Streaming Algorithms for Subspace Analysis: an Edge- and Hardware-oriented review

Alex Marchioni, *Student Member, IEEE*, Luciano Prono, *Student Member, IEEE* Mauro Mangia, *Member, IEEE*, Fabio Pareschi, *Senior Member, IEEE*, Riccardo Rovatti, *Fellow, IEEE*, and Gianluca Setti, *Fellow, IEEE*

Abstract—Subspace analysis is a basic tool for coping with high-dimensional data and is becoming a fundamental step in early processing of many signals elaboration tasks.

Nowadays trend of collecting huge quantities of usually very redundant data by means of decentralized systems suggests these techniques be deployed as close as possible to the data sources. Regrettably, despite its conceptual simplicity, subspace analysis is ultimately equivalent to eigenspace computation and brings along non-negligible computational and memory requirements.

To make this fit into typical systems operating at the *edge*, specialized *streaming* algorithms have been recently devised that we here classify and review giving them a coherent description, highlighting features and analogies, and easing comparisons.

Implementation of these methods is also tested on a common edge digital hardware platform to estimate not only abstract functional and complexity features, but also more practical running times and memory footprints on which compliance with real-world applications hinges.

Index Terms—Principal Components Analysis, Principal Subspace, Minor Component Analysis, Minor Subspace, Streaming Algorithms, Edge computing.

I. INTRODUCTION

THE IDEA behind Subspace Analysis (SA) in its different declinations (principal or minor subspace identification or tracking) is to find a subspace of the original signal space such that the corresponding data sets projection has some special properties.

In principal component/subspace analysis (PCA/PSA) projection is the embedding of the data set into a lower dimensional subspace that maintains most of the original energy, while in minor component/subspace analysis (MCA/MSA) projection should produce always vanishing vectors.

PCA/PSA has hystorically a huge number of applications, among which, for example, pattern identification in computer network traffic analysis [1], as well as anomaly detection [2], biomedical application [3], [4], blind source separation [5], surveillance [6], [7], though the list is by far not exhaustive.

A. Marchioni, M. Mangia and R. Rovatti are with the Department of Electrical, Electronic, and Information Engineering, University of Bologna, 40136 Bologna, Italy, and also with the Advanced Research Center on Electronic Systems, University of Bologna, 40125 Bologna, Italy (e-mail: alex.marchioni@unibo.it, mauro.mangia2@unibo.it, riccardo.rovatti@unibo.it).

L. Prono is with the Department of Electronics and Telecommunications, Politecnico di Torino, 10129 Torino, Italy (e-mail: luciano.prono@polito.it).

F. Pareschi and G. Setti are with the Department of Electronics and Telecommunications, Politecnico di Torino, 10129 Torino, Italy, and also with the Advanced Research Center on Electronic Systems (ARCES), University of Bologna, 40125 Bologna, Italy (e-mail: fabio.pareschi@polito.it, gianluca.setti@polito.it).

Fig. 1. General scheme of a streaming algorithm for subspace analysis.

The most straightforward solution to subspace finding are *batch* methods, such as the eigenvalue decomposition (EVD) or the singular-value decomposition (SVD) techniques [8]. They require to acquire and store large chunks of available data, and process them with algorithms that make use of standard linear algebra computations. For this reason, batch methods imply a large memory usage and have a complexity in the order of at least $\mathcal{O}(n^2k)$, where n is the original dimension of the data and a subspace of dimension k is sought.

Yet, nowadays, identifying and tracking principal and minor subspaces is becoming particularly important due to the increasing availability of massive, but often highly redundant, collections of data. It may become an engineering nonsense to transport and store all this data, when a large part is eventually discarded as a result of information extraction procedures.

For this reason, subspace analysis is one of the best candidates for implementation at the *edge* of the cloud, i.e., those devices that appear along the data path from sensors to central processing facilities. Edge devices typically have hub functionalities as they are in charge of concentrating data coming from multiple physically close sources and send their aggregation throught the Internet.

Hence, they are the first stage of the processing chain in which data may become highly-dimensional while exhibiting a noteworthy level of redundancy and ideal candidates for early processing and dimensionality reduction.

Yet, hubs do not support the ideally unlimited and easily scalable computational and storage resources that characterize the cloud model, and this implies the need for tailored approaches.

In this light, a number of streaming algorithms (see, e.g.

[9]–[11]) have been recently presented. Here, data windows are not stored but processed as they are available, resulting in a complexity in the order of $\mathcal{O}(nk^2)$ or, at the price of some approximation, even $\mathcal{O}(nk)$.

Figure 1 shows the general scheme of streaming algorithms for subspace analysis. Among the many proposals, often differing only for subtle details, our review selects those that incarnate main principles and approaches, are sufficiently lightweight to fit in a low-resources device, and have performances that are representative of what can be obtained even by more specialized implementations. For example, we only consider purely streaming approaches as block-wise methods are usually a straightforward extension of sample-wise methods but require a larger use of memory and computation resources without yielding a significant improvement in subspace identification.

All methods are expressed within the same coherent framework that facilitates comparison in terms of both functionality (estimation accuracy) and resource needs. We also investigate and discuss the deployment of these methods on edge devices distinguishing high-end from low-end devices. The former class of devices usually features an operative system that supports high level programming languages, while the latter class requires direct programming.

The paper is organized as follows. Sec. II proposes methods classification while Sec. III presents a list of streaming algorithms along with their relationship. Methods performance are in Sec. IV and, before Conclusion, Sec. V compares the computational budget and memory footprint on reference edge devices.

II. METHODS CLASSIFICATION

We model the data stream as a discrete-time stochastic process that generates occurrences $\mathbf{x}_t \in \mathbb{R}^n$ with $t = 0, 1, \dots$. The algorithms we describe process vectors $\mathbf{x}_t$ sequentially to extract a characterization of the whole data-set. We assume that $\mathbf{E}[\mathbf{x}_t] = \mathbf{0}_n$ for every t , where $\mathbf{0}_n$ is the n -dimensional null vector and $\mathbf{E}[\cdot]$ indicates the expectation operator. We also assume the covariance/correlation matrix $\Sigma_t = \mathbf{E}[\mathbf{x}_t \mathbf{x}_t^\top]$, where $\cdot^\top$ indicates transposition, is either constant or slowly varying for a long span of subsequent time instants, so that we may safely drop the time indication, i.e., $\Sigma_t \approx \Sigma$. If this constraint holds we can also use $\mathbf{x}$ instead of $\mathbf{x}_t$ to represent a generic signal occurrence.

As for any correlation matrix, Σ has an EVD $\Sigma = \Psi \Lambda \Psi^\top$, Λ is a diagonal matrix with diagonal entries $\lambda_0 \geq \lambda_1 \geq \dots \geq \lambda_{n-1} \geq 0$, and the columns of $\Psi \in \mathbb{R}^{n \times n}$ are the corresponding eigenvectors in orthonormal form.

We aim at identifying the so-called *principal subspace*, i.e., the one spanned by the $m < n$ eigenvectors associated with the largest eigenvalues, or the so-called *minor subspace* or *noise subspace*, i.e., the one spanned by the remaining $n - m$ eigenvectors associated with the smallest eigenvalues.

In any case, the target subspace is expressed as the span of the columns of a matrix $\mathbf{U} \in \mathbb{R}^{n \times k}$ that is column-orthonormal, i.e., such that $\mathbf{U}^\top \mathbf{U} = \mathbf{I}_k$, where $\mathbf{I}_k$ stands for the $k \times k$ identity matrix.

For the principal subspace estimation $k = m$ while $k = n - m$ is in case of minor subspace. In several applications $m \ll n$ [12] such that the size of $\mathbf{U}$ drastically changes in passing to the task identifying the minor subspace from the one that focuses on the principal subspace.

Streaming algorithms are iterative. They maintain an estimation $\mathbf{U}_t$ that is updated every time a new vector $\mathbf{x}_t$ is acquired in such a way that $\mathbf{U}_t \rightarrow \mathbf{U}$ when t grows. The streaming algorithms we review reach this goals in different ways. Here we list some key feature that can be used to make distinctions or draw connections.

