AN IMPROVED FULLY PARALLEL STOCHASTIC GRADIENT ALGORITHM FOR SUBSPACE TRACKING

Jeroen Dehaene
Harvard University, Pierce Hall, Cruft lab 311, 29 Oxford street, Cambridge MA 02138, U.S.A.
email : jeroen@arcadia.harvard.edu

Marc Moonen, Joos Vandewalle[*]
Katholieke Universiteit Leuven, E.E Dept. (ESAT), K. Mercierlaan 94, 3001 Leuven - Belgium
email: marc.moonen@esat.kuleuven.ac.be URL: http://www.esat.kuleuven.ac.be/ moonen/

ABSTRACT

A new algorithm is presented for principal component analysis and subspace tracking, which improves upon classical stochastic gradient based algorithms (SGA) as well as several other related algorithms that have been presented in the literature. The new algorithm is based on and inherits its main properties from a continuous-time algorithm, closely related to the QR flow. It gives the same estimates as classical SGA algorithms but requires only $\mathcal{O}(N\cdot\kappa)$ operations per update instead of $\mathcal{O}(N\cdot\kappa^2)$, where N is the dimension of the input vector and κ is the number of principal components to be estimated. A parallel version with $\mathcal{O}(\kappa)$ parallelism (processors) and throughput $\mathcal{O}(N^{-1})$ and is straightforwardly derived. A fully parallel version, with throughput independent of the problem size ($\mathcal{O}(1)$), may be obtained at the expense of $\mathcal{O}(N^2)$ additional operations.

1 INTRODUCTION

Subspace estimation, is a well known problem in signal processing, with a broad spectrum of applications, such as beamforming, adaptive noise cancellation, direction of arrival estimation and frequency estimation. We refer to [6, 12, 8, 10, 2, 14] and references therein.

We derive a new parallelizable algorithm for subspace tracking and principal component analysis. The algorithm is based on and inherits its main properties from a continuous-time algorithm (an ODE), which is closely related to the QR flow and which has been analyzed in [3, Ch. 7]. We refer to the continuous-time algorithm as algorithm C and to the discrete-time algorithm as algorithm D.

Algorithm D improves on closely related algorithms, like classical stochastic gradient ascent algorithms (SGA, see for instance [13]), Oja's neural stochastic gradient ascent algorithm (NSGA) [10], the spherical subspace tracking algorithm (SST) [2] and the projection subspace algorithm (PAST) [14]. The new algorithm gives the same estimates as classical SGA algorithms but requires only $\mathcal{O}(N\cdot\kappa)$ operations per update instead of $\mathcal{O}(N\cdot\kappa^2)$, where N is the dimension of the input vector and κ is the number of principal components to be estimated. Unlike other $\mathcal{O}(N\cdot\kappa)$ algorithms the algorithm estimates the κ first principal components, and automatically converges to a set of orthogonal vectors from any full rank initial condition under mild assumptions for the input signal.

A parallel version with $\mathcal{O}(\kappa)$ parallelism (processors) and throughput $\mathcal{O}(N^{-1})$ is straightforwardly derived. A fully parallel version, with throughput independent of the problem size ($\mathcal{O}(1)$), may be obtained at the expense of $\mathcal{O}(N^2)$ additional operations.

2 SUBSPACE TRACKING

Let $\mathbf{x}_{[k]} \in R^N$ be a vector of given input signals at time k ($k = 1,2,\ldots$), generated by

$$\mathbf{x}_{[k]} = M_{[k]}\cdot\mathbf{s}_{[k]} + \mathbf{n}_{[k]}$$

where $M_{[k]}$ is an $N\times\kappa$ full column rank matrix, $\mathbf{s}_{[k]} \in R^\kappa$ ($\kappa < N$) is the source signal, with nonsingular correlation matrix $E\{\mathbf{s}_{[k]}\cdot\mathbf{s}_{[k]}^T\}$ and finally $\mathbf{n}_{[k]}$ is the noise signal with mostly $E\{\mathbf{n}_{[k]}\cdot\mathbf{n}_{[k]}^T\} = \sigma_N^2 I$. The column space of $M_{[k]}$ is called the *signal subspace*, and its orthogonal complement is called the *noise subspace*. The *subspace tracking problem* consists in estimating the signal subspace at each time k, given $\mathbf{x}_{[k]}$.

