A Conservative Aggressive Subspace Tracker

Koby Crammer

Dept. of Computer and Information Science, University of Pennsylvania , Philadelphia, PA 19104
crammer@cis.upenn.edu

Abstract

The need to track a subspace accurately describing well a stream of points arises in many signal processing applications. In this work, we present a very efficient algorithm using a machine learning approach, which its goal is to de-noise the stream of input points. The algorithm guarantees the orthonormality of the representation it uses. We demonstrate the merits of our approach using simulations.

Index Terms: Adaptive filtering, subspace tracking, machine learning, semi-definite programming, eigen vectors

1. Introduction

Subspace tracking has been a widely studied problem in the field of signal processing. The problem we consider is on-the-fly tracking of linear subspaces which approximates best a given sequence of points. Formally, let $\boldsymbol{x}_t \in \mathbb{R}^d$ for $t = 1, 2, \ldots$ be a series of column vectors. Recent algorithms (e.g. [1, 2, 3]) formalize the problem as finding the major subspace of an instantaneous correlation matrix $\sum_{\tau}^{t} \beta^{t-\tau} \boldsymbol{x}_\tau \boldsymbol{x}_\tau^\top$ (for $0 < \beta < 1$). This major subspace is represented using a rectangular matrix $A_t \in \mathbb{R}^{d\times r}$ for a small value r. We follow up previous work [4] and propose an alternative approach which views the problem as a filtering problem where each input vector is a sum of two components: $\boldsymbol{y}_t$ which lies in a low dimensional linear subspace and $\boldsymbol{\nu}_t$ – an unknown noise for this point.

This matrix A_t can be used for de-noising by constructing the matrix $Q_t = A_t A_t^\top$. If A_t is composed of orthonormal vectors, Q_t is a *projection*, that is, it is symmetric, positive semidefinite (PSD) and idempotent (its eigenvalues are either zero or one). In this paper we adopt this view and focus on learning matrices P_t of this form. In the first stage of the derivation we do not reduce explicitly the number of components of a vector (by using the matrix A). In other words, we seek a low-dimensional subspace, but represent data in the original vector space of dimension d. Only in the second stage we use a low rank representation. Since the restriction that the eigenvalues will be *either* zero or one is algorithmically challenging because it involves integer programming, we relax the idempotency assumption. Our learning algorithm seeks linear transformations which are symmetric and positive semidefinite. We refer to these transformations (P or P_t) as projections. When the projections are also idempotent (i.e. all eigenvalues are either zero or one), we will refer to them as *idempotent projections* (Q). The algorithm described below always maintains a linear transformation with eigenvalues *between* zero and one. This is often considered the natural relaxation of idempotency.

Notation: For a matrix P, the property of being a positive semi-definite matrix is denoted by $P \succeq 0$. Given a vector $\boldsymbol{x}$, we denote by $X = \boldsymbol{x}\boldsymbol{x}^\top$ the outer-product of $\boldsymbol{x}$ with itself. A unit vector is denoted by $\hat{\boldsymbol{x}} = \boldsymbol{x}/\|\boldsymbol{x}\|$, and $\hat{X} = \hat{\boldsymbol{x}}\hat{\boldsymbol{x}}^\top$ is a rank-one symmetric matrix with eigenvalue 1. Finally, $\|P\|_p$ is ℓ_p norm of the vector generated by concatenating the columns of P.

2. Algorithm

Crammer [4] proposed a machine-learning based approach to online or adaptive filtering by tracking an appropriate linear subspace. The algorithm works in rounds. On round t the algorithm receives an input vector $\boldsymbol{x}_t$ and outputs a filtered or clean vector $P_t\boldsymbol{x}_t$. It also updates in prediction function, that is the symmetric positive semi-definite matrix P_t. In the next two sections we present the basic update rule and regularization. We then slightly modify the algorithm to maintain low rank matrices P_t and provide an efficient $O(dr)$ update (where r is the dimension of the reduced space). This is a major improvement over the $O(d^3)$ complexity of the algorithm of [4].