1) *principal and minor subspaces*: most algorithms are able to target principal subspaces. Some of them are also able to target minor subspaces and few methods are designed for the minor subspace only.

2) *eigenvectors or subspaces*: principal and minor subspaces may be identified by computing the eigenvectors $\Psi_{|m} = [\psi_0, \dots, \psi_{m-1}]$ (principal components) and the eigenvectors $\Psi_{|n-m} = [\psi_{n-m}, \dots, \psi_{n-1}]$ (minor components) or, alternatively, by giving any other basis of their spans.

3) *objective function*: identifying the k -dimensional principal (minor) subspace is equivalent to finding the column-orthonormal matrix $\mathbf{U} \in \mathbb{R}^{n \times k}$ that maximizes (minimizes) the variance of the projection $\mathbf{y} = \mathbf{U}^\top \mathbf{x}$ [13], [14]. Hence, some approaches consider the objective function

$$J_{\text{Var}}(\mathbf{U}) = \mathbf{E}[\|\mathbf{U}^\top \mathbf{x}\|^2] = \text{tr}(\mathbf{U}^\top \Sigma \mathbf{U}) \quad (1)$$

where $\|\cdot\|$ denotes the l_2 norm, $\text{tr}(\cdot)$ computes the trace of its argument, and where $\mathbf{U}$ is constrained to be column-orthonormal.

Maximizing (minimizing) projection variance is equivalent to minimizing (maximizing) the average norm of residual vector $\mathbf{r} = \mathbf{x} - \mathbf{U}\mathbf{U}^\top \mathbf{x}$. Hence, some methods [15], [16] consider the objective function

$$J_{\text{MSE}}(\mathbf{U}) = \mathbf{E}[\|\mathbf{x} - \mathbf{U}\mathbf{U}^\top \mathbf{x}\|^2] \quad (2)$$

As a further alternative, it is possible to define an objective function based on the so-called *spiked model* [17] whereby the observable $\mathbf{x}$ is assumed as an expansion of an m -dimensional signal $\mathbf{s}$ such that $\mathbf{s} = \Psi_{|m}^\top \mathbf{x}$ [18], [19]. Despite their obvious link, some methods consider $\mathbf{x}$ and $\mathbf{s}$ separately and minimize (3) to enforce their relationship.

$$J_{\text{SM}}(\mathbf{U}) = \mathbf{E}[\|\mathbf{x} - \mathbf{U}\mathbf{s}\|^2] \quad (3)$$

As a remark, the signal model behind (3) make this objective function effective only in case of principal subspace identification, i.e., $J_{\text{SM}}(\mathbf{U}) = 0$ implies $\text{span}(\mathbf{U}) = \text{span}(\Psi_{|m})$.

Only one of the algorithms we review does not use optimization to identify the subspace. Its description is in Section III.

4) *column-orthonormality*: Though the final $\mathbf{U}$ must be orthonormal, the orthonormality of each $\mathbf{U}_t$ depends on the algorithm. Two main classes exist. The one containing procedures that ensure every $\mathbf{U}_t$ to be orthonormal, and the one containing procedures that produce $\mathbf{U}_t$ that is only approximately orthonormal.

Fig. 2. Classification of methods and their relationships. It is possible to define two main families, methods based on the stochastic gradient and methods properly designed for signals with intrinsic dimensionality equal to k (spiked model). Connection between methods are also reported in according to their description: *a* - [21] discusses the relationships between both methods showing that (8) it is possible to approximate the Oja's update as the one proposed by Krasulina without the orthonormalization; *b* & *c* - [22]–[24] present methods derived from (8) that has been further extended in [25] to HFRANS; *d* - [26] shows that the application of the ISVD to the missing data case is equivalent to a suitable configuration of GROUSE.

In general, algorithms ensuring orthonormality have complexity $\mathcal{O}(nk^2)$, while algorithms limiting to approximate orthonormality are able to achieve $\mathcal{O}(nk)$.

Those ensuring orthonormality can be further distinguished depending on the technique used. Some do so by applying a specific orthonormalization procedure (typically a QR-decomposition) at each step. In this case, if the update produces a matrix U'_t that is not column-orthonormal, then the columns of the final U_t span the same subspace as the columns of U'_t , while being orthonormal.

Some others do so by projecting onto the Grassmannian manifold [14], [20] that contains all possible $n \times k$ orthonormal matrices. Using projection implies that the span of the result of the non-orthonormal update U'_t is not necessarily the same as that of the final orthonormal U_t , and some of the improvements in selecting U'_t might be lost.

As a third option, orthonormality can be guaranteed by constraining the updates of U_t to be along Grassmannian geodesics. In this case, the updated estimation improves over the previous one without leaving the acceptability region.

Tab. I classifies the methods, whose detailed description is given in the next Section, along directions from 1) to 4) above. In the same Table, we also report the computational complexity and the field of interest in which the methods were originally conceived.

III. METHODS DESCRIPTION

This section describes the methods we then test and implement. For each method we give the *update step*, i.e., the sequence of operations leading from the matrix U_{t-1} that is the estimate of the subspace after $t-1$ input observations, to the matrix U_t that is the current estimation and takes into account also the occurrence associated to x_t .

Most update steps compute intermediate quantities that have a proper role, namely $y_t = U_{t-1}^\top x_t$ that is the vector of coefficients expressing the projection of x_t onto the subspaces as it is estimated at time $t-1$, and $r_t = x_t - U_{t-1}y_t$ that is the residual of such a projection. Note that if U_{t-1} is orthonormal, the two vectors y_t and r_t are orthogonal.

For the sake of brevity, the computation of y_t and r_t are not explicitly mentioned in the descriptions of the methods.

A. Oja

Originally proposed in [27] for the principal subspace estimation with $k = m = 1$, and then extended to the rank- k cases in [28] it starts from a random column-orthonormal $U_0 \in \mathbb{R}^{n \times k}$. At the t -th sample it updates the estimation of $U = \Psi|_k$ according to the input data x_t as

$$U_t = \Omega(U_{t-1} + \gamma_t x_t x_t^\top U_{t-1}) \quad (4)$$

where $\Omega(\cdot)$ is an operator that orthonormalizes the columns of its argument, e.g., gives the Q matrix in the QR decomposition of its argument [29, Chapter 2]. The parameter γ_t is the step size or learning rate that may change with time.

Notably, authors of [15], [30], [31] show that Oja is an extension of the well-know power method [32] that, in turn, is equivalent to solving a maximization problem where the objective function is (1) and U_t is constrained to be column-orthonormal. In particular, the gradient of (1) with respect to U is ΣU such that (4) is equivalent to the update of a stochastic gradient descent algorithm where Σ is approximated by $x_t x_t^\top$, γ_t is the learning rate, and $\Omega(\cdot)$ forces the update to yield a column-orthonormal matrix.

Actually, to save some computation, one may think of applying Ω only after a certain number of updates [15]. Yet, proper sizing of the number of steps without orthonormalization depends on the application.

Furthermore, when minimization is considered instead of maximization, with same objective function and constraint, one may aim at identifying the minor subspace $U = \Psi_{k|}$ with $k = n-m$. With respect to (4), here stochastic gradient descent algorithm performing minimization follows the opposite of the gradient of (1), i.e.,

$$U_t = \Omega(U_{t-1} - \gamma_t x_t x_t^\top U_{t-1}) \quad (5)$$

Lastly, since the convergence of the method depends on the choice of the initial matrix U_0 , [33] proposes a procedure of warm start that avoids the random initialization.

B. Krasulina

Originally proposed in [34], [35] and recently revised in [13] to include the $k > 1$ case, it also starts from a random column-orthonormal $U_0 \in \mathbb{R}^{n \times k}$. The update step is

$$U_t = \Omega(U_{t-1} + \gamma_t r_t y_t^\top) \quad (6)$$

According to [13], [16], the update in (6) converges to a matrix that approximates the minimizer of (2).