A simple stochastic gradient algorithm is given as follows, where the columns of $A_{[k]}$ give an estimate of the signal subspace at each time k.

Initialize $A_{[0]} \Leftarrow A_o$
for $k = 1,\ldots,\infty$

Step 1 : Time-update

$$\tilde{A}_{[k]} \Leftarrow A_{[k-1]} + \alpha\cdot\mathbf{x}_{[k]}\cdot\mathbf{x}_{[k]}^T\cdot A_{[k-1]}$$

Step 2 : Re-orthogonalization

$$\tilde{A}_{[k]} = Q\cdot R \quad \text{(QR-factorization)}$$

$$A_{[k]} \Leftarrow Q$$

end

[*]Marc Moonen is a research associate with the Belgian N.F.W.O. (National Fund for Scientific Research). This research was supported by the N.F.W.O. project G.0292.95, and also carried out in the framework of a Concerted Action Project of the Flemish Community, entitled "Model-based Information Processing Systems". The scientific responsibility is assumed by its authors.

Here α is a step-size parameter. For the derivation of this algorithm, we refer to, *e.g.*, [13]. The algorithm has several desirable properties, *e.g.*, in a stationary set-up the columns of A converge to the eigenvectors of M (eigenvector corresponding to the largest eigenvalue in the first column, *etc.*).
The operation count of the first step is $\mathcal{O}(N \cdot \kappa)$. The operation count of the second step, however, is $\mathcal{O}(N \cdot \kappa^2)$. The second step also complicates (disallows) pipelined processing. Therefore, in this paper, it is suggested to perform the reorthogonalization of step 2 in a different fashion.

3 ALGORITHM D

It is readily proved (by exploiting the orthogonality of $A_{[k-1]}$) that the 'rank-one updated orthogonal matrix' $\tilde{A}_{[k]}$ can be reorthogonalized in a cheaper fashion, namely with only $\mathcal{O}(N \cdot \kappa)$ operations. The improved algorithm is as follows

for $k = 1, \ldots, \infty$

Step 1 : Time-update

$$\tilde{A}_{[k]} \Leftarrow A_{[k-1]} + \alpha \cdot \mathbf{x}_{[k]} \cdot \underbrace{\mathbf{x}_{[k]}^T \cdot A_{[k-1]}}_{\mathbf{y}_{[k]}^T}$$

Step 2 : Re-orthogonalization

$$\beta = (2\alpha + \alpha^2 \mathbf{x}_{[k]}^T \mathbf{x}_{[k]})^{-\frac{1}{2}}$$

$$\left[\begin{array}{c|c} A_{[k]} & \star \\ \hline 0 & \delta \end{array}\right] \Leftarrow \left[\begin{array}{c|c} \tilde{A}_{[k]} & 0 \\ \hline -\mathbf{y}_{[k]} & \beta \end{array}\right] \cdot \tilde{Q}$$

end

Here $\tilde{Q}$ is an orthogonal matrix that zeroes the elements of the last row in the compound matrix, *i.e.* $\tilde{Q}$ is such that

$$\left[\begin{array}{c|c} 0 & \delta \end{array}\right] \Leftarrow \left[\begin{array}{c|c} -\mathbf{y}_{[k]} & \beta \end{array}\right] \cdot \tilde{Q}$$

and which may, *i.e.*, be constructed as a sequence of Givens transformations [7]. The Givens transformations zero the elements of $\mathbf{y}_{[k]}$, one after the other, by combining them with the rightmost element (β). The orthogonalized matrix $A_{[k]}$ is then obtained by applying the same transformation $\tilde{Q}$ to the $\tilde{A}_{[k]}$, together with an all-zeroes column.

Suffice it to say that this algorithm delivers exactly the same results as the original algorithm when A_o is orthogonal. If A_o is not orthogonal, the orthogonalization is 'spread out' over time, *i.e.* $A_{[k]}$ converges to the set of orthogonal matrices.

