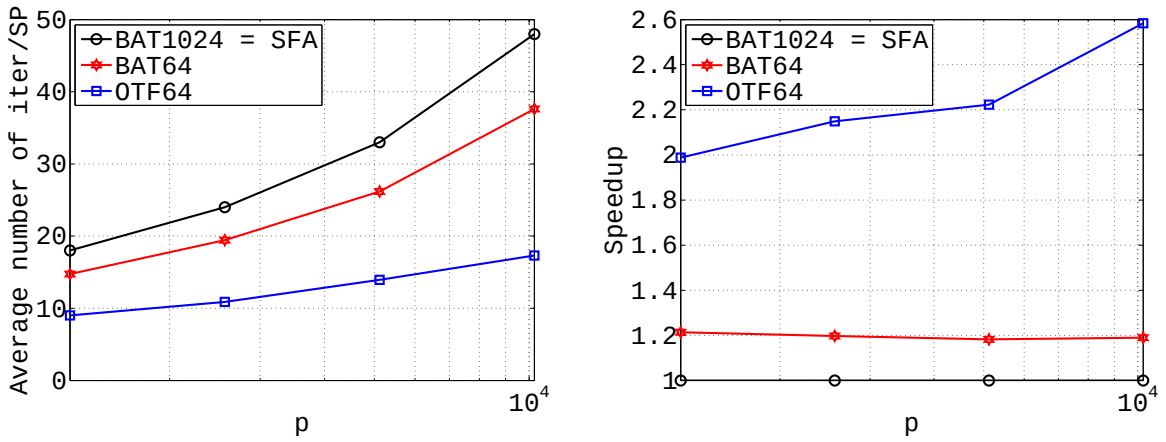

Figure 4: Dynamic Replacement: "On-the-fly" (OTF) is better than "Batches" (BAT), which is better than "start-from-all" (SFA).

architectures: multi-core processors, GPUs and computer clusters, leading to $24 = 8 \times 3$ methods based on AM.

In this the rest of this section we first perform several numerical experiments illustrating the speedups obtained by parallelization on these three computing architectures. We then conclude with a real-life numerical example (large text corpora) and a few implementation remarks.

5.1 Multi-core speedup

Here we solve 9 random L_1 constrained L_1 variance SPCA instances of sizes $p = 100 \times 2^i$, $i = 1, \ldots, 9$, $n = p/10$, with 100 SPs each, on a machine using $1, 2, 4$ and 8 cores; see Figure 5.

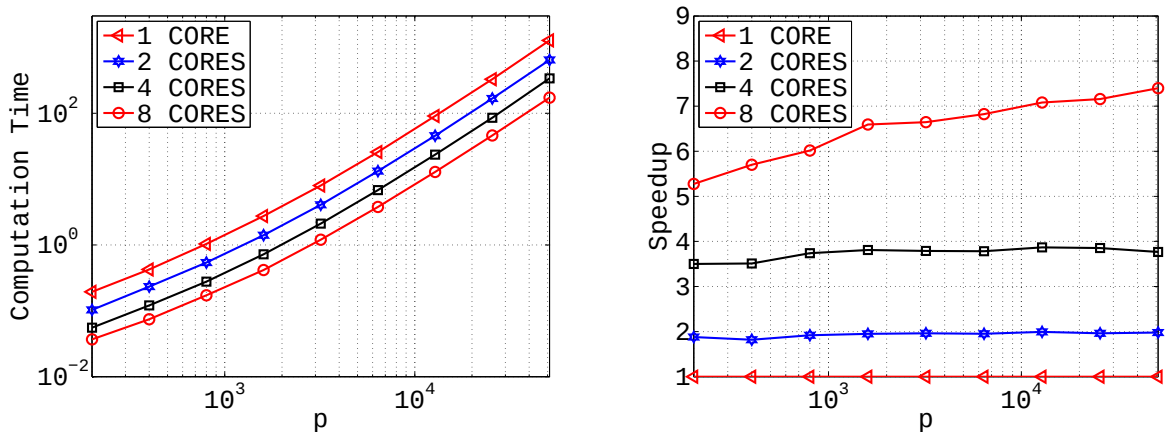

Figure 5: Multi-core speedup is proportional to the number of cores.

The plot on the LEFT shows the total computational time; the plot on the RIGHT shows the speedup of multi-core codes compared to the single-core code. Note that the speedup is consistently close to the number of cores for the 2 and 4-core setups across all problem sizes, and is growing

with p from 5× to about 7.5× in the 8-core setup.