As for Oja, a warm start approach has been proposed in [13] and the minor subspace can be targeted by simply change sign of the last equation in (6) to yield

$$\mathbf{U}_t = \Omega(\mathbf{U}_{t-1} - \gamma_t \mathbf{r}_t \mathbf{y}_t^\top) \quad (7)$$

In this case the method converge to $\mathbf{U} = \Psi_{|k|}$ with $k = n - m$.

Oja and Krasulina are deeply linked. In fact, [21] proves that the

$$\Omega(\mathbf{U}_{t-1} + \gamma_t \mathbf{x}_t \mathbf{x}_t^\top \mathbf{U}_{t-1}) = \mathbf{U}_{t-1} + \gamma_t \mathbf{r}_t \mathbf{y}_t^\top + o(\gamma_t^2) \quad (8)$$

thus ultimately establishing equivalent convergence properties for (4) and (6) as $\gamma_t \rightarrow 0$ for growing t .

C. HFRANS

The analogy of the two previous methods highlighted by (8) has led to class of methods that try to provide column-orthonormality avoiding the Ω operator.

Since $\mathbf{U}_t = \mathbf{U}_{t-1} + \gamma_t \mathbf{r}_t \mathbf{y}_t^\top$ tends to be column-orthonormal for $t \rightarrow \infty$ an extreme option is to consider $o(\gamma_t^2)$ negligible and simply avoid orthonormalization.

Such a choice can be acceptable when targeting $\mathbf{U} = \Psi_{|k|}$. Yet, when aiming at identifying $\mathbf{U} = \Psi_{k|}$, the very small amplitude of the projection of the signal on the minor subspace makes the whole procedure extremely error-prone and may spoil convergence.

To overcome this impasse, [22] first proposed a term to approximate the $o(\gamma_t^2)$ residue in (8) giving raise to the so-called OOja method that is then extended [23] with a policy that adapts γ_t to both $\mathbf{U}_t$ and $\mathbf{x}_t$, leading to the so-called NOOja method.

OOja and NOOja further evolved [25] into HFRANS. This method is also an adjustment of FRANS, a previous Rayleigh quotient-based adaptive noise subspace method [24]. In HFRANS, Householder transformations are introduced to grant the numerical stability needed to the cope with the minor subspace case.

The method starts from a random $\mathbf{U}_0 \in \mathbb{R}^{n \times k}$ matrix and uses the following update rule that depends on a given $0 < \gamma < 2$,

$$\begin{aligned} \tau_t &= \frac{1}{\|\mathbf{y}_t\|^2} \left[\left(1 - (2 - \gamma) \gamma \frac{\|\mathbf{y}_t\|^2}{\|\mathbf{x}_t\|^2} \right)^{-\frac{1}{2}} - 1 \right] \\ \hat{\mathbf{u}}_t &= (1 + \tau_t \|\mathbf{y}_t\|^2) \mathbf{x}_t - \frac{\tau_t \|\mathbf{x}_t\|^2}{\gamma} \mathbf{U}_{t-1} \mathbf{y}_t \\ \mathbf{u}_t &= \frac{\hat{\mathbf{u}}_t}{\|\hat{\mathbf{u}}_t\|} \\ \mathbf{U}_t &= \mathbf{U}_{t-1} - 2 \mathbf{u}_t \mathbf{u}_t^\top \mathbf{U}_{t-1} \end{aligned} \quad (9)$$

D. PAST

The Project Approximation Subspace Tracking (PAST) [16], [36] is an algorithm obtained by minimizing (3) in which the expectation is unrolled in time as an exponentially weighted sum, i.e., by setting

$$\mathbf{U}_t = \arg \min_{\mathbf{U} \in \mathbb{R}^{n \times k}} \sum_{l=1}^t \beta^{t-l} \|\mathbf{x}_l - \mathbf{U} \mathbf{s}_l\|^2 \quad (10)$$

without the constraint of $\mathbf{U}_t$ being column-orthonormal and where $\beta \in [0, 1]$ is the forgetting factor that weights the prior samples. Equation (10) is based on the fact that signal observances $\mathbf{x}_l$ are generated accordingly to a spiked model, i.e., $\mathbf{s}_l = \Psi_{|k|}^\top \mathbf{x}_l$.

Since, at each signal occurrence, only $\mathbf{x}_l$ is known, $\mathbf{s}_l$ is approximated by the projection vector $\mathbf{y}_l = \mathbf{U}_{l-1}^\top \mathbf{x}_l$, i.e., by adopting the last estimated $\mathbf{U}$. Thanks to this approximation, the problem has a closed solution and $\mathbf{U}_t$ can be retrieved by mean of recursive least squares (RLS) methods, which allow for a computational cost as low as $\mathcal{O}(nk)$.

Iterations start from a random $\mathbf{U}_0 \in \mathbb{R}^{n \times k}$ and $\mathbf{P}_0 = \delta \mathbf{I}_k$ for some $\delta > 0$ and the update step is

$$\begin{aligned} \mathbf{g}_t &= \mathbf{P}_{t-1} \mathbf{y}_t / (\beta + \mathbf{y}_t^\top \mathbf{P}_{t-1} \mathbf{y}_t) \\ \mathbf{P}_t &= \beta^{-1} (\mathbf{P}_{t-1} - \mathbf{g}_t \mathbf{P}_{t-1} \mathbf{y}_t) \\ \mathbf{U}_t &= \mathbf{U}_{t-1} - \mathbf{r}_t \mathbf{g}_t^\top \end{aligned} \quad (11)$$

Though $\mathbf{U}_t$ is not guaranteed to be column-orthonormal for any finite t , [16], [37] show that column-orthonormality is achieved asymptotically as t increases. In applications where columns-orthonormality is important at each step, variants of PAST can be adopted. For instance, PASTd in [16] is a version based on the deflation technique that comes at the cost of an increase in complexity. Otherwise in [38] nearly-orthonormality is provided by correction terms applied to each update of $\mathbf{U}$ that keeps complexity at $\mathcal{O}(nk)$.

PAST is the base for other approaches such as [39] that deal with the case in which data is perturbed by colored noise, [19] that is designed to cope with the missing components in the vectors $\mathbf{x}_t$, and [40], [41] where the Approximated Power Iteration (API) extends the standard power method by exploiting the same approximation used in PAST.

E. ISVD

Given any $n \times t$ matrix $\mathbf{A}$, Singular Value Decomposition (SVD) [32, Chapter 2] finds three factors $\mathbf{P}$, $\mathbf{D}$ and $\mathbf{Q}$ such that $\mathbf{A} = \mathbf{P} \mathbf{D} \mathbf{Q}^\top$ with $\mathbf{P}$ and $\mathbf{Q}$ being square orthonormal matrices of size $n \times n$ and $t \times t$ respectively, and $\mathbf{D}$ is a diagonal $n \times t$ matrix whose diagonal entries are called *singular values*. SVD is symbolized as $\mathbf{A} \xrightarrow{\text{SVD}} \mathbf{P}, \mathbf{D}, \mathbf{Q}$.

Let also $\mathbf{X}_t = [\mathbf{x}_1 \ \dots \ \mathbf{x}_t]$ be the matrix containing samples up to the t -th. Subspace analysis has to do with the SVD of $\mathbf{X}_t$ with a very large t . In fact, if $\mathbf{X}_t \xrightarrow{\text{SVD}} \bar{\mathbf{U}}_t, \bar{\mathbf{S}}_t, \bar{\mathbf{V}}_t$, then the singular values $\sigma_j(t)$ in $\bar{\mathbf{S}}_t$ are such that $\sigma_j^2(t) \rightarrow \lambda_j$ and $\bar{\mathbf{U}}_t \rightarrow \Psi$ as $t \rightarrow \infty$.

In [46] it is showed that the SVD of $\mathbf{X}_t$ can be effectively computed from the SVD of $\mathbf{X}_{t-1}$. Even more, [42] shows that this is possible even if we focus on the so-called *thin* SVD (tSVD), i.e., on a decomposition that computes only the first k columns of $\bar{\mathbf{U}}_t$ and $\bar{\mathbf{V}}_t$, the former set of columns being exactly the matrix $\mathbf{U}$ we look for when targeting the principal subspace.