Finally, it is seen that the operation count is $\mathcal{O}(N \cdot \kappa)$ per update. For a further analysis of this algorithm, we refer to [4] and the full paper [5].

4 PIPELINED ALGORITHM

The improved algorithm allows a fully pipelined implementation. A signal flow graph (SFG) of the algorithm is given in **Figure 1**. Black squares represent memory cells (delay elements), storing the elements of $A_{[k]}^T$, as indicated (the matrix is stored in transposed form for convenience). The white rectangles correspond to multiply-add cells. The matrix-vector product

$$\mathbf{y}_{[k]} = A_{[k]}^T \mathbf{x}_{[k]}$$

(*cfr.* step 1) is accumulated from right to left. The vector $\mathbf{y}_{[k]}$ thus becomes available at the left-hand side of the array, as indicated. The time-update

$$\tilde{A}_{[k]} \Leftarrow A_{[k-1]} + \alpha \cdot \mathbf{x}_{[k]} \cdot \mathbf{y}_{[k]}^T$$

is then performed from left to right, by means of the second series of multiply-add cells. Finally, the white hexagons represent orthogonal transformations (Givens transformations), with functionality

$$\left[\begin{array}{c} a' \\ b' \end{array}\right] = \left[\begin{array}{cc} \cos\phi & \sin\phi \\ -\sin\phi & \cos\phi \end{array}\right] \cdot \left[\begin{array}{c} a \\ b \end{array}\right]$$

. These transformations are initiated (computation of the ϕ's) at the left-hand side (by means of the available $\mathbf{y}_{[k]}^T$ and the β that is fed in from the top) and then propagated to the right (*cfr.* the orthogonal update in step 2).

Pipeline delays may be introduced on the top-to-bottom connections, which straightforwardly leads to an $\mathcal{O}(N^{-1})$ throughput implementation with a linear array of κ processors (one processor for each row).

In the horizontal direction, the SFG exhibits feedback paths *i.e.* critical paths of length up to $2N$, which disallow pipelined processsing. However, it can be shown that suitable delays may be introduced to the SFG by performing algorithmic transformations which compensate for the interference of crossing data flows and thus eliminate the critical feedback loops. The algorithmic transformations used here are similar to those used in [9, 11] to obtain a fully pipelined recursive least squares algorithm. An intermediate result is shown in **Figure 2**, where pipeline delays have been introduced on all left-to-right connections, in between two columns, and algorithmic corrections have been added accordingly. The required algorithmic correction corresponds to simply applying the updating transformations (time-update and reorthogonalization, *i.e.* multiply-adds and rotations, propagated from left to right) to the intermediate results in the computation of $\mathbf{y}_{[k]}$ (accumulated from right to left) together with a vector-vector product which is computed in the upper part of the signal flow graph. For a detailed derivation of the required algorithmic correction, we refer to the full paper [5].

Figure 3 shows the resulting signal flow graph, when pipeline delays are introduced between any two adjacent columns. By applying 2-slowing and retiming, one can now easily obtain a *fully pipelined* $\mathcal{O}(N^0)$ *throughput array*, with a main array of $\kappa \times N$ processors and an additional $N \times N$ triangular array of (simpler) multiply-add cells.

References

[1] R.D. DeGroat, "Noniterative subspace updating," *IEEE Transaction on Signal Processing*, vol. 40, no. 3, pp. 571–577, 1992.

[2] R.D. DeGroat and E.M. Dowling, "Sphericalized subspace tracking: convergence and detection schemes," in *Proceedings of the 26th annual Asilomar conference*, pp. 561–565, 1992.

[3] J. Dehaene, *Continuous-time matrix algorithms, systolic algorithms and adaptive neural networks*. PhD thesis, Katholieke Universiteit Leuven, October 1995. ftp.esat.kuleuven.ac.be/pub/SISTA/dehaene/phd/

[4] J. Dehaene, M. Moonen, J. Vandewalle, "A new analysis of a class of continuous-time algorithms for principal component analysis and subspace tracking," K.U. Leuven, ESAT-SISTA report 1995-48I. Submitted for publication. http://www.esat.kuleuven.ac.be/ moonen/publications.html.

