On the Convergence of the Dynamic Inner PCA Algorithm

Sungho Shin[†], Alexander D. Smith[†], S. Joe Qin[‡], and Victor Zavala[†*]
[†]University of Wisconsin-Madison Madison, WI 53705, USA.
[‡]University of Southern California, Los Angeles, CA 90089, USA.

Abstract overview

Dynamic inner principal component analysis (DiPCA) is a powerful method for the analysis of time-dependent multivariate data. DiPCA extracts dynamic latent variables that capture the most dominant temporal trends by solving a large-scale, dense, and nonconvex nonlinear program (NLP). A scalable decomposition algorithm has been recently proposed in the literature to solve these challenging NLPs. The decomposition algorithm performs well in practice but its convergence properties are not well understood. In this work, we show that this algorithm is a specialized variant of a coordinate maximization algorithm. This observation allows us to explain why the decomposition algorithm might work (or not) in practice and can guide improvements. We compare the performance of the decomposition strategies with that of the off-the-shelf solver Ipopt. The results show that decomposition is more scalable and, suprisingly, delivers higher quality solutions.

Keywords

PCA, dynamic data modeling, time series analysis, scalable

Motivation and Setting

Principal component analysis (PCA) is a widely used method for dimensionality reduction of *static* multivariate data. PCA identifies latent variables that capture most information (variance) of the original data set. Dynamic inner PCA (DiPCA) is a recently proposed generalization of PCA that is used for dimensionality reduction of *time-dependent* data (Dong and Qin, 2018). DiPCA extracts time series for latent variables that contain most information of the original data set. DiPCA has a key advantage over augmented lagged data-based techniques, such as dynamic PCA and canonical variate analysis (Chiang et al., 2000), in that the extracted dynamic latent variables are easy to interpret (Dong and Qin, 2018). The technique can be used in diverse application areas such as feature extraction, process monitoring, and fault detection.

DiPCA Formulation

We consider time-series data $\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_{n+s} \in \mathbb{R}^m$ (where m is the feature dimension). The data is collected in the matrix $\mathbf{X} \in \mathbb{R}^{(n+s)\times m}$. We consider dynamic latent variables given by $t_i = \mathbf{w}^\top \mathbf{x}_i$ for $i \in \mathbb{I}_{1:n+s}$, where $\mathbf{w} \in \mathbb{R}^m$ is a weight vector for the latent variable subspace and $\mathbb{I}_{1:n+s} := \{1, \cdots, n+s\}$ is a time index set. The latent variables are collected in the vector $\mathbf{t} \in \mathbb{R}^{n+s}$. We assume that the latent variables follow an autoregressive (AR) process of the form:

$$t_i = \beta_1 t_{i-1} + \cdots \beta_s t_{i-s} + r_i, \quad i \in \mathbb{I}_{s+1:n+s}, \tag{1}$$

where $\boldsymbol{\beta} \in \mathbb{R}^s$ is the coefficient vector for the autoregressive model, and $r_i \in \mathbb{R}$ is the residual at time i. We consider all vectors as column vectors and use convention $\mathbf{v} := (v_1, \cdots, v_{n_v})$.

In DiPCA, one aims to find the weights $\mathbf{w}$ and AR coefficients $\boldsymbol{\beta}$ of the autoregressive latent variable model (1) that *maximize the covariance* between the latent variables $t_{s+1}, \cdots, t_{n+s}$ and their corresponding latent model predictions $\hat{t}_{s+1}, \cdots, \hat{t}_{n+s}$, where $\hat{t}_i := \beta_1 t_{i-1} + \cdots + \beta_s t_{i-s}$ for $i \in \mathbb{I}_{s+1:n+s}$. The weights and AR coefficients are found by solving an optimization problem of the form:

$$\max_{\mathbf{w},\boldsymbol{\beta}} \sum_{i=s+1}^{n+s} t_i \hat{t}_i, \quad \text{s.t. } \|\mathbf{w}\|_2^2 \leq 1, \ \|\boldsymbol{\beta}\|_2^2 \leq 1. \tag{2}$$

Here, the norm constraints on $\mathbf{w}$ and $\boldsymbol{\beta}$ are used to avoid arbitrary scaling of the objective. The solution of problem (2) extracts the latent variable space $\mathbf{w}$ that capture the most dynamic variation in the data. With $\mathbf{w}$, a subspace of latent time series that are most predictable from their past data can be obtained. One can extract all the latent time series by *deflating* the data matrix as $\mathbf{X} \leftarrow \mathbf{X} - \mathbf{t}\mathbf{p}^\top$ with $\mathbf{p} := X^\top \mathbf{t}/\mathbf{t}^\top \mathbf{t}$ and by re-solving (2). The last latent time series is the one that contains the least information. The whole set of latent time series can be used to reconstruct the data matrix and a subset can be used to approximate it.