The method relies on the fact that from the decomposition $\mathbf{X}_{t-1} \xrightarrow{\text{tSVD}} \mathbf{U}_{t-1}, \mathbf{S}_{t-1}, \mathbf{V}_{t-1}$ one may express the data matrix $\mathbf{X}_t$ as

TABLE I
CLASSIFICATION OF SA METHODS IN THIS PAPER

Method		Principal/minor subspace	Eigenvectors or subspace	Objective function	Column orthonormalization	Complexity	Original field
Oja	[27] [28] [15] [30] [31] [33]	both	eigenvectors	(2)	QR	nk^2	neural networks
Krasulina	[34] [35] [13]	both	eigenvectors	(1)	QR	nk^2	
HFRANS	[25] [24] [22] [23]	minor	eigenvectors	(1)	approximate	nk	
PAST	[16] [36] [38] [39] [37]	principal	subspace	(3)	approximate	nk	adaptive filtering
ISVD	[42] [43]	principal	eigenvectors	N/A	QR	nk^2	
GROUSE	[18] [44] [45]	principal	subspace	(3)	Grassmannian	nk	matrix completion

$$\mathbf{X}_t = \begin{bmatrix} \mathbf{U}_{t-1} & \frac{\mathbf{r}_t}{\|\mathbf{r}_t\|} \end{bmatrix} \begin{bmatrix} \mathbf{S}_{t-1} & \mathbf{y}_t \\ \mathbf{0}_k^\top & \|\mathbf{r}_t\| \end{bmatrix} \begin{bmatrix} \mathbf{V}_{t-1}^\top & \mathbf{0}_k \\ \mathbf{0}_k^\top & 1 \end{bmatrix} \quad (12)$$

The equality in (12) holds exactly only if the rank of $\mathbf{X}_{t-1}$ is k and is otherwise an approximation. Moreover, (12) is the foundation of ISVD method which compute the SVD of the inner matrix at each update. With more details, the method computes the SVD of

$$\begin{bmatrix} \beta \mathbf{S}_{t-1} & \mathbf{y}_t \\ \mathbf{0}_k^\top & \|\mathbf{r}_t\| \end{bmatrix},$$

which represents an adjusted version of the inner matrix in (12) where the parameter $0 < \beta < 1$ is added to control the memory of the algorithm. The computed SVD yields two orthonormal factors $\mathbf{P}$ and $\mathbf{Q}$ and a diagonal factor $\mathbf{D}$ whose product can be plugged into (12) to obtain an inner diagonal factor and thus yield the updated tSVD of $\mathbf{X}_t$. Such a representation is then shrunk to the minimum by keeping only the first k columns of the left and right factor, and only the first k columns and rows of the central factor.

Overall, starting from a random orthonormal $\mathbf{U}_0 \in \mathbb{R}^{n \times k}$, the update step computes

$$\begin{aligned} \begin{bmatrix} \beta \mathbf{S}_{t-1} & \mathbf{y}_t \\ \mathbf{0}_k^\top & \|\mathbf{r}_t\| \end{bmatrix} &\xrightarrow{\text{SVD}} \mathbf{P}, \mathbf{D}, \mathbf{Q} \\ \mathbf{S}_t &= (\mathbf{D})_{\Gamma_k} \\ \mathbf{U}_t &= \left(\begin{bmatrix} \mathbf{U}_{t-1} & \frac{\mathbf{r}_t}{\|\mathbf{r}_t\|} \end{bmatrix} \mathbf{P} \right)_{|_k} \end{aligned} \quad (13)$$

where $(\cdot)_{|_k}$ is the same operator used before which selects the first k columns of its argument, while $(\cdot)_{\Gamma_k}$ selects the first k columns and the first k rows of its argument.

Besides, the speed of convergence is highly affected by the condition number of $\mathbf{X}_t$. To partially overcome this problem authors in [43] propose the Polar Incremental Matrix Completion (PIMC) algorithm which adapts the memory factor to the norm of the observed samples $\beta = \frac{a_t}{\|\mathbf{S}_t\|_F}$ where $a_t^2 = a_{t-1}^2 + \|\mathbf{x}_t\|_2^2$, $a_0 = 1$ and $\|\cdot\|_F$ denotes the Frobenius norm of a matrix.

F. GROUSE

Grassmannian Rank-One Update Subspace Estimation (GROUSE) is a streaming algorithm for subspace tracking proposed in [18]. Although it is designed to deal with the case in which some components of $\mathbf{x}_t$ are unknown, we here consider the version for complete data. The idea consists in applying the stochastic gradient descent to minimize (3) while making moves that do not exit the set of all possible column-orthonormal matrices, i.e., the the Grassmannian manifold of the k -dimensional subspaces of $\mathbb{R}^n$.

Starting from one of such matrices, the update rule for the case in which all the components of $\mathbf{x}_t$ are known, is

$$\begin{aligned} \mathbf{p}_t &= \mathbf{U}_{t-1} \mathbf{y}_t \\ \theta_t &= \arctan \left[(1 - \alpha_t) \frac{\|\mathbf{r}_t\|}{\|\mathbf{p}_t\|} \right] \\ \mathbf{z}_t &= \cos(\theta_t) \frac{\mathbf{p}_t}{\|\mathbf{p}_t\|} + \sin(\theta_t) \frac{\mathbf{r}_t}{\|\mathbf{r}_t\|} \\ \mathbf{U}_t &= \mathbf{U}_{t-1} + \left(\frac{\mathbf{z}_t}{\|\mathbf{z}_t\|} - \frac{\mathbf{p}_t}{\|\mathbf{p}_t\|} \right) \frac{\mathbf{y}_t^\top}{\|\mathbf{y}_t\|} \end{aligned} \quad (14)$$

where the expression for θ_t is derived from eq. (3)–(4) in [44] and where α_t is meant to mitigate the effect of noise.

Convergence of GROUSE is analyzed in [44], [45] and [26] shows that GROUSE and ISVD are strictly linked. In particular, the application of the ISVD to the missing data case is equivalent to GROUSE for a certain choice of its parameters.

IV. SAMPLE SUBSPACE ANALYSIS PERFORMANCE

In this section we test the functional performance of the methods described in Section III by asking to each method to correctly detect a subspace characterizing a set of signal observations.

Data are generated according to the spiked model [17] with the subspace structure of the signal that does not change in time, meaning that

$$\mathbf{x}_t = \mathbf{\Phi} \mathbf{s}_t + \mathbf{v}_t \quad (15)$$

where $\mathbf{\Phi} \in \mathbb{R}^{n \times m}$, with $m < n$, is a column-orthonormal matrix that expands instances $\mathbf{s}_t \in \mathbb{R}^m$ of a zero-mean random

Fig. 3. Performance comparison for subspace identification: (a) identification of the principal subspace; (b) identification of the minor subspace.

Gaussian source with zero mean and covariance $\mathbf{I}_m$, while $\mathbf{v}_t \in \mathbb{R}^n$ represents realizations of a zero-mean Gaussian noise term with covariance $\rho \mathbf{I}_n$ such that $0 < \rho < 1$ controls the noise level.

With this model we have $\Sigma = \mathbf{E}[\mathbf{x}_t \mathbf{x}_t^\top] = \Phi \Phi^\top + \rho \mathbf{I}_n$, and since Φ is column-orthonormal and $\rho < 1$, then Φ itself spans the m -dimensional principal subspace, while its orthogonal complement $\Phi^\perp$ spans the $(n - m)$ -dimensional minor subspace. As a result, the target $n \times k$ matrix $\mathbf{U}$ is Φ in case of principal subspace estimation and it is $\Phi^\perp$ for the minor subspace.

To quantify the effectiveness in subspace analysis, for each method we monitor the sequence of reconstruction errors

$$e_t = \|\mathbf{U} - \mathbf{U}_t \mathbf{U}_t^\top \mathbf{U}\|_F \quad (16)$$

where $\|\cdot\|_F$ indicates the Frobenius norm of its argument.

In case of correct estimation we have $\mathbf{U}_t = \mathbf{U}$ and thus $e_t = 0$ while the error is maximum when $\mathbf{U}_t$ is orthogonal to $\mathbf{U}$, yielding $e_t = k$.