[5] J. Dehaene, M. Moonen, J. Vandewalle. "An improved stochastic gradient algorithm for principal component analysis and subspace tracking," K.U. Leuven, ESAT-SISTA report 1995-49I. Submitted for publication. http://www.esat.kuleuven.ac.be/ moonen/publications.html.

[6] E.F. Deprettere, ed., *SVD and Signal Processing, Algorithms, Applications and Architectures*. Norht-Holland, 1988.

[7] G. Golub and C.F. Van Loan, *Matrix computations*. Johns Hopkins University Press, 1991.

[8] M. Moonen and B. De Moor, eds., *SVD and Signal Processing III, Algorithms, Applications and Architectures*. Elsevier, 1995.

[9] M. Moonen and J.G. McWhirter, "A systolic array for recursive least squares by inverse updating," *Electronics Letters*, vol. 29, no. 13, pp. 1217–1218, 1993.

[10] E. Oja, "Principal components, minor components, and linear neural networks," *Neural Networks*, vol. 5, pp. 927–935, 1992.

[11] I.K. Proudler, J.G. McWhirter, M. Moonen, and G. Hekstra, "The formal derivation of a systolic array for recursive least squares estimation," *IEEE Transactions on Circuits and Systems-II: Analog and Digital Signal Processing*, Vol. 43, Nr. 3, March 1996, pp 247-254.

[12] R.J. Vaccaro, ed., *SVD and Signal Processing, II, Algorithms, Applications, and Architectures*. Elsevier, 1991.

[13] B. Yang, "Gradient based subspace tracking algorithms and systolic implementation," *International Jounal of High Speed Electronics and Systems*, vol. 4, no. 2, 1993.

[14] B. Yang, "Projection approximation subspace tracking," *IEEE Transactions on Signal Processing*, vol. 43, no. 1, pp. 95–107, January 1995.