2.1. Update Rule

We now describe our algorithm for subspace tracking. The algorithm employs a non-negative parameter ϵ, which is as a threshold to the amount of error we are willing to suffer. A more detailed discussion can be found in [4]. After an input point $\boldsymbol{x}_t$ has been observed we update our current subspace (represented by P_t) based on this point. The update rule for the algorithm sets the new matrix P_{t+1} to be the solution to the following projection problem [5],

$$\min_P \frac{1}{2}\|P - P_t\|^2 \text{ s.t. } \frac{1}{2}\|\boldsymbol{x}_t - P\boldsymbol{x}_t\|^2 \le \epsilon \, , P = P^T \, , P \succeq 0 \, .$$

The second and third constraints ensure that the eigenvalues of P_{t+1} are positive real numbers. Thus, similarly to other subspace methods we will be able to reduce the dimension by performing eigen-decomposition. The solution of the optimization problem is the projection of P onto the intersection of the positive semidefinite cone and the second order body of matrices P that satisfy $\|\boldsymbol{x}_t - P\boldsymbol{x}_t\|^2 \le 2\epsilon$ and are centered at the identity matrix I . Clearly the subset of matrices defined by the intersection is not empty as it contains the identity matrix.

This optimization problem attempts to keep P_{t+1} as close to P_t as possible, while forcing P_{t+1} to have a low reconstruction error, $\|\boldsymbol{x}_t - P\boldsymbol{x}_t\|^2 \le 2\epsilon$ on recent input vector. The resulting algorithm is *conservative* whenever the current model P_t already suffers a small reconstruction error, that is, $P_{t+1} = P_t$ whenever $\|\boldsymbol{x}_t - P_t\boldsymbol{x}_t\|^2 \le 2\epsilon$. In contrast, on rounds for which $\|\boldsymbol{x}_t - P_t\boldsymbol{x}_t\|^2 > 2\epsilon$ we aggressively force P_{t+1} to satisfy the constraint $\|\boldsymbol{x}_t - P_{t+1}\boldsymbol{x}_t\|^2 \le 2\epsilon$. We refer to this algorithm as CAST (conservative-aggressive subspace tracker).

The update rule for this algorithm is derived in length by Crammer [4] and is given by the following set of equations,

$$P'_{t+1} = P_t + \gamma_t \hat{X}_t - \frac{\gamma_t}{2-\gamma_t}(P_t\hat{X}_t + \hat{X}_t P_t) + \frac{\gamma_t^2}{2-\gamma_t}\hat{X}_t P_t \hat{X}_t \, . \quad (1)$$

where the parameter γ_t satisfies $\gamma_t \in [0,1]$ and is computed as follows. Crammer [4] shows that whenever γ_t is non-zero, the inequality constraint $\frac{1}{2}\|\boldsymbol{x}_t - P_{t+1}\boldsymbol{x}_t\|^2 \leq \epsilon$ is satisfied as equality. Long algebraic manipulations yield that the left side of this equality constraint is given by the function,

$$f(\gamma) = a\left[4\left\|\hat{\boldsymbol{x}}_t - P_t\hat{\boldsymbol{x}}_t\right\|^2 + (\gamma^2 - 4\gamma)\left(1 - \hat{\boldsymbol{x}}_t^\top P_t \hat{\boldsymbol{x}}_t\right)^2\right], \tag{2}$$

where $a = (1-\gamma)^2\|\boldsymbol{x}_t\|^2/(2(2-\gamma)^2)$.

Theoretically, we can solve analytically the equation $f(\gamma_t) = \epsilon$ since it is a degree four polynomial. In practice we use [4, Lemma 1] which states that the function $f(\gamma)$ is monotone and that $\gamma_t \in [0,1]$. Thus, we can find a value of γ which is far of the exact solution by at most δ in time complexity of $-\log_2(\delta)$. To summarize the description of this part of the algorithm: after receiving $\boldsymbol{x}_t$ the algorithm checks whether the Euclidean distance between $\boldsymbol{x}_t$ and $P_t\boldsymbol{x}_t$ is below the predefined threshold, $\|\boldsymbol{x}_t - P_t\boldsymbol{x}_t\|^2 \leq 2\epsilon$. If so, it does nothing. Otherwise, it finds a value for γ_t such that $f(\gamma_t) \approx \epsilon$ and sets the value of P_{t+1} according to (1). We initialize $P_1 = 0$.