Simulations consider $n = 100$, $m = 10$ and noise amplitude $\rho = 10^{-3}$. For each task, 100 Montecarlo trials are performed where random column-orthonormal Φ and random column-orthonormal initialization matrices $\mathbf{U}_0$ are drawn independently. Each single Montecarlo trial is composed by a stretch of 1000 sample windows $\mathbf{x}_t$ generated independently following (15).

The parameters controlling each method are set as reported in Table II: Oja and Krasulina need a learning rate γ_t , PAST and ISVD need to set a forgetting factor β , and finally GROUSE and HFRANS depend on α (controlling the effect of noise) and γ (that can be seen as a re-scaled learning rate). Parameters are tuned to optimize the capability to identifies the target subspace.

For the methods adopting a learning rate we tested the classical trend $\gamma_t = c/t^d$ with $d \in \{0, 1/2, 1\}$ and c selected to increase convergence speed. Since, in the fixed subspace case, performances are approximately the same for the three values of d , we selected $d = 0$ and report in Table II only the value of $\gamma_t = \gamma = c$. Such a constant learning rate allows for substantial updates even when t grows.

TABLE II
ALGORITHM PARAMETERS, TUNED VALUE WITH CORRESPONDING RANGES AND UPDATE TIMES (AVERAGE AND STD) FOR THE RPPI EDGE-ORIENTED IMPLEMENTATION

method	parameter	value	range	update time [μ s] ave. $\pm$ std
Principal Subspace identification				
Oja	γ	1	[0.001, 100]	354 ± 10
Krasulina	γ	0.1	[0.001, 100]	378 ± 11
PAST	β	0.99	[0.7, 0.999]	116 ± 5
ISVD	β	0.9	[0.7, 0.999]	585 ± 13
GROUSE	α	0.2	[0, 0.5]	304 ± 10
Minor Subspace identification				
Oja	γ	0.1	[0.001, 100]	3446 ± 60
Krasulina	γ	0.1	[0.001, 100]	3472 ± 27
HFRANS	γ	1	[0.001, 100]	540 ± 29

For HFRANS, the learning rate tuning does not depend on t while the value of α in GROUSE is set to cope with the noise level.

Finally, the forgetting factors of ISVD and PAST are selected as the largest possible values that make the fixed subspace case converge.

Results in terms of e_t are shown in Figure 3. Figure 3a is for the principal subspace while Figure 3b is for the minor. Solid lines represent median values of the Montecarlo trials while shaded areas indicate the spread containing 50% of the values.

As expected, all methods are able to deal with the subspace identification task. As expected, ISVD and PAST are the fastest to converge as they have been designed with the spiked model in mind, GROUSE is in the order of the two historical approaches Oja and Krasulina. This behavior reflects the fact that the main GROUSE novelty is in its ability to manage the missing data case.

Fig. 4. update times averaged with high-end device over 1000 iteration for both tasks: (a) behavior in case of principal subspace identification changing n with a constant $n/k = 10$; (b) behavior in case of minor subspace identification changing n with a constant $n/k = 10/9$;

V. IMPLEMENTATION ON EDGE DEVICES

Since edge devices are employed on a very large variety of applications and for many different tasks, several kinds of device have been proposed ranging from single-board system equipped with a single microcontroller to more complex system that can be composed by different modules. Examples in the latter class are: SONY Spresence dev board¹ with a 6 core cortex ARM M4 microcontroller, Raspberry pi4 model B² with a quad-core ARM A72 microcontroller and STM STM32MP157A dev board³ equipped with both an AMR M4F micorcontroller and a dual ARM A7 microcontroller. All these system are able to support an operative system such that high level programming languages could be considered in the developing of the reviewed methods, e.g., Python with dedicated libraries for digital signal processing and numerical computing. On the contrary, single board system does not support high level programming languages with the need of device direct programming. To cope with both scenarios two subsection report implementation details.

A. High-end edge devices

We here focus on the Raspberry pi 4 model B which could be considered a reference for the high-end edge devices family. With more details, we refer to the board equipped with a 1GB of RAM and having as operative system the Raspberry Pi OS (32-bit) Lite 5.4.51-v7l+. Clock frequency ranges from 600 MHz to 800 MHz⁴. Methods performance has been computed by adopting a Python (version 3.7.3) implementation with the packages Scipy (version 1.1.0) and Numpy (version 1.16.2) for the required algebraic manipulations.

To avoid misleading comparison between methods performing the identification of the principal subspace with respect to the ones performing identification of the minor subspace we remark that the $n \times k$ matrix to be estimated is an $n \times m$ in

case of principal subspace while minor subspace identification looks at an $n \times (n - m)$ matrix. In case of $m \ll n$, the computational budget for the minor subspace case amply exceeds what is expected for the principal subspace. We consider here the setting $n = 100$ and $m = 10$ (the one adopted in Sec. IV), averages and standard deviations of the update times are in Tab. II where 1000 different updates have been performed for each method and for each task. As expected, computational time in case of minor subspace identification is higher compared to the the methods identifying the principal subspace.

For principal subspace identification, the most time consuming method is ISVD which computes a singular value decomposition of an $(k + 1) \times (k + 1)$ matrix at each update. On the contrary, PAST is the method with the lowest computational time. This is mainly due to the fact the PAST does not require any complex operation as either orthonormalization or SVD. As a further remark, note that, as expected, standard deviations are negligible since difference between single time updates are only due to the operative system activity. In case of minor subspace identification the methods to be preferred in terms of computational times is HFRANS. As for PAST this is a method that does not require any complex operation at each update.

To enlarge this analysis several n, k couples of values were tested in order better compare the reviewed methods in terms of required computational time. Here we keep constant the ratio between n and k and Fig. 4a shows how the average computational times increases with n in case of $n/k = 100/10$ for the principal subspace and $n/k = 100/90$ for the minor subspace.

Regardless of the task to be performed, we can identify two different classes of method. Oja, Krasulina and ISVD that require at each step either an orthonormalization of the columns or an SVD and HFRANS, GROUSE and PAST that do not require complex operations thus reducing the computational time with respect to the first group of methods.

¹online <https://developer.sony.com/develop/spresence/specifications>

²online <https://www.st.com/en/evaluation-tools/stm32mp157a-ev1.html>

³online <https://www.st.com/en/evaluation-tools/stm32mp157a-ev1.html>

⁴Clock frequency is dynamically managed by the firmware depending on the CPU temperature.

B. Low-end edge devices

The description of how we propose to direct implement discussed methods on low resources devices is here presented. In details, we developed a C library⁵ targeting the ARM Cortex microcontroller (MCU) family of devices. The library back-end is mainly based on the well know ARM CMSIS-DSP library⁶.

The implementation of the methods is not straight forward as we want to reduce to the minimum the memory requirements and the processing time of the methods. In Appendix A it is possible to find the tricks that have been used for the implementation of fast and low-memory matrix operations.

Both GROUSE and PAST implementations adopts all these gimmicks to reduce as possible memory footprint and computational complexity. Other than this, Oja, Krasulina and ISVD require an extra care due two main main bottlenecks, namely QR decomposition and SVD operation. The overcoming of these strong limitations is described hereafter.

After that, speed performance, energy consumption and memory requirements on the MCU are analyzed, with a final assessment on the performance for both subspace identification tasks.

1) *QR decomposition*: The operator $\Omega(\cdot)$ that makes column-orthonormal its argument is in (4) and (6), the two update rules for Oja and Krasulina methods. It can be obtained through the QR decomposition operation where several implementation have been proposed.

Householder QR decomposition (Hous-QR) [47] is one of the most used algorithms as it guarantees a great numerical stability, even in case of column-orthonormalization of ill-conditioned matrices. This means that the output matrix is always as nearest as possible to a column-orthonormal matrix, i.e., the difference between I_k and $U_t^T U_t$ is in the order of the machine epsilon. Nonetheless, Hous-QR is an iterative algorithm that requires multiple matrix multiplications for each iteration, thus making it computationally very expensive.