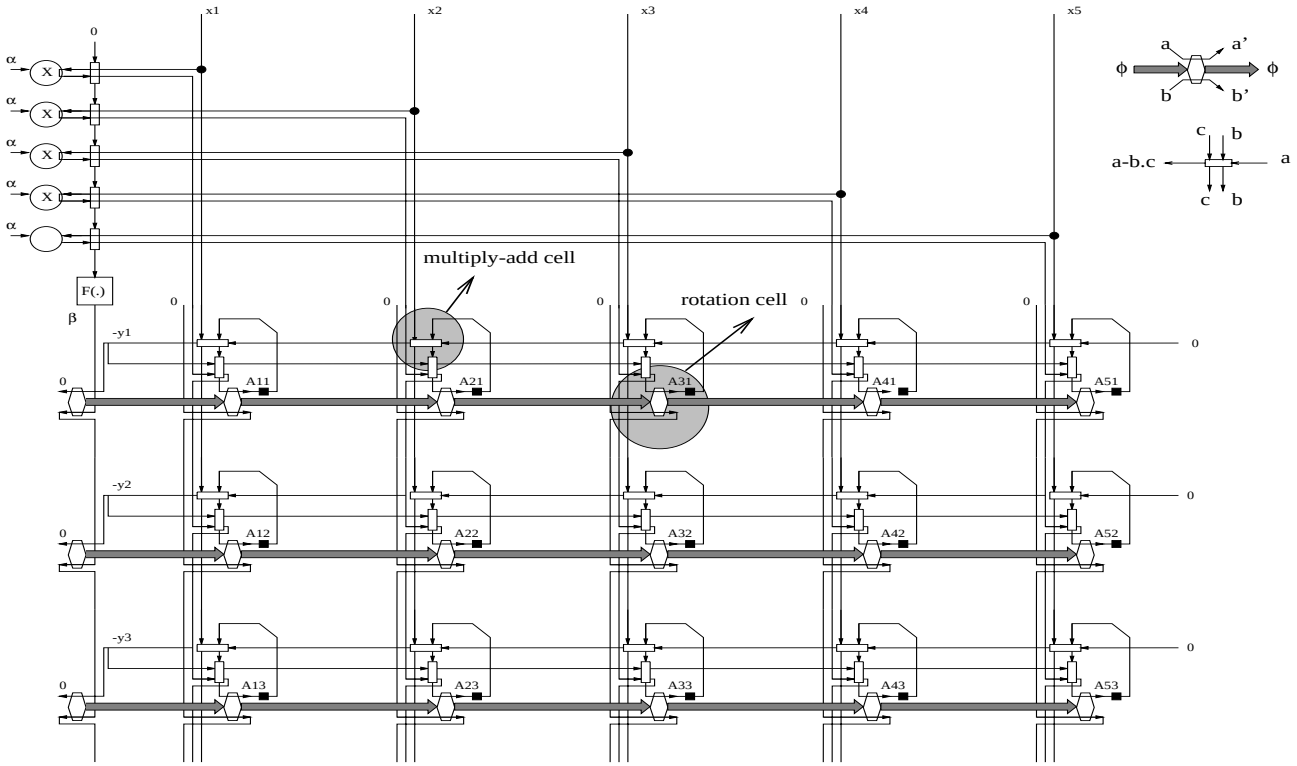

Figure 1 : Signal flow graph for subspace tracking

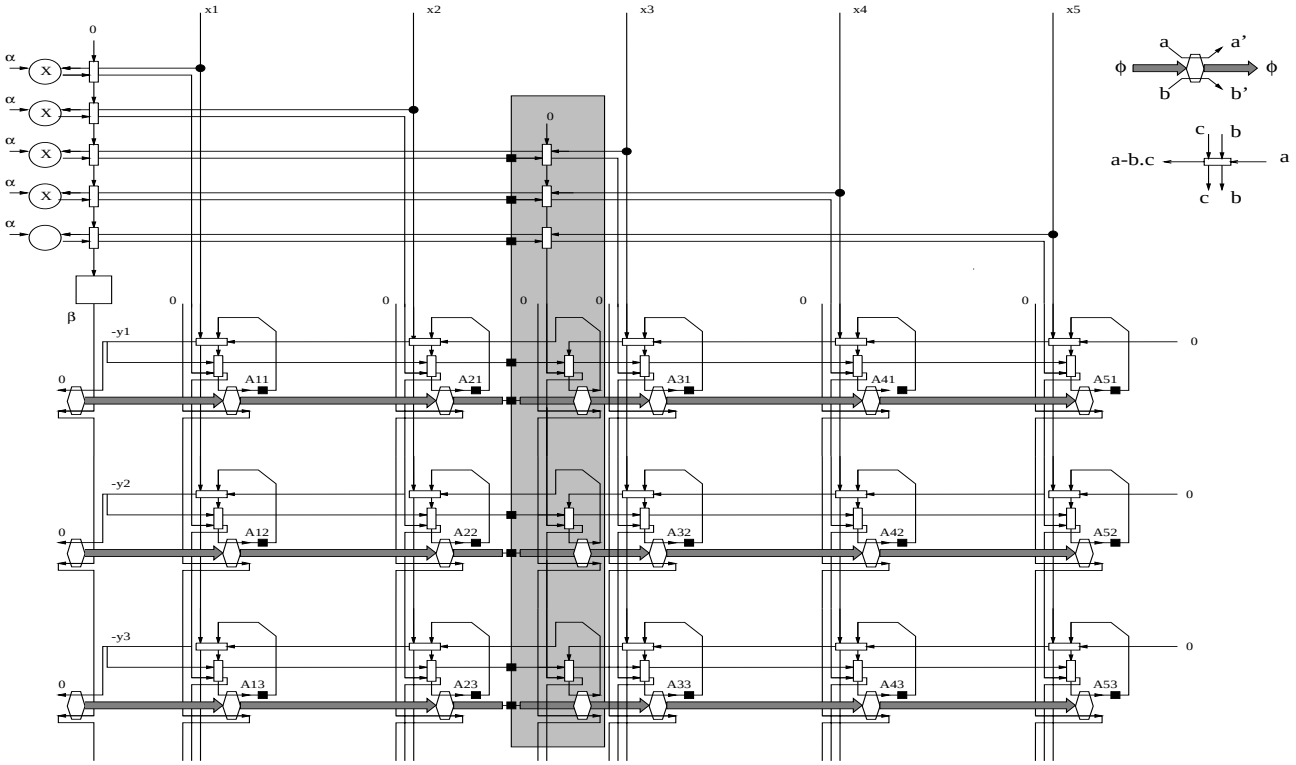