2.2. Regularization

In the update step described until now , overfitting arises when the eigenvalues of the linear operator P_t have large components orthogonal to the target subspace. As a consequence, the *filtered* output $P_t\boldsymbol{x}_t$ will include noise components as well as the true signal. The algorithm may suffer from this problem since in our setting there is no feedback. Therefore, our algorithm approximate the true feedback using $\boldsymbol{x}_t$, which contains also noise components. Furthermore, the only goal of the update rule of the algorithm is to reduce the loss related to $\boldsymbol{x}_t$, ignoring any other issues such as the rank or trace of the transformation P_t. Therefore, the algorithm favors an increase in the trace of the transformations since, in general, that reduces the loss suffered. A concrete example can be found in [4, Sec. 4].

We thus add a second step to the update rule, after the primary update of (1). The algorithm employs an additional parameter $r > 0$, which is used to bound the trace of the eigenvalues after the update. Intuitively, the parameter r specifies a continuous requirement that replaces a standard rank requirement. Only in the next stage we will enforce a low rank requirement for the matrices P_t.

Specifically the update rule is defined as follows : we first perform the projection step of (1). Then we set P_{t+1} to be the solution of the following optimization problem,

$$P_{t+1} = \arg\min_P \frac{1}{2}\left\|P - P'_{t+1}\right\|^2 \text{ s.t. } \mathrm{Tr}\,(P) \leq r\,, P \text{ is PSD.}$$

We omit the derivation of the solution due to lack of space and proceed with a formal description of it. To solve the optimization problem one needs to compute the eigenvectors $\boldsymbol{v}_j$ of P'_{t+1} and the corresponding eigenvalues $\lambda'_j \geq 0$. To be more concrete we write $P'_{t+1} = \sum_{j=1}^d \lambda'_j \boldsymbol{v}_j\boldsymbol{v}_j^\top$. Then the optimal solution is given by $P_{t+1} = \sum_{j=1}^d \lambda_j \boldsymbol{v}_j\boldsymbol{v}_j^\top$ where

$$\lambda_j = \max\{\lambda'_j - \eta, 0\} \text{ such that } \mathrm{Tr}\,(P_{t+1}) = \sum_j \lambda_j = r\,. \tag{3}$$

Finding the value of η given the set of eigenvalues λ'_j can be computed in $O(d\,log\,d)$ time (see [4]). In a nut shell, the algorithm uses the fact that $\mathrm{Tr}\,(P_{t+1}) = \sum_j \lambda_j$ is monotone piecewise linear function of η. Thus, the algorithm seeks for the optimal value of η in each of the intervals for which the function is linear. Since it takes $O(d^3)$ time to perform eigenvector decomposition the total runtime of the update step is $O(d^3)$.

2.3. Low Rank Implementation

The two steps of the algorithm described above are clearly not practical as they require $O(d^2)$ and $O(d^3)$ time, respectively. We now describe an alternative computation of the above two steps that will allow to naturally enforce a low-rank constraint and efficient computation. We begin by assuming that the current projection is of a low rank, that is $P_t = D_t F_t D_t^\top$ where $D_t \in \mathbb{R}^{d\times r}$ for a small value of r. We assume that D_t is orthonormal $D_t^\top D_t = I_r$ and F_t is positive semi definite. Let us define the matrix $\underline{A}_t = \left[\begin{array}{c|c} D_t & \hat{\boldsymbol{x}}_t \end{array}\right] \in \mathbb{R}^{d\times(r+1)}$ to be the concatenation of D_t and $\hat{\boldsymbol{x}}_t$. We also define a symmetric matrix

$$\underline{C}_t = \left[\begin{array}{c|c} F_t & \boldsymbol{d}_t \\ \hline \boldsymbol{d}_t^\top & f_t \end{array}\right] \in \mathbb{R}^{(r+1)\times(r+1)}\,, \tag{4}$$

$$\begin{array}{rcrl} \boldsymbol{d}_t & = & -\frac{\gamma_t}{2-\gamma_t}\frac{1}{\|\boldsymbol{x}_t\|}F_t\boldsymbol{y}_t & \in \mathbb{R}^r \\ f_t & = & \gamma_t + \frac{\gamma_t^2}{2-\gamma_t}\frac{\boldsymbol{y}_t^\top F_t \boldsymbol{y}_t}{\|\boldsymbol{x}_t\|^2} & \in \mathbb{R} \\ \boldsymbol{y}_t & = & D_t^\top\boldsymbol{x}_t & \in \mathbb{R}^r \end{array} \tag{5}$$