A less expensive algorithm is the Modified Gram-Schmidt QR decomposition (MGS-QR) [48]. However, the price for a reduced complexity comes with the possibility of numerical instability: U_t is only guaranteed to be approximately column-orthonormal, especially if the argument of the QR operator is an ill-conditioned matrix.

Finally, it is also possible to employ Cholesky decomposition (Chol-QR) [49]. This algorithm is even faster if compared to MGS-QR [50], with the price of an even greater numerical instability. This translates in an output matrix U_t that is always only approximately column-orthonormal. While this is a critical problem for many applications, in the case of Oja and Krasulina methods we noticed that they are quite robust to this limitation, i.e., both methods well tolerate matrices U_t that are approximately columns-orthonormal. Because of this, Chol-QR does not affect the performance of the two algorithms and thus has been selected for the final implementation.

Chol-QR algorithm is quite simple and its description can be found in Algorithm 1.

Algorithm 1 Cholesky-based QR decomposition

- 1: $\text{cholesky}(A^T A) \rightarrow LL^T$ where L is lower triangular
 - 2: $R \leftarrow L^T$
 - 3: $R^{-1} \leftarrow BS(R)$ (BS : backward substitution)
 - 4: $Q \leftarrow AR^{-1}$
-

The implementation of Cholesky decomposition is based on ARM CMSIS-DSP library. Also, the inversion of R can be easily performed with backward substitution technique [12] as it is an upper triangular matrix. For the same reason, R and R^{-1} can be stored as the two triangular parts of a single square matrix, because R^{-1} is instead lower triangular.

2) *SVD operation*: The SVD operation $A \xrightarrow{SVD} P, D, Q$ is required in the ISVD method (Equation 13). The algorithm used on MCU is Golub-Reinsch SVD (GR-SVD) [51], whose computational time is comparable to Chol-QR. The implementation of GR-SVD is based on Daniel Matterson's CControl library⁷.

In order to accelerate the final implementation, the evaluation of Q has been elided as it is not needed for ISVD. Also, GR-SVD can be performed directly overwriting matrix A with matrix P during computation, thus keeping the memory requirements to the minimum.

The whole methods implementations have been tested on STM32H743ZIT (rev. V), MCU based on ARM Cortex M7 family, with a 32-bit floating point unit, $f_{CLK} = 480$ MHz and both instruction cache and data cache enabled⁸.

With this setup, the energy consumption of a single update has been estimated as $E_{\text{update}} = V_{DD} \times I_{DD} \times t_{\text{update}}$, where V_{DD} is the supply voltage, I_{DD} is the absorbed current and t_{update} is number of clock cycles necessary to apply a single update rule of a method divided by the clock frequency. Values are obtained from datasheet. In particular, we refer to current values corresponding to $V_{DD} = 1.8$ V and with both no peripherals or all the peripherals enabled.

The results obtained for time per update and energy consumption per update are reported in Table III. Figure 5 shows instead how time per update scales with n with a fixed n/k ratio, which is proportional to energy consumption.

For each method, memory footprint can be split in three parts. The first part consists in the stack memory, this is a fixed cost that we remove from our analysis since it is negligible, in general, in the computation of the whole memory footprint and since it does not depend on either the adopted method or the considered setting (n and m values). The second part is equal for any method and consists in the input vector x and matrix U_t , respectively of size n and $n \times k$. Finally the third part consists of other extra memory buffers of various size required by each method.

The size of the extra buffers, along with the overall memory rule for each method and the memory required in case $n = 100$

⁵online repository https://github.com/SSIGPRO/streaming_pca

⁶online repository https://github.com/ARM-software/CMSIS_5

⁷online repository <https://github.com/DanielMartensson/CControl>

⁸code has been compiled with fast target gcc option (-Ofast) in order to maximize the speed performance.

TABLE III
PERFORMANCE ON STM32H743ZIT (REV. V) @ 1.8 V, 480 MHz, CACHE
ON ($n = 100, m = 10$)

Method	Clock cycles per update	Time per update	Energy per update (no peripherals)	Energy per update (all peripherals)
Principal subspace				
Oja	109.97 k	229.10 μ s	45.36 μ J	90.72 μ J
Krasulina	114.86 k	239.28 μ s	47.38 μ J	94.76 μ J
ISVD	136.02 k	283.37 μ s	56.11 μ J	112.21 μ J
PAST	20.18 k	42.04 μ s	8.32 μ J	16.65 μ J
GROUSE	22.96 k	47.83 μ s	9.47 μ J	18.94 μ J
Minor subspace				
Oja	8711.27 k	18 148.47 μ s	3593.40 μ J	7186.80 μ J
Krasulina	8748.48 k	18 226.01 μ s	3608.75 μ J	7217.50 μ J
HFRANS	235.30 k	490.20 μ s	97.06 μ J	194.12 μ J

TABLE IV
MEMORY REQUIREMENTS RULE FOR EACH METHOD (MEMORY EXAMPLE
WITH $n = 100, m = 10, 32$ -BIT SCALARS)

Method	Size of extra buffers	Overall memory rule	Memory example
Principal subspace			
Oja	$k \times k$	$n(k+1) + k^2$	4.80 kB
Krasulina	$k \times k$	$n(k+1) + k^2$	4.80 kB
ISVD	$k+1 \times k+1, k+1, k+1$	$n(k+1) + k(k+4) + 3$	4.97 kB
PAST	$k \times k, n, k$	$n(k+2) + k(k+1)$	5.24 kB
GROUSE	n, k	$n(k+2) + k$	4.84 kB
Minor subspace			
Oja	$k \times k$	$n(k+1) + k^2$	17.20 kB
Krasulina	$k \times k$	$n(k+1) + k^2$	17.20 kB
HFRANS	n, k	$n(k+2) + k$	9.29 kB

and $m = 10$, can be found in Table IV ⁹.

Finally, the subspace identification performance of the algorithms on MCU has been assessed with the same setup reported in Section IV and can be seen in Figure 6.

VI. CONCLUSION

We review six major approaches to subspace identification by streaming computation that tackle both principal and minor analysis.

The aim is to give a coherent presentation of the methods that gave rise to most of the subtle variants currently available in the Literature, and perform some elementary comparison with an eye to their deployment at the edge of a system that concentrates data from a host of distributed nodes.

Beyond functional testing, we implement the methods in both an high-end edge device and a low-end edge devices to assess how much their computational requirements are compatible with the need of nowadays data collection systems that must reduce redundancy as early as possible along the signal chain, to maintain efficiency in presence of massive data flows.

⁹without buffered multiplication technique described in Appendix A, memory requirements for Oja and Krasulina are about 45% higher while computation is about 19% faster compared to the results shown in this section.

At least two of the methods reviewed (PAST and HFRANS) excel in this as they are able to deliver substantial accuracy in principal subspace identification (PAST) and minor subspace identification (HFRANS) with very limited computational requirements.

If heavier computations can be tolerated, maximum speed convergence in the identification and tracking of principal subspaces can be obtained by ISVD.

APPENDIX A

FAST AND MEMORY EFFICIENT ARITHMETIC ON MCU

Being the arithmetic based on ARM CMSIS-DSP library, on MCU a single $n \times k$ matrix is stored as a linear vector of size nk . In this way memory contiguity is maximized and the number of memory accesses is reduced to the minimum.

A brief overview of the techniques used for vector and matrix multiplications follows.

1) *Loop unroll*: as loops are massively used, unrolling highly increases the performance. In fact, it allows to reduce the amount of data to be transferred from memory and reduces the number of updates of the loop index. This is performed automatically at compile time by using -Ofast gcc option.

2) *Register blocking*: this technique is illustrated in Figure 7(a). The idea behind matrix-matrix multiplication can be simply reduced to multiple subsequent vector-vector dot products. Each dot product is performed through multiply and accumulate operations and generates one of the scalars in the output matrix. By using multiple accumulators at the same time, i.e., by interleaving two or more dot products, local registers are better employed and performance greatly increases.