Figure 2 : Introducing pipeline delays

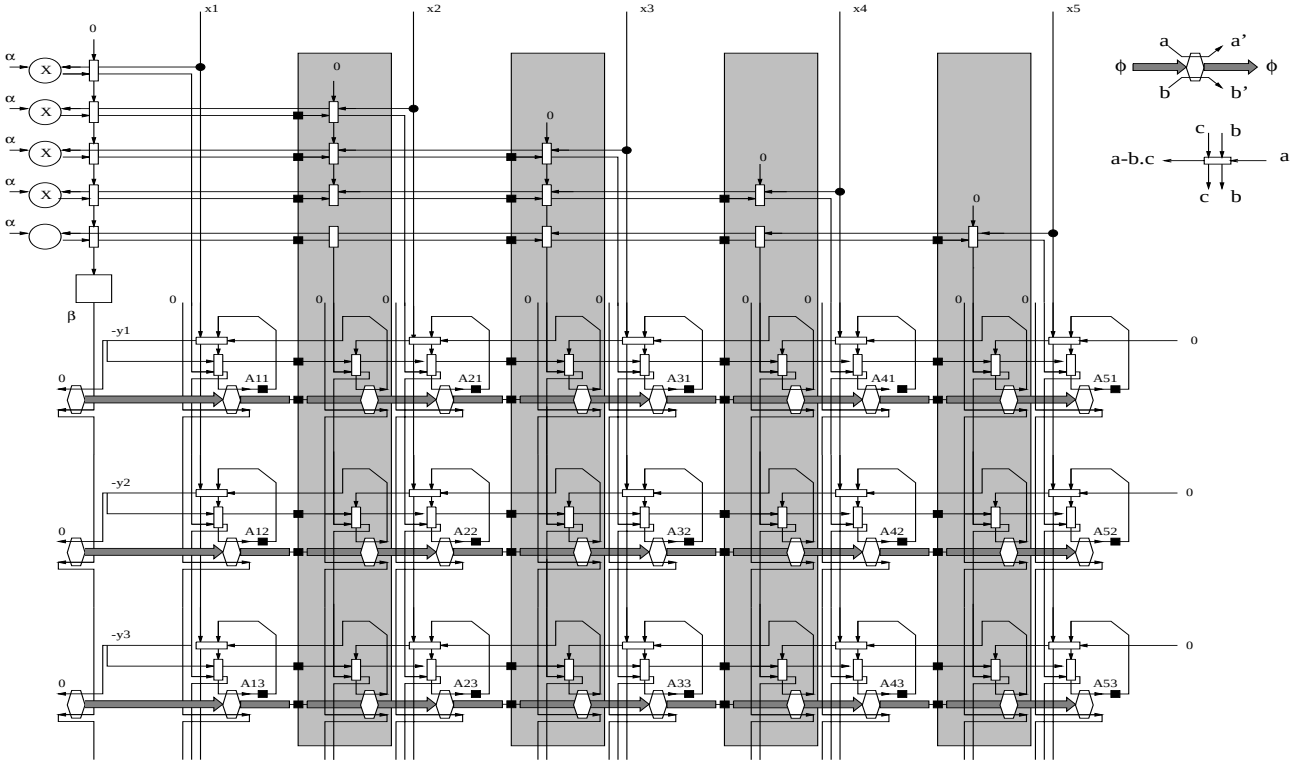

Figure 3 : Signal flow graph with pipeline delays