Then it is straightforward to verify that

$$P_{t+1} = \underline{A}_t\,\underline{C}_t\,\underline{A}_t^\top \tag{6}$$

The matrix $\underline{A}_t$ defined above is not orthonormal. We now fix this by projecting the vector $\boldsymbol{x}_t$ on the subspace orthogonal to the range of D_t. Specifically, we define,

$$\boldsymbol{z}_t = \hat{\boldsymbol{x}}_t - D_t D_t^\top \hat{\boldsymbol{x}}_t \quad , \quad \hat{\boldsymbol{z}}_t = \boldsymbol{z}_t/\|\boldsymbol{z}_t\|, \tag{7}$$

where both $\boldsymbol{z}_t, \hat{\boldsymbol{z}}_t \in \mathbb{R}^d$. We use the vector $\hat{\boldsymbol{z}}_t$ to define the new basis matrix

$$\underline{D}_t = \left[\begin{array}{c|c} D_t & \hat{\boldsymbol{z}}_t \end{array}\right] \in \mathbb{R}^{d\times(r+1)}\,. \tag{8}$$

By construction we have that $\underline{D}_t$ is orthonormal, that is $\underline{D}_t^\top\underline{D}_t = I_{r+1}$. Finally, we define the matrix operator which transform from an A_t representation to a D_t representation, $\underline{H} \in \mathbb{R}^{(r+1)\times(r+1)}$ which is the solution of the following equation,

$$\underline{A}_t = \underline{D}_t\,\underline{H}_t\,. \tag{9}$$

It is straightforward to verify that ($\mathbf{0}_r \in \mathbb{R}^r$ is the zero vector),

$$\underline{H}_t = \left[\begin{array}{c|c} I_r & \boldsymbol{g}_t \\ \hline \mathbf{0}_r^\top & h_t \end{array}\right] \in \mathbb{R}^{(r+1)\times(r+1)}\,, \tag{10}$$

$$\boldsymbol{g}_t = D_t^\top\hat{\boldsymbol{x}}_t \in \mathbb{R}^r \quad , \quad h_t = \|\boldsymbol{z}_t\| \in \mathbb{R}, \tag{11}$$

Combining (6) and (9) we get

$$P_{t+1} = \underline{D}_t(\underline{H}_t C_t \underline{H}_t^\top)\underline{D}_t^\top = \underline{D}_t G_t \underline{D}_t^\top\;,\; G_t = \underline{H}_t C_t \underline{H}_t^\top. \tag{12}$$

Finally we compute the eigen-decomposition of G_t,

$$G_t = \underline{V}_t \underline{\Lambda'}_t \underline{V}_t^\top\,, \tag{13}$$

where $\underline{V}_t \in \mathbb{R}^{(r+1)\times(r+1)}$ is orthonormal $\underline{V}_t^\top\underline{V}_t = I_{r+1}$ and $\underline{\Lambda'}_t \in \mathbb{R}^{(r+1)\times(r+1)}$ is diagonal with non-negative elements. Note that since $\underline{V}_t$ and $\underline{D}_t$ are orthonormal, so does their product. Therefore the diagonal elements of the matrix $\underline{\Lambda'}_t$ are exactly the top $r+1$ eigenvalues of P_{t+1}. We assume without loss of generality that the eigenvalues are ordered, that is $\lambda'_{t,1} \geq \lambda'_{t,2} \geq \cdots \geq \lambda'_{t,r+1}$ where $\underline{\Lambda'}_t = \mathrm{diag}(\lambda'_{t,1} \ldots \lambda'_{t,r+1})$. Let

$\underline{\Lambda}_t'$ be the eigenvalues before applying the second stage of the algorithm (3). Then we know that

$$\lambda_{t,q} = \max\{\lambda'_{t,q} - \eta, 0\} \quad \text{for } q = 1 \dots r+1 \ , \tag{14}$$

and the value of η is set such that, $\sum_q \lambda_{t,q} = r$.