5.2 GPU speedup

Here we solve 8 random L_1 penalized L_1 variance SPCA instances with p varying roughly between 10^3 and 10^5, and $n = p/200$. We solved all formulations with $\{1, 16, 256\}$ SPs on a single-core CPU and a GPU; the results are shown in Figure 6.

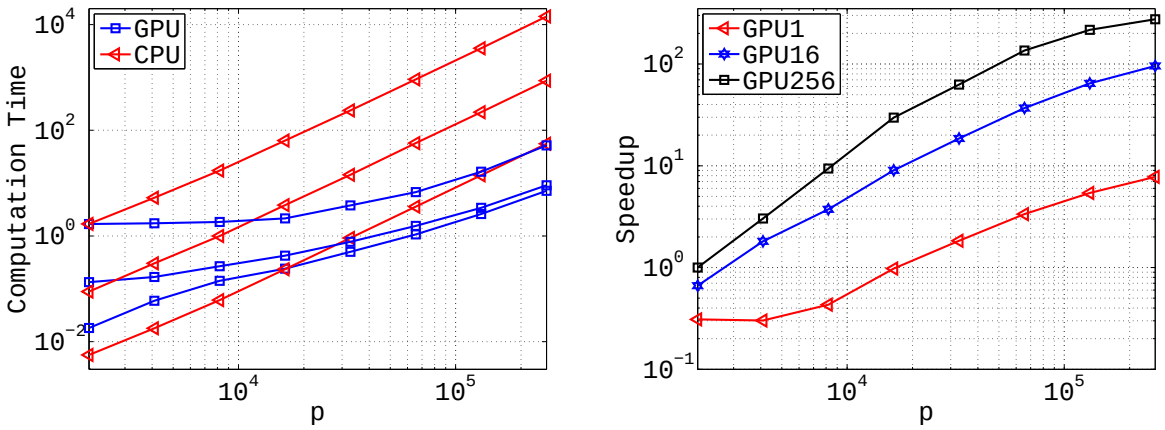

Figure 6: GPU code can achieve 125× speedup compared to single-core when 256 starting points are used.

The plot on the LEFT shows the total computational time. The red lines with triangle markers correspond to the single-core setup, the "higher" the line, the more starting points were used. The blue lines with square markers correspond to our GPU codes. While the runtime increases linearly with problem size for the single-core codes, it grows slowly for the GPU codes. Note that the GPU code may actually be slower for small problem sizes. Looking at the RIGHT plot, we see that the GPU code is capable of a 100-125× speedup; this happens for large problem sizes and 256 SPs. The speedup can reach 100× for 16 SPs as well.

5.3 Cluster code

In this experiment we solved several L_1 penalized L_2 variance SPCA problems with a *fully dense* matrix $A \in \mathbf{R}^{n\times p}$; the results are in Table 4. We focus our discussion on the largest of the problems only (last three lines of the table), one with $n = 6 \times 10^3$ and $p = 8 \times 10^6$. We used a cluster of 800 CPUs; storage of the data matrix required 357.6 GB of memory. The matrix was first loaded from files to memory; this process took $t_1 = 92$ seconds. Subsequently, the loaded data was distributed to CPUs where needed, which took additional $t_2 = 713$ seconds. Finally we run the AM method with 1, 32 and 64 starting points and measured the average time of a single iteration; the results are $t_3^1 = 4.1$, $t_3^1 = 51.1$ and $t_3^1 = 134.9$ seconds, respectively. When using a single starting point, the method would converge in about a minute. The t_3^k column of Table 4 depicts the time it takes for the solver to perform k iterations. We treated the problem directly, without using any safe feature elimination techniques [14]. Such preprocessing could, however, be able to expand the reach of our cluster code to even larger problem sizes.