3) *Buffered multiplication*: matrix-matrix multiplications where the first matrix is $n \times k$ and the second one is $k \times k$ are common (Oja, Krasulina, ISVD). The result is a $n \times k$ matrix, and the total memory required for this operation would be $(2n + k)k$. If $n \gg k$, memory requirements can be almost halved by storing the output matrix in the same memory location of the first input matrix, thus overwriting it. This is possible if, for each output row, the input row is temporarily copied in a k -sized buffer. On the other hand, this slightly increases the computation time. This method can be applied also in the transposed case ($k \times k$ times $k \times n$ matrix-matrix multiplication). This technique is illustrated in Figure 7(b).

4) *Vector outer product and matrix addition merging*: some methods (Krasulina, PAST, GROUSE, HFRANS) requires update operations of type $A = A + ab^T$, where $A \in \mathbb{R}^{n \times k}$, $a \in \mathbb{R}^n$ and $b \in \mathbb{R}^k$. Instead of evaluating first $B = ab^T$ and then performing $A = A + B$, it is possible to directly sum each entry of B to each entry of A the moment it is evaluated, thus reducing the memory requirements and the computation time of the overall operation. This is shown in Figure 7(c).

5) *Transposition of square matrices*: in general, the transposition of a rectangular matrix requires the copy of the entire matrix in another memory space. In the case of a square matrix it is instead possible to directly transpose the matrix overwriting the original, by swapping each of the values in the two triangular parts of the matrix.

Fig. 5. Maximum update time (over 10 updates) on STM32H743ZIT (rev. V) @ 480 MHz for principal subspace identification (a) and minor subspace identification (b). Values are reported with varying n and constant ratio n/k . Times are greater in the case of minor subspace as k is much greater.

Fig. 6. Performance comparison for subspace identification on STM32H743ZIT (rev. V) with a single random instance of $\Phi \in \mathbb{R}^{100 \times 10}$ and noise $\rho = 10^{-3}$: (a) identification of the principal subspace ($k = 10$); (b) identification of the minor subspace ($k = 90$). Shaded areas indicate the 50% spread of Figure 3 for comparison purposes.

6) *Transposed multiplications*: when performing multiplications such as AB^T or $A^T B$, it is possible to avoid the transposition operations entirely by modifying multiplication operations by scanning the transposed matrix row-first instead of column-first (or vice-versa).

7) *Column concatenation*: ISVD method requires the concatenation of a column vector. Columns are not memory-contiguous while rows are. In order to greatly simplify the operation and reduce the computation time of the operation, it is possible to transpose the whole method so that the column concatenation operation becomes a row concatenation.

Fig. 7. Matrix arithmetic optimization techniques: register blocking in (a) consists in performing multiple vectors dot-products “at the same time”; buffered multiplication in (b) copies each row to a buffer before overwriting the original matrix; vector outer product and matrix addition merging in (c) is performed by adding and overwriting each value of A with each of the values generated by ab^T .

REFERENCES

- [1] P. Zhang, X. Huang, X. Sun, H. Wang, and Y. Ma, “Privacy-preserving anomaly detection across multi-domain networks,” in *9th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)*, 2012, pp. 1066–1070. doi: 10.1109/FSKD.2012.6234272
- [2] A. Marchioni, M. Mangia, F. Pareschi, R. Rovatti, and G. Setti, “Subspace energy monitoring for anomaly detection @Sensor or @Edge,” *IEEE Internet of Things Journal*, pp. 1–1, 2020. doi: 10.1109/jiot.2020.2985912
- [3] V. Senger and R. Tetzlaff, “New signal processing methods for the development of seizure warning devices in epilepsy,” *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 63, no. 5, pp. 609–616, May 2016. doi: 10.1109/TCSI.2016.2553278
- [4] B. Yu, T. Mak, X. Li, F. Xia, A. Yakovlev, Y. Sun, and C. S. Poon, “Real-time FPGA-based multichannel spike sorting using hebbian eigenfilters,” *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 1, no. 4, pp. 502–515, Dec. 2011. doi: 10.1109/JET-CAS.2012.2183430