By definition the rank of P_{t+1} equals the rank of P_t plus one. At this point we enforce the low-rank constraint and omit the eigenvector of G_t with the smallest eigenvalue. Concretely, we discard the $(r+1)$th column of the product $\underline{V}_t \underline{D}_t$. Let us decompose the matrix $\underline{V}_t$ as following,

$$\underline{V}_t = \left[\begin{array}{c|c} V_t & \boldsymbol{v}_t \end{array} \right] \in \mathbb{R}^{(r+1)\times(r+1)} \ , \tag{15}$$

where V_t are the eigenvectors corresponding the top r eigenvalues and $\boldsymbol{v}_t$ is the vector corresponding to the smallest eigenvalue. We further decompose $\boldsymbol{v}_t$ into two parts: let $\boldsymbol{u}_t$ be the first r components of $\boldsymbol{v}_t$ and a_t be its $(r+1)$th element. Also, let Λ_t be the square sub-matrix of $\underline{\Lambda}_t$ defined by erasing the $(r+1)$th row and $(r+1)$th column of $\underline{\Lambda}_t$. After enforcing the rank constraint we get that, $P_{t+1} = \underline{D}_t \, V_t \, \Lambda_t \, V_t^\top \, \underline{D}_t^\top$. We can stop at this point by defining $D_{t+1} = \underline{D}_t \, V_t$. This update requires $O(d^2 r)$ steps, which we like to avoid.

We now present an efficient decomposition of P_{t+1} which require only $O(dr)$ steps. Similar to Badeau, David and Richard [3], we define the following projection matrix,

$$L_t = \left[\begin{array}{c} I_r - \frac{\boldsymbol{u}_t \boldsymbol{u}_t^\top}{a_t+1} \\ \hline -\boldsymbol{u}_t^\top \end{array} \right] \in \mathbb{R}^{(r+1)\times r} \ . \tag{16}$$

It is readily verified that $L_t^\top L_t = I_r$ and that $L_t^\top \boldsymbol{v}_t = \boldsymbol{0}$. Thus the operator L_t projects into the subspace spanned by V_t (with possibly an additional rotation). We thus write the new projection matrix as, $P_{t+1} = (\underline{D}_t \, L_t) \, (L_t^\top \, V_t \, \Lambda_t \, V_t^\top \, L_t) \, (L_t^\top \, \underline{D}_t^\top)$.

From the last equation we get the recursion rule,

$$D_{t+1} = \underline{D}_t \, L_t \ \ , \ \ F_{t+1} = L_t^\top \, V_t \, \Lambda_t \, V_t^\top \, L_t \tag{17}$$

The matrix F_{t+1} can be computed in time $O(r^3)$. We now show that the matrix D_{t+1} can be computed in time $O(dr)$. Indeed, substituting equations (8) and (16) in (17) we get

$$D_{t+1} = \underline{D}_t \, L_t = D_t - (D_t \boldsymbol{u}_t)\boldsymbol{u}_t^\top/(a_t+1) - \hat{\boldsymbol{z}}_t \boldsymbol{u}_t^\top \ . \tag{18}$$

The algorithm is summarized in Table 1 using the efficient implementation. For convenience its description is presented in four block. In the first block we compute efficiently $P_t \boldsymbol{x}_t$ using two sub-computations $\boldsymbol{y}_t$ and $\boldsymbol{q}_t$. Then the Euclidean distance between the projection $P_t \boldsymbol{x}_t$ and the input vector $\boldsymbol{x}_t$ is computed (ℓ_t). If its value is below the pre-defined threshold ϵ then no update is performed. Otherwise we compute the update of (1) in the second block. We compute the value of γ_t and then compute the matrices C_t , $\underline{D}_t$, $\underline{H}_t$. At this point we can compute the matrix $P_{t+1} = \underline{D}_t \, (\underline{H}_t \, C_t \, \underline{H}_t^\top) \, \underline{D}_t^\top$ (12). Instead of doing so, we perform the regularization step in the third block, by computing the eigen-spectrum of the low dimensional matrix G_t ((12),(12)). Finally, in the fourth block, we efficiently update the projection P_{t+1} by updating the matrices F_{t+1} and D_{t+1} ((18),(17)).