$n \times p$	memory	# CPUs	GRID	SP	t_1	t_2	t_3^1	t_3^4	t_3^{16}
$10^4 \times 2 \cdot 10^5$	14.9 GB	20	10×2	1	42.68	0.86	0.56	2.06	8.48
$10^4 \times 2 \cdot 10^5$	14.9 GB	20	10×2	32	-	-	4.60	18.89	87.84
$10^4 \times 2 \cdot 10^5$	14.9 GB	20	10×2	64	-	-	10.47	37.88	166.60
$6 \cdot 10^3 \times 4 \cdot 10^5$	17.8 GB	40	10×4	1	26.89	86.33	0.78	3.15	9.96
$6 \cdot 10^3 \times 4 \cdot 10^5$	17.8 GB	40	10×4	32	-	-	7.39	27.72	125.14
$6 \cdot 10^3 \times 4 \cdot 10^5$	17.8 GB	40	10×4	64	-	-	13.19	58.36	201.51
$6 \cdot 10^3 \times 10^6$	44.7 GB	100	10×10	1	49.22	104.26	0.45	2.44	11.62
$6 \cdot 10^3 \times 10^6$	44.7 GB	100	10×10	32	-	-	6.37	29.72	115.73
$6 \cdot 10^3 \times 10^6$	44.7 GB	100	10×10	64	-	-	14.14	52.64	219.8
$6 \cdot 10^3 \times 4 \cdot 10^6$	178.8 GB	400	10×40	1	129.69	611.69	1.24	5.12	31.46
$6 \cdot 10^3 \times 4 \cdot 10^6$	178.8 GB	400	10×40	32	-	-	17.50	61.36	255.80
$6 \cdot 10^3 \times 4 \cdot 10^6$	178.8 GB	400	10×40	64	-	-	31.36	141.61	525.08
$6 \cdot 10^3 \times 8 \cdot 10^6$	357.6 GB	800	10×80	1	92.12	713.45	4.14	15.82	95.51
$6 \cdot 10^3 \times 8 \cdot 10^6$	357.6 GB	800	10×80	32	-	-	51.11	324.26	619.45
$6 \cdot 10^3 \times 8 \cdot 10^6$	357.6 GB	800	10×80	64	-	-	134.89	690.06	-

Table 4: Result from Cluster. For first 1 experiment the dimensions of virtual grid was the same as loaded data and hence t_2 is small. For the rest we used 64x64 pieces.

5.4 Large text corpora

In the first experiment we tested the AM method with L_0 constrained L_2 variance formulation (with $s = 5$) on two medium-size data sets from the *Machine Learning Repository* [18]: news articles appeared in New York Times and abstracts of articles published in PubMed. Each data set is formatted as a matrix $A \in \mathbf{R}^{n \times p}$, where the rows of A correspond to news articles in the NYTimes data set and to abstracts in PubMed, and the columns correspond to words. The number of appearances of word j in article or abstract i is the (i, j)-th entry of A; the matrices are hence clearly sparse. The NYTimes data set has 102,660 articles, 300,000 words, and approximately 70 million nonzero entries. The PubMed data set contains 141,043 articles, 8.2 million words, and approximately 484 million nonzeroes. The matrices can be stored in 0.778 GB and 5.42 GB memory space, respectively. We have customized the AM method to exploit sparsity as much as possible. In Table 5 we present the first 5 sparse principal components (5 words each). Clearly, the first PC for NYT is about sports, the second about business, the third about elections, the fourth about education and the fifth about United States. Similar interpretations can be given to the PubMed PCs.

5.5 Implementation details

For single and multi-core architectures we developed our codes using the CBLAS interface. In particular, we use both the GSL BLAS and the Intel MKL [15] implementations (single-core) and the GotoBLAS2 [16] and Intel MKL implementations (multi-core). Parallelization in the multi-core case is performed by the OpenMP interface. When comparing the performance of single-core and multi-core architectures, we use Intel MKL library for both serial and parallel versions of the same algorithm for consistency. Nevertheless, in our experience, GotoBLAS2 implementation of these algorithms are faster than the Intel MKL implementation. We use CuBLAS version 4.0 [17] on GPU (and make use of Thrust whenever possible for operations such as sorting, memory arrangements and data allocation on GPU). For comparisons between single-core and GPU architectures, we use

NYT 1st PC	NYT 2nd PC	NYT 3rd PC	NYT 4th PC	NYT 5th PC
game	companies	campaign	children	attack
play	company	president	program	government
player	million	al gore	school	official
season	percent	bush	student	US
team	stock	george bush	teacher	united states
PubMed 1st PC	PubMed 2nd PC	PubMed 3rd PC	PubMed 4th PC	PubMed 5th PC
disease	cell	activity	cancer	age
level	effect	concentration	malignant	child
patient	expression	control	mice	children
therapy	human	rat	primary	parent
treatment	protein	receptor	tumor	year

Table 5: First 5 sparse PCs for NYTimes and PubMed data sets.

the GSL BLAS implementation on the single-core. On a cluster, linear algebra is done with Intel MKL's PBLAS, while communication between nodes is via MPI.