- [5] Xiao-Long Zhu, Xian-Da Zhang, Zi-Zhe Ding, and Ying Jia, "Adaptive nonlinear PCA algorithms for blind source separation without prewhitening," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 53, no. 3, pp. 745–753, Mar. 2006. doi: 10.1109/TCSI.2005.858489
- [6] L. A. Thomaz, E. Jardim, A. F. da Silva, E. A. B. da Silva, S. L. Netto, and H. Krim, "Anomaly Detection in Moving-Camera Video Sequences Using Principal Subspace Analysis," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 65, no. 3, pp. 1003–1015, Mar. 2018. doi: 10.1109/TCSI.2017.2758379
- [7] C. Ding *et al.*, "Non-contact human motion recognition based on UWB radar," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 8, no. 2, pp. 306–315, Jun. 2018. doi: 10.1109/JETCAS.2018.2797313
- [8] I. T. Jolliffe, *Principal components analysis*. Elsevier, 1986. ISBN 978-0-08-044894-7
- [9] L. Balzano, Y. Chi, and Y. M. Lu, "Streaming PCA and Subspace Tracking: The Missing Data Case," *Proceedings of the IEEE*, vol. 106, no. 8, pp. 1293–1310, Aug. 2018. doi: 10.1109/JPROC.2018.2847041
- [10] T. V. Marinov, P. Mianjy, and R. Arora, "Streaming principal component analysis in noisy setting," in *Proceedings of the 35th international conference on machine learning*, J. Dy and A. Krause, Eds., vol. 80. PMLR, Jul. 2018, pp. 3413–3422.
- [11] N. Lassami, A. Aïssa-El-Bey, and K. Abed-Meraim, "Low cost sparse subspace tracking algorithms," *Signal Processing*, vol. 173, p. 107522, Aug. 2020. doi: 10.1016/j.sigpro.2020.107522
- [12] S. Yin, S. X. Ding, X. Xie, and H. Luo, "A Review on Basic Data-Driven Approaches for Industrial Process Monitoring," *IEEE Transactions on Industrial Electronics*, vol. 61, no. 11, pp. 6418–6428, Nov. 2014. doi: 10.1109/TIE.2014.2301773
- [13] C. Tang, "Exponentially convergent stochastic k-PCA without variance reduction," in *Adv. Neural Inf. Process. Syst.* 32, H. Wallach, H. Larochelle, Beygelzimer A., F. D'Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 12393–12404.
- [14] N. Tripuraneni, N. Flammarion, F. Bach, M. I. Jordan, S. Bubeck, V. Perchet, and P. Rigollet, "Averaging stochastic gradient descent on riemannian manifolds nilesh tripuraneni," in *Proc. Mach. Learn. Res.*, S. Bubeck, V. Perchet, and P. Rigollet, Eds., vol. 75, 2018, pp. 650–687.
- [15] R. Arora, A. Cotter, K. Livescu, and N. Srebro, "Stochastic optimization for PCA and PLS," in *50th annual allerton conference on communication, control, and computing (allerton)*, Oct. 2012, pp. 861–868. doi: 10.1109/Allerton.2012.6483308
- [16] B. Yang, "Projection Approximation Subspace Tracking," *IEEE Trans. Signal Process.*, vol. 43, no. 1, pp. 95–107, 1995. doi: 10.1109/78.365290
- [17] I. M. Johnstone, "On the distribution of the largest eigenvalue in principal components analysis," *Ann. Statist.*, vol. 29, no. 2, pp. 295–327, Apr. 2001. doi: 10.1214/aos/1009210544
- [18] L. Balzano, R. Nowak, and B. Recht, "Online identification and tracking of subspaces from highly incomplete information," in *48th annu. Allert. Conf. Commun. Control. Comput. Allert. 2010*, 2010, pp. 704–711. doi: 10.1109/ALLERTON.2010.5706976
- [19] Y. Chi, Y. C. Eldar, and R. Calderbank, "PETRELS: Subspace estimation and tracking from partial observations," in *IEEE int. Conf. Acoust. Speech signal process. (ICASSP)*, Mar. 2012, pp. 3301–3304. doi: 10.1109/ICASSP.2012.6288621
- [20] A. Edelman, T. A. Arias, and S. T. Smith, "The geometry of algorithms with orthogonality constraints," *SIAM J. Matrix Anal. Appl.*, vol. 20, no. 2, pp. 303–353, Jan. 1998. doi: 10.1137/S0895479895290954
- [21] E. Oja, "Principal components, minor components, and linear neural networks," *Neural Networks*, vol. 5, no. 6, pp. 927–935, Nov. 1992. doi: 10.1016/S0893-6080(05)80089-9
- [22] K. Abed-Meraim, A. Chkeif, and Y. Hua, "Orthogonal Oja Algorithm," *IEEE Signal Process. Lett.*, vol. 7, no. 5, pp. 116–119, May 2000. doi: 10.1109/97.841157
- [23] S. Attallah and K. Abed-Meraim, "Fast algorithms for subspace tracking," *IEEE Signal Process. Lett.*, vol. 8, no. 7, pp. 203–206, Jul. 2001. doi: 10.1109/97.928678
- [24] S. Attallah and K. Abed-Meraim, "Low-cost adaptive algorithm for noise subspace estimation," *Electron. Lett.*, vol. 38, no. 12, pp. 609–611, Jun. 2002. doi: 10.1049/el:20020388
- [25] S. Attallah, "The generalized Rayleigh's quotient adaptive noise subspace algorithm: a householder transformation-based implementation," *IEEE Trans. Circuits Syst. II Express Briefs*, vol. 53, no. 1, pp. 3–7, Jan. 2006. doi: 10.1109/TCSII.2005.854587
- [26] L. Balzano and S. J. Wright, "On GROUSE and incremental SVD," in *5th IEEE international workshop on computational advances in multi-sensor adaptive processing, (CAMSAP)*, 2013, pp. 1–4. doi: 10.1109/CAMSAP.2013.6713992
- [27] E. Oja, "Simplified neuron model as a principal component analyzer," *J. Math. Biol.*, vol. 15, no. 3, pp. 267–273, 1982. doi: 10.1007/BF00275687
- [28] E. Oja and J. Karhunen, "On stochastic approximation of the eigenvectors and eigenvalues of the expectation of a random matrix," *J. Math. Anal. Appl.*, vol. 106, no. 1, pp. 69–84, Feb. 1985. doi: 10.1016/0022-247X(85)90131-3
- [29] R. A. Horn and C. R. Johnson, *Matrix analysis*. Cambridge university press, 2012.
- [30] I. Mitliagkas, C. Caramanis, and P. Jain, "Memory limited, streaming PCA," in *Adv. Neural Inf. Process. Syst.* 26, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2013, pp. 2886–2894.
- [31] M. Hardt and E. Price, "The noisy power method: A meta algorithm with applications," in *Adv. Neural Inf. Process. Syst.* 27, G. Z., W. M., C. C., L. N. D., and W. K. Q., Eds. Curran Associates, Inc., 2014, pp. 2861–2869.
- [32] G. H. Golub and F. V. L. Charles, *Matrix computations*. Johns Hopkins University Press, 2012. ISBN 978-1-4214-0794-4
- [33] Z. Allen-Zhu and Y. Li, "First efficient convergence for streaming k-PCA: A global, gap-free, and near-optimal rate," in *58th annual symposium on foundations of computer science (FOCS)*, vol. 2017-10. IEEE Computer Society, Nov. 2017, pp. 487–492. doi: 10.1109/FOCS.2017.51
- [34] T. P. Krasulina, "The method of stochastic approximation for the determination of the least eigenvalue of a symmetrical matrix," *USSR Comput. Math. Math. Phys.*, vol. 9, no. 6, pp. 189–195, Jan. 1969. doi: 10.1016/0041-5553(69)90135-9
- [35] T. P. Krasulina, "Method of stochastic approximation in the determination of the largest eigenvalue of the mathematical expectation of random matrices," *Autom. Remote Control*, no. 2, pp. 215–221, 1970.
- [36] B. Yang, "Subspace tracking based on the projection approach and the recursive least squares method," in *IEEE int. Conf. Acoust. Speech, signal process. (ICASSP)*, vol. 4, Apr. 1993, pp. 145–148. doi: 10.1109/ICASSP.1993.319615
- [37] Y. Miao and Y. Hua, "Fast subspace tracking and neural network learning by a novel information criterion," *IEEE Trans. Signal Process.*, vol. 46, no. 7, pp. 1967–1979, 1998. doi: 10.1109/78.700968
- [38] K. Abed-Meraim, A. Chkeif, and Y. Hua, "Fast orthonormal PAST algorithm," *IEEE Signal Process. Lett.*, vol. 7, no. 3, pp. 60–62, Mar. 2000. doi: 10.1109/97.823526
- [39] T. Gustafsson, "Instrumental variable subspace tracking using projection approximation," *IEEE Trans. Signal Process.*, vol. 46, no. 3, pp. 669–681, 1998. doi: 10.1109/78.661334
- [40] R. Badeau, G. Richard, B. David, and K. Abed-Meraim, "Approximated power iterations for fast subspace tracking," in *7th Int. Symp. Signal process. Its appl. (ISSPA)*, vol. 2. IEEE, 2003, pp. 583–586. doi: 10.1109/ISSPA.2003.1224944
- [41] R. Badeau, B. David, and G. Richard, "Fast approximated power iteration subspace tracking," *IEEE Trans. Signal Process.*, vol. 53, no. 8, pp. 2931–2941, Aug. 2005. doi: 10.1109/TSP.2005.850378
- [42] M. Brand, "Fast low-rank modifications of the thin singular value decomposition," *Linear Algebra Appl.*, vol. 415, no. 1, pp. 20–30, May 2006. doi: 10.1016/j.laa.2005.07.021
- [43] R. Kennedy, C. J. Taylor, and L. Balzano, "Online completion of ill-conditioned low-rank matrices," in *IEEE glob. Conf. Signal Inf. Process. Glob. (GlobalSIP)*, Feb. 2014, pp. 507–511. doi: 10.1109/GlobalSIP.2014.7032169
- [44] D. Zhang and L. Balzano, "Global convergence of a Grassmannian gradient descent algorithm for subspace estimation," in *Proc. 19th int. Conf. Artif. Intell. Stat., A. Gretton and C. C. Robert, Eds.*, 2016, pp. 1460–1468.
- [45] L. Balzano and S. J. Wright, "Local convergence of an algorithm for subspace identification from partial data," *Found. Comput. Math.*, vol. 15, no. 5, pp. 1279–1314, Oct. 2015. doi: 10.1007/s10208-014-9227-7
- [46] J. R. Bunch and C. P. Nielsen, "Updating the singular value decomposition," *Numerische Mathematik*, vol. 31, no. 2, pp. 111–129, Jun. 1978. doi: 10.1007/BF01397471
- [47] R. Schreiber and C. Van Loan, "A Storage-Efficient WY Representation for Products of Householder Transformations," *SIAM Journal on Scientific and Statistical Computing*, vol. 10, no. 1, pp. 53–57, Jan. 1989. doi: 10.1137/0910005
- [48] Fuyun Ling, D. Manolakis, and J. Proakis, "A recursive modified Gram-Schmidt algorithm for least-squares estimation," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 34, no. 4, pp. 829–836, Aug. 1986. doi: 10.1109/TASSP.1986.1164877

- [49] T. Terao, K. Ozaki, and T. Ogita, “LU-Cholesky QR algorithms for thin QR decomposition,” *Parallel Computing*, vol. 92, p. 102571, Apr. 2020. doi: 10.1016/j.parco.2019.102571
- [50] A. Imakura and Y. Yamamoto, “Efficient implementations of the modified Gram–Schmidt orthogonalization with a non-standard inner product,” *Japan Journal of Industrial and Applied Mathematics*, vol. 36, no. 2, pp. 619–641, Jul. 2019. doi: 10.1007/s13160-019-00356-4
- [51] G. H. Golub and C. Reinsch, “Singular value decomposition and least squares solutions,” *Numerische Mathematik*, vol. 14, no. 5, pp. 403–420, Apr. 1970. doi: 10.1007/BF02163027