The first two blocks correspond to the first part of the algorithm described in Sec. 2.1. The first block decides if the algorithm should update its projection model D_t, F_t. If an updated occurs then the second block describes the first update step of Sec. 2.1 (equations (1),(2)). The third block corresponds to the regularization step of Sec. 2.2. Finally, the fourth block provides a description of the efficient low-rank update of the matrices D_t, F_t as described in Sec. 2.3.

$\boldsymbol{y}_t = D_t^\top \boldsymbol{x}_t$	(5)	$d\,r$
$\boldsymbol{q}_t = F_t \boldsymbol{y}_t$		r^2
$\ell_i = \\|\boldsymbol{x}_t - D_t \boldsymbol{q}_t\\|^2$		$rd + d$
if $\ell_i \leq 2\epsilon$ then set $D_{t+1} = D_t$; $F_{i+1} = F_t$ else		1
$\psi_i = \boldsymbol{y}_t^\top F_t \boldsymbol{y}_t^\top / \\|\boldsymbol{x}_t\\|^2$		r^2
find γ_t using binary search, see text	(2)	$O(-\log(\delta))$
$\boldsymbol{d}_t = -\frac{\gamma_t}{2-\gamma_t} \frac{1}{\\|\boldsymbol{x}_t\\|} \boldsymbol{q}_t$	(5)	r
$f_t = \gamma_t + \frac{\gamma_t^2}{2-\gamma_t} \psi_i$	(5)	1
$\underline{C}_t = [F_t, \boldsymbol{d}_t; \boldsymbol{d}_t^\top, f_t]$	(4)	
$\boldsymbol{z}_t = \hat{\boldsymbol{x}}_t - D_t \boldsymbol{z}_t$	(7)	r^2
$\hat{\boldsymbol{z}}_t = \frac{\boldsymbol{z}_t}{\\|\boldsymbol{z}_t\\|}$	(7)	r
$\underline{D}_t = [D_t, \hat{\boldsymbol{z}}_t]$	(8)	
$\boldsymbol{g}_t = \boldsymbol{y}_t / \\|\boldsymbol{x}_t\\|$	(11)	r
$h_t = \\|\boldsymbol{z}_t\\|$	(11)	r
$\underline{H}_t = [I_r, \boldsymbol{g}_t; \boldsymbol{0}_r^\top, h_t]$	(10)	
$G_t = \underline{H}_t \, C_t \, \underline{H}_t^\top$	(12)	$2r^3$
$[\underline{V}_t, \underline{\Lambda'}_t] = \text{eig}(G_t)$	(13)	$O(r^3)$
compute $\underline{\Lambda}$	(14)	$O(r\log(r))$
$[\Lambda_t, \boldsymbol{0}; \boldsymbol{0}^\top, \sigma_{t,r+1}] = \underline{\Lambda}$		
$[V_t, \boldsymbol{v}_t] = \underline{V}_t$	(15)	
$[\boldsymbol{u}_t; a_t] = \boldsymbol{v}_t$		
$L_t = \left[I_r - \frac{\boldsymbol{u}_t \boldsymbol{u}_t^\top}{a_t+1}; -\boldsymbol{u}_t^\top\right]$	(16)	$2r^2 + 2r$
$F_{t+1} = L_t^\top \, V_t \, \Lambda_t \, V_t^\top \, L_t$	(17)	r^3
$D_{t+1} = D_t - (D_t \boldsymbol{u}_t)\boldsymbol{u}_t^\top/(a_t+1) - \hat{\boldsymbol{z}}_t \boldsymbol{u}_t^\top$	(18)	$2dr$
		$4dr + O(r^3)$

Table 1: pseudo-code of the CAST algorithm

3. Simulations

We demonstrate the utility of the algorithmic approach described above for subspace tracking and signal reconstruction. We followed the experimental setup of Davila [2] and generated the following signal: $x(t) = \cos(0.3\pi t) + \cos(0.7\pi t + 0.35\pi) + v(t)$ for $t = 1 \dots 999$ and $x(t) = \cos(0.6\pi t) + \cos(0.8\pi t + 0.35\pi) + v(t)$ for $t = 1000 \dots 2000$, where $v(t)$ is Gaussian white noise producing SNR of 40dB in the first experiment and 10dB in the second experiment. From this one dimensional signal we generated a set of vectors defined by, $\boldsymbol{x}_t = x(t)...x(t-d+1)$. We set $d = 50$ and $r = 4$.