*To whom all correspondence should be addressed

DiPCA Algorithm

The DiPCA problem (2) is a nonconvex nonlinear program (NLP). We now analyze a decomposition algorithm (that we refer to as DiPCA algorithm I) that seeks to find solutions for this NLP. DiPCA algorithm I was proposed by Dong and Qin (2018). We first note that (2) can be expressed in the following equivalent form:

$$\max_{\mathbf{w},\boldsymbol{\beta}} \ \mathbf{w}^\top \mathbf{Y}_{\boldsymbol{\beta}} \mathbf{w} \quad \text{s.t. } \|\mathbf{w}\|_2^2 \leq 1,\ \|\boldsymbol{\beta}\|_2^2 \leq 1, \tag{3}$$

where $\mathbf{Y}_{\boldsymbol{\beta}} := \sum_{i=1}^s \beta_i \mathbf{Y}_i$ and

$$\mathbf{Y}_i := \frac{1}{2}\left(\mathbf{X}_{s+1}^\top \mathbf{X}_{s+1-i} + \mathbf{X}_{s+1-i}^\top \mathbf{X}_{s+1}\right),\ i \in \mathbb{I}_{1:s}$$
$$\mathbf{X}_i := [\mathbf{x}_i\ \cdots\ \mathbf{x}_{i+n-1}]^\top,\ i \in \mathbb{I}_{1:s+1}.$$

The algorithm aims to find a solution of the NLP by solving its first-order optimality conditions. To derive these, we note that the Lagrangian of (3) is:

$$\mathcal{L}(\mathbf{w},\boldsymbol{\beta}) := \mathbf{w}^\top \mathbf{Y}_{\boldsymbol{\beta}} \mathbf{w} - \lambda_w(\|\mathbf{w}\|_2^2 - 1) - \frac{\lambda_{\boldsymbol{\beta}}}{2}(\|\boldsymbol{\beta}\|_2^2 - 1),$$

where $\lambda_{\boldsymbol{\beta}}$ and $\lambda_{\mathbf{w}}$ are Lagrange multipliers. The first-order conditions are:

$$2\mathbf{Y}_{\boldsymbol{\beta}}\mathbf{w} - 2\lambda_w \mathbf{w} = 0,\ \|\mathbf{w}\|_2^2 = 1 \tag{4a}$$
$$\mathbf{w}^\top \mathbf{Y}_i \mathbf{w} - \lambda_{\boldsymbol{\beta}} \beta_i = 0,\ i \in \mathbb{I}_{1:s},\ \|\boldsymbol{\beta}\|_2^2 = 1. \tag{4b}$$

Here, $\|\mathbf{w}\|_2^2, \|\boldsymbol{\beta}\|_2^2 = 1$ follow from the observation that the inequality constraints are always active. Due to nonconvexity, solving (4) as nonlinear equations (e.g., using Newton's method) is computationally challenging. To avoid this, the DiPCA algorithm I uses the iterative scheme:

$$\mathbf{w}^{(\ell+1)} = \mathbf{d}^{(\ell)}/\|\mathbf{d}^{(\ell)}\|_2 \tag{5a}$$
$$\boldsymbol{\beta}^{(\ell+1)} = \mathbf{c}^{(\ell+1)}/\|\mathbf{c}^{(\ell+1)}\|_2, \tag{5b}$$

where ℓ is the iteration counter and

$$\mathbf{d}^{(\ell)} := \mathbf{Y}_{\boldsymbol{\beta}^{(\ell)}} \mathbf{w}^{(\ell)},\ c_i^{(\ell)} := (\mathbf{w}^{(\ell)})^\top \mathbf{Y}_i \mathbf{w}^{(\ell)},\ i \in \mathbb{I}_{1:s}.$$

We observe that (4a) is an eigenvalue problem and that (5a) attempts to approximately solve this (with fixed $\boldsymbol{\beta}^{(\ell)}$). We will see in the next section that (5a) is an iteration of the so-called power method (widely used for the solution of eigenvalue problems and static PCA). We also observe that (5b) exactly solves (4b) (for fixed $\mathbf{w}^{(\ell)}$).