6 Conclusion

We propose a unifying framework for solving 8 SPCA formulations in which all have the same form and are solved by the same algorithm: the alternating maximization (AM) method. We observed that AM is in all cases equivalent to the GPower method applied to a suitable convex function. Five of these formulations were previously studied in the literature and three were not; notably the L_1 constrained L_1 (robust) variance seems to be new. For each of these formulations we have written 4 efficient codes—one serial and three parallel—aimed at single-core, multi-core and GPU workstations and a cluster. All these codes are enabled with efficient parallel implementations of a multiple-starting-point globalization strategy which aims to find PCs explaining more variance; with speedup per starting point achieving up to two orders of magnitude. The most efficient of these implementations is "on-the-fly". We demonstrated that our cluster code is able to solve a very large problem with a 357 GB fully dense data matrix.

Acknowledgements

The work of P. Richtárik and M. Takáč was supported by the EPSRC grant EP/I017127/1 (Mathematics for Vast Digital Resources) and the Centre for Numerical Algorithms and Intelligent Software (funded by EPSRC grant EP/G036136/1 and the Scottish Funding Council). P. Richtárik was also partially supported by the EPSRC grant EP/J020567/1 (Algorithms for Data Simplicity). S. D. Ahipaşaoğlu was partially supported by the SUTD Start-Up Research Grant, Project Number: SRG ESD 2012 033.

References

[1] Jollife, I. (1986) Principal component analysis, Springer Verlag, NY.

[2] Mackey, L. (2008) Deflation Methods for Sparse PCA. *Advances in Neural Information Processing Systems.* **21**:1017-1024.

[3] Kwak, N. (2008) Principal component analysis based on L_1 norm maximization. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **30**:1672-1680.

[4] Zou, H., Hastie, T. and Tibshirani, R. (2004) Sparse principal component analysis. *Technical report*, Stanford University.

[5] Moghaddam, B., Weiss, Y. and Avidan, S., (2006) Spectral bounds for sparse PCA: exact and greedy algorithms. *In: Advances in Neural Information Processing Systems* **18**:915-922.

[6] d'Aspremont, A., El Ghaoui, L., Jordan, M.I. and Lanckriet, G.R.G. (2007) A direct formulation for sparse PCA using semidefinite programming. *SIAM Review* **48(3)**:434-448.

[7] d'Aspremont, A., Bach, F. and El Ghaoui, L. (2008) Optimal solutions for sparse principal component analysis. *Journal of Machine Learning Research* **9**:1269-1294.

[8] Lu, Z.S. and Zhang, Y. (2009) An augmented lagrangian approach for sparse principal component analysis. Mathematical Programming, Series A. DOI: 10.1007/s10107-011-0452-4.

[9] Journée, M., Nesterov, Y., Richtárik, P. and Sepulchre, R. (2010) Generalized power method for sparse principal component analysis. *Journal of Machine Learning Research* **11**:517-553.

[10] Luss, R. and Teboulle, M. (2011) Conditional gradient algorithms for rank-one matrix approximations with a sparsity constraint. *Technical report.*

[11] Meng, D., Zhao, Q. and Xu, Z. (2012) Improve robustness of sparse PCA by L_1-norm maximization. *Pattern Recognition* **45**:487-497.

[12] Richtárik, P. (2011) Finding sparse approximations to extreme eigenvectors: generalized power method for sparse PCA and extensions. *In: Proceedings of Signal Processing with Adaptive Sparse Structured Representations.*

[13] Bah, B., Tanner, J. (2010) Improved bounds on restricted isometry constants for Gaussian matrices. *SIAM Journal on Matrix Analysis and Applications* **31**:2882-2898.

[14] Zhang, Y., El Ghaoui, L. (2011) Large-scale sparse principal component analysis with application to text data. *In: Advances in Neural Information Processing Systems* **24**:532-539.

[15] Intel Math Kernel Library (Intel MKL),
http://software.intel.com/en-us/articles/intel-mkl/

[16] GotoBLAS2, Texas Advanced Computing Center,
http://www.tacc.utexas.edu/tacc-projects/gotoblas2

[17] NVIDIA CUDA Basic Linear Algebra Subroutines (CuBLAS),
http://developer.nvidia.com/cublas

[18] David Newman, Bag of Words Data Set, University of California, Irvine,
http://archive.ics.uci.edu/ml/datasets/Bag+of+Words