We evaluate three algorithms. OPAST [1] and YAST [3] which compute the same signal subspace as SP2 [2]. In both cases we set the forgetting rate to be $\beta = 0.99$ (as in [2]). The third algorithm is CAST presented above. We set $\epsilon = 0.002$ when the noise level was 40dB and $\epsilon = 2$ when the noise level was 10dB. These values reflects the noise level and , in practice, can be evaluated by measuring the difference between a known signal and its noisy version.

Each algorithm was evaluated using two measures. The first measure is, $\delta_t = \|D_t(D_t^\top D_t)^{-1} D_t^\top - D_t(D_t^\dagger D_t^{\dagger\top})^{-1} D_t^\dagger\|_F^2$ [2], where $D_t^\dagger$ is the best signal subspace out of the two subspaces corresponding the two sinusoidal signals. The second evaluation-measure computes the cumulative sum of the reconstruction error of the input signal $x(t)$. Concretely, we first projected the input vector $\boldsymbol{x}_t$ given at time t on the instantaneous subspace defined by $D_t D_t^\top$ of time t. Then, we used this sequence of projections to reconstruct the original one-dimensional signal $x(t)$ by averaging the corresponding components of the projection. Denote by $\hat{s}(t)$ the reconstructed signal. The evaluation measure at time t is given by $E_t = \sum_\tau^t (s(\tau) - \hat{s}(\tau))^2$ where $s(t)$ is the true signal, without any noise added.

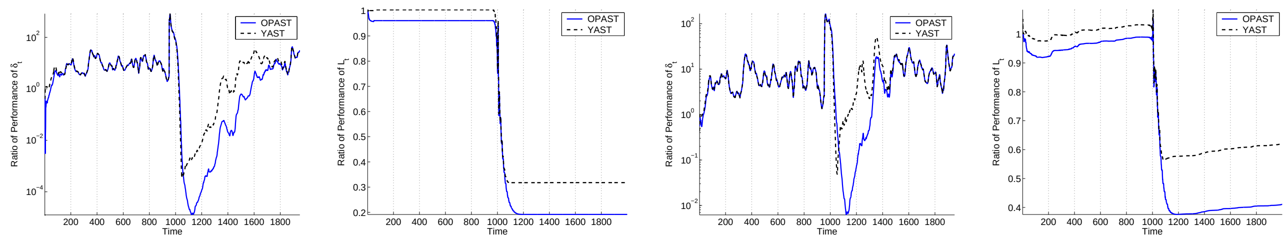

Figure 1: Simulations with SNR of 40dB (two left plots) and 10dB (two right plots). Two evaluation measures δ_t (1st and 3rd from left) and E_t (2nd and 4th from right). Two algorithms CAST/OPAST (solid line) and CAST/YAST (dashed line). The CAST algorithm outperforms other algorithm for values below 1 (or 10^0).

The results are summarized in Fig. 1. In all cases we show the ratio of the performance of CAST with either OPAST or YAST. The two left hand-side plots summarize the results for SNR 40dB and the two hand-side plots summarize the results for SNR 10dB. The first and third plots (from the left) use the first evaluation measure δ_t and the second and fourth plots use the second evaluation measure E_t.

We first compare the results using δ_t. In both noise levels, where there is no transition, CAST recovers the target subspace with error about 10 times worse than OPAST and YAST. Note that in absolute terms the δ_t-error of CAST is small and equals about 10^{-4} for the low noise level and about $.01 - .05$ for the high noise level. When transition occurs we note two advantages of CAST. First, it identifies the transition before the two other algorithms. This can be indicated by the small maxima just before $t = 1000$. Second, it recovers from the transition faster than the other algorithms. This phenomena can be indicated by the *wide* minima around $t = 1100$. The performance of YAST is better than the performance of OPAST [3].