The DiPCA algorithm I has shown to be rather effective at solving the first-order conditions of the NLP but no convergence guarantees have been established. Moreover, it is clear that, due to nonconvexity, the solution of the first-order conditions does not guarantee that a solution is a maximum.

Main Results

We now propose a coordinate maximization algorithm for solving the NLP (3) and show that a simplified variant of this approach is equivalent to the DiPCA algorithm. In coordinate maximization, one partitions the set of decision variables and solves the optimization problem over a subset of variables while fixing the rest, and repeat the procedure for each subset. This approach can be interpreted as a *block Gauss-Seidel* or *alternating maximization* scheme. Coordinate maximization is not guaranteed to converge to a local solution but is often used in applications since fixing a set of variables often reduces complexity and enables deriving closed-form solutions over the complementary variable set. To see this, we partition the decision variables into $\boldsymbol{\beta}$ and $\mathbf{w}$. We consider solving for $\mathbf{w}$ while fixing $\boldsymbol{\beta}$:

$$\mathbf{w}^{(\ell+1)} = \operatorname*{argmax}_{\mathbf{w}} \ \mathbf{w}^\top \mathbf{Y}_{\boldsymbol{\beta}^{(\ell)}} \mathbf{w} \quad \text{s.t. } \|\mathbf{w}\|_2^2 \leq 1. \tag{6a}$$

It is obvious that (6a) is an eigenvalue problem, and such an observaion was also made in (Dong and Qin, 2018, Theorem 1). Next, we consider solving for $\boldsymbol{\beta}$ while fixing $\mathbf{w}$:

$$\boldsymbol{\beta}^{(\ell+1)} = \operatorname*{argmax}_{\boldsymbol{\beta}} \ (\mathbf{c}^{(\ell+1)})^\top \boldsymbol{\beta} \quad \text{s.t. } \|\boldsymbol{\beta}\|_2^2 \leq 1. \tag{6b}$$

We observe that (6b) has a closed-form solution, which is equivalent to (5b). We denote (6) as DiPCA algorithm II. The DiPCA algorithms I and II are summarized in Algorithm 1.

Algorithm 1 Pseudocode for the DiPCA algorithms

$\ell \leftarrow 0$ and $\epsilon \leftarrow +\infty$
$\mathbf{Y}_i \leftarrow (1/2)\left(\mathbf{X}_{s+1}^\top \mathbf{X}_{s+1-i} + \mathbf{X}_{s+1-i}^\top \mathbf{X}_{s+1}\right)$
$\mathbf{y}_i \leftarrow \mathbf{Y}_i \mathbf{w}^{(0)}$ for $i \in \mathbb{I}_{1:s}$ and $\mathbf{d}^{(0)} \leftarrow \sum_{i=1}^s \beta_i^{(0)} \mathbf{y}_i$
while $\epsilon > \epsilon_{\text{tol}}$ **do**
  $\mathbf{w}^{(\ell+1)} \leftarrow \mathbf{d}^{(\ell)}/\|\mathbf{d}^{(\ell)}\|_2$
  $\mathbf{y}_i \leftarrow \mathbf{Y}_i \mathbf{w}^{(\ell+1)}$
  $c_i^{(\ell+1)} \leftarrow (\mathbf{w}^{(\ell+1)})^\top \mathbf{y}_i$ for $i \in \mathbb{I}_{1:s}$
  $\lambda^{(\ell+1)} \leftarrow \|\mathbf{c}^{(\ell+1)}\|_2$
  $^*\boldsymbol{\beta}^{(\ell+1)} \leftarrow \mathbf{c}^{(\ell+1)}/\lambda^{(\ell+1)}$
  $\mathbf{d}^{(\ell+1)} \leftarrow \sum_{i=1}^s \beta_i^{(\ell+1)} \mathbf{y}_i$
  $\epsilon \leftarrow \|\mathbf{d}^{(\ell+1)} - \lambda^{(\ell+1)} \mathbf{w}^{(\ell+1)}\|_\infty$ and $\ell \leftarrow \ell + 1$
end while

* In DiPCA algorithm II, only performed if $\lambda^{(\ell+1)}/\lambda^{(\ell)} - 1 < \epsilon_{\text{tol}}$.

Solving (6a) is equivalent to finding the dominant eigenvector of $\mathbf{Y}_{\boldsymbol{\beta}^{(\ell)}}$. Solving (6a) via an eigenvalue decomposition is computationally inefficient if the feature space m is large. A more scalable approach to find the dominant eigenvector is known as the *power method*. The power iteration for finding the dominant eigenvalue of matrix $\mathbf{A}$ is given by:

$$\mathbf{b}^{(k+1)} = \mathbf{A}\mathbf{b}^{(k)}/\|\mathbf{A}\mathbf{b}^{(k)}\|_2, \quad k = 1, 2, \cdots, \tag{7}$$

where k is the iteration counter. With (7), $\mathbf{b}^{(k)}$ geometrically converges to the dominant eigenvector of $\mathbf{A}$ if $|\lambda_1(\mathbf{A})| > |\lambda_2(\mathbf{A})|$ and $\mathbf{b}^{(0)}$ has a nonzero component of the dominant eigenvector, where $\lambda_1(\cdot)$ and $\lambda_2(\cdot)$ denote the largest and the second largest eigenvalues.

The above derivations reveal connections between DiPCA algorithm I (5) and II (6). One can see that (5b)

and (6b) are identical, but (5a) and (6a) are not. Rather, (5a) performs *one iteration of the power method* (7) with matrix $\mathbf{A} = \mathbf{Y}_{\boldsymbol{\beta}^{(\ell+1)}}$. We can also interpret (5a) as solving (6a) with a *linearized* objective function. This approach is advantageous in computational efficiency since it avoids performing multiple power iterations to solve the eigenvalue problem. On the other hand, performing one iteration of (7) may not guarantee the improvement of the objective value and thus DiPCA algorithm I might face convergence issues, especially when the dominant eigenvalue is negative. Note that DiPCA algorithm I and II do not require performing matrix factorizations.

The above derivations also reveal metrics for monitoring convergence. By evaluating the residual $\mathbf{s}^{(\ell)}$ of (4a) at iteration ℓ with $\lambda^{(\ell)} := (\mathbf{w}^{(\ell)})^\top \mathbf{Y}_{\boldsymbol{\beta}^{(\ell)}} \mathbf{w}^{(\ell)}$, we obtain $\mathbf{s}^{(\ell)} := \mathbf{d}^{(\ell)} - \lambda^{(\ell)} \mathbf{w}^{(\ell)}$. We also note that (4b) is automatically satisfied. Consequently, one can stop the algorithm if $\|\mathbf{s}^{(\ell)}\|_\infty < \epsilon_{\text{tol}}$, for user-defined tolerance ϵ_{tol}.

At a fixed point $(\mathbf{w}, \boldsymbol{\beta})$ of iteration (5), one can show that the first-order conditions (4) hold with $\lambda_{\mathbf{w}} = \lambda_{\boldsymbol{\beta}} = \mathbf{w}^\top \mathbf{Y}_{\boldsymbol{\beta}} \mathbf{w}$. The second-order conditions hold (the fixed point is a maximum point) if the reduced Hessian $\mathbf{Z}^\top \mathbf{H} \mathbf{Z}$ is negative definite; this is equivalent to the condition that the inertia of $\mathbf{K}$ is $(n_+, n_-, n_0) = (2, m+s, 0)$, where

$$\mathbf{K} := \begin{bmatrix} \mathbf{H} & \mathbf{G}^\top \\ \mathbf{G} & \end{bmatrix}, \quad \mathbf{G} := \begin{bmatrix} \mathbf{w}^\top & \\ & \boldsymbol{\beta}^\top \end{bmatrix}, \quad \lambda := \mathbf{w}^\top \mathbf{Y}_{\boldsymbol{\beta}} \mathbf{w}$$

$$\mathbf{H} := \begin{bmatrix} \mathbf{Y}_{\boldsymbol{\beta}} - \lambda \mathbf{I} & \mathbf{Y}_1 \mathbf{w} & \cdots & \mathbf{Y}_s \mathbf{w} \\ \mathbf{w}^\top \mathbf{Y}_1 & -\frac{1}{2}\lambda & & \\ \vdots & & \ddots & \\ \mathbf{w}^\top \mathbf{Y}_s & & & -\frac{1}{2}\lambda \end{bmatrix},$$

and $\mathbf{Z}$ is a null-space basis matrix of $\mathbf{G}$.