These observations are supported by examining the cumulative reconstruction loss E_t. Before the transition, and after it, all algorithms reconstruct the signal with about the same accuracy, thus the inferiority of CAST to identify the target subspace is not reflected in its de-noising performance. Furthermore, during the transition the cumulative reconstruction loss of CAST is about $0.3 - 0.5$ of the reconstruction loss of the other algorithms. Again, this indicates that it tracks the new subspace faster then the other algorithms.

We define the final reconstruction error (FRE) for each of the algorithms to be $10\log_{10}((\sum_t^T s(t)^2)/E_T)$, where T is the length of the signal. For the low noise level of 40dB we get a FRE of 12dB for OPAST, 14dB for YAST, 19dB for CAST. Thus, all algorithms increases the noise, but CAST increase the noise is the most moderate way. Note that for all algorithms, if we compute the FRE using only the prefix of size 900 (before the transition) we get a shared value of 21dB for all algorithms. For the high noise levels the FRE values are 11dB for OPAST, 13dB for YAST, 15dB for CAST. Thus CAST reduces the noise level by 5dB, which is better the OPAST and YAST. Again, the FRE computed on the prefix of length 900 is about 18dB for all algorithms.

We computed the orthonormality error at time t for each of the algorithms, $10\log_{10}(\|D_t^\top D_t - I_r\|_2^2)$. The maximal value for OPAST was about -300dB and for YAST about -286dB (for both noise levels). For CAST the maximal value was -138dB (SNR 40dB) and -205dB (SNR 10dB), both are fairly low values. It is not clear why the noise level influences the orthonormality error for CAST, which we still investigating.

The complexity of OPAST is $4dr + O(r^2)$ [1], of YAST is $7dr+O(r^2)$ [3] and of CAST is $4dr+O(r^3)$. (Note that YAST is designed mainly for temporal data, while the two other algorithms are general.) Finally, the complexity of both OPAST and YAST is *fixed* independent of the value of the parameter β, and all parts of the algorithms are executed for any given input. For CAST, only the first stage (first block in Table 1) of the algorithm is executed for all input (complexity $2dr$), the other three steps (last three blocks in Table 1) are executed conditioning the outcome of the first state. In our simulations these stages were executed in about 0.85 of the rounds. Thus, the actual complexity was about $2dr + 0.85 \times 2dr = 3.7dr$. In general, we can tradeoff accuracy vs efficiency using the parameter ϵ. For high values of ϵ the algorithm will run faster but produce less accurate subspaces (and the reconstruction error will be higher). On the other hand, low values of ϵ can be used to compute subspaces more accurately, but will involve more operations. A natural lower bound of ϵ is around the noise level. Below this value the algorithm will start tracking the noise and not the underling subspace. As an example, we ran the algorithm with a double value of ϵ. The performance was reduced, but the last three stages were performed for only 0.1 of the rounds, leading to a complexity of $2dr + 0.2dr$. Thus, when the noise level is relatively small we can track the underling subspace efficiently by setting ϵ to a relatively high value.

4. Summary

We presented an algorithm for subspace tracking in general, and data de-noising in particular, based on ideas from machine learning. The system has three steps: projection, regularization and enforcing low-rank. The sensitivity parameter ϵ can be used naturally to trade-off accuracy vs. efficacy. The simulations results indicate the usefulness of the approach, especially during subspace transitions. Currently, we are investigating other alternatives which will perform even better in between transitions.

5. References

[1] K. Abed-Meraim, A. Chkeif, and Y. Hua, "Fast orthonormal past algorithm," *IEEE Signal Proc. Let.*, vol. 7, 2000.

[2] C. Davila, "Efficient, high performance, subspace tracking for time-domain data," *IEEE Tran. on Signal Processing*, 2000.

[3] R. Badeau, B. David, and G. Richard, "Yet another subspace tracker," in *ICASSP*, 2005.

[4] K. Crammer, "Online tracking of linear subspaces," in *COLT*, 2006.

[5] K. Crammer, O. Dekel, S. Shalev-Shwartz, and Y. Singer, "Online passive aggressive algorithms," in *NIPS 16*, 2003.