Numerical Experiments

We compare the performance of the DiPCA algorithm I and II with that of the off-the-shelf nonlinear programming solver Ipopt (Wächter and Biegler, 2006). Our benchmarks consist of 20 time series obtained from data for a chemical sensor with $m = 5106$, $n = 71$, $s = 4$ (Cao et al., 2018). We add artificial noise to the data with iid Gaussian random variables $N(0, \sigma^2)$ and use $\epsilon_{\text{tol}} = 10^{-6}$. The code is implemented in Julia and run on a Intel(R) Xeon(R) CPU E5-2698 v3 @ 2.30GHz. For Ipopt, we solve the NLP (2) in the space of $\mathbf{t}, \boldsymbol{\beta}, \mathbf{w}$.

The results are shown in the form of cumulative plots for objective value, computational time, and the fraction of negative eigenvalues among the eigenvalues of the reduced Hessian (Figure 1-2). One can see that the performance of DiPCA algorithm I and II is similar. When the noise is small ($\sigma = 1$), we can see that the DiPCA algorithms significantly outperform Ipopt in terms of computation time, but their performance is very similar in terms of objective values. When the noise is large ($\sigma = 10$), the computation time drastically increases for both approaches

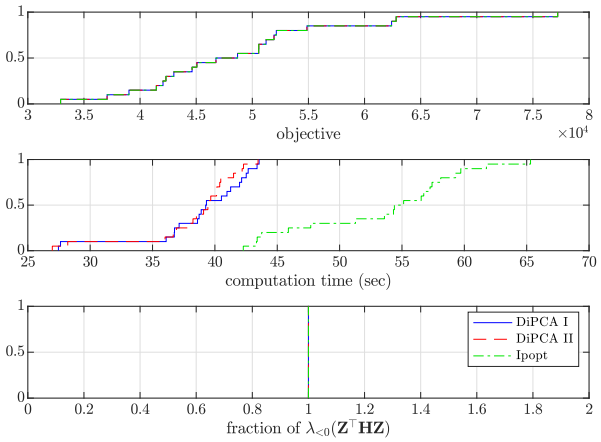

Figure 1: Benchmark results with $\sigma = 1$.

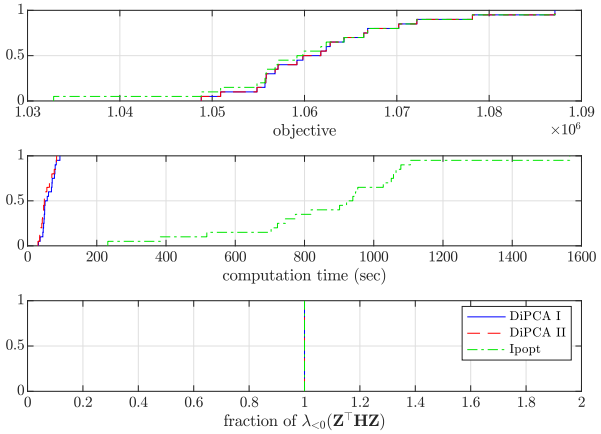

Figure 2: Benchmark results with $\sigma = 10$ (right).

(DiPCA and Ipopt), but DiPCA is significantly faster. The superior efficiency of DiPCA is due to the fact that Ipopt needs to perform matrix factorizations (while DiPCA algorithms does not). We have found that high noise adversely affects the conditioning of the problem (matrix $\mathbf{Y}_{\boldsymbol{\beta}}$) and this slows down the convergence (the power iteration becomes less efficient). In the high noise case, we also found that the DiPCA algorithms find better solutions (compared to Ipopt) in terms of objective values. This seems to indicate that coordinate maximization handles nonconvexity better.

References

Cao, Y., Yu, H., Abbott, N. L., and Zavala, V. M. (2018). Machine learning algorithms for liquid crystal-based sensors. *ACS sensors*, 3(11):2237–2245.

Chiang, L. H., Russell, E. L., and Braatz, R. D. (2000). *Fault detection and diagnosis in industrial systems*. Springer Science & Business Media.

Dong, Y. and Qin, S. J. (2018). A novel dynamic pca algorithm for dynamic data modeling and process monitoring. *Journal of Process Control*, 67:1–11.

Wächter, A. and Biegler, L. T. (2006). On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical programming*, 106(1):25–57.