SPARSE-BEFAP : A FAST IMPLEMENTATION OF FAST AFFINE PROJECTION AVOIDING EXPLICIT REGULARISATION

Geert Rombouts, Marc Moonen

KULeuven/ESAT-SISTA
Kardinaal Mercierlaan 94
3001 Heverlee, Belgium
geert.rombouts@esat.kuleuven.ac.be
marc.moonen@esat.kuleuven.ac.be

ABSTRACT

It is well known that regularisation of the covariance matrix is necessary in Affine Projection Algorithm (APA) based adaptive filters. A technique to achieve regularisation is using non–successive equations in the system of equations that is solved in the APA–algorithm, leading to the Sparse–APA algorithm [5][4]. In this paper, a fast exact frequency domain implementation of Sparse–APA is derived which does not rely on any of the approximations which are found in the classical fast affine projection algorithms, while only a very small number of extra computations is needed.

1 INTRODUCTION

The so–called affine projection algorithm (APA) has become a popular tool in adaptive filtering, especially in the context of acoustic echo cancellation. In a practical acoustic echo canceller for instance, adaptation is switched off when near end talk is present. In addition however, regularisation is indispensible to make APA–based algorithms robust to continuously present near end noise, e.g. people talking in the background. This near end noise will be amplified by the inverse of the correlation matrix of the input signal [2], so if its condition number is bad, a large influence on the adaptation will result. The defining equations for plain APA are

$$
\begin{aligned}
X_k &= \left[\begin{array}{cccc} \mathbf{x}_k & \mathbf{x}_{k-1} & \dots & \mathbf{x}_{k-P} \end{array}\right] && (1)\\
\mathbf{x}_i &= \left[\begin{array}{cccc} x_i & x_{i-1} & \dots & x_{i-N} \end{array}\right]^T && (2)\\
\mathbf{d}_k &= \left[\begin{array}{cccc} d_k & d_{k-1} & \dots & d_{i-P} \end{array}\right]^T && (3)\\
\mathbf{e}_k &= \mathbf{d}_k - X^T \mathbf{w}_{k-1} && (4)\\
\mathbf{g}_k &= [\Xi_k + \delta I]^{-1} \mathbf{e}_k && (5)\\
\Xi_k &= X_k^T X_k && (6)\\
\mathbf{w}_k &= \mathbf{w}_{k-1} + \mu X_k \mathbf{g}_k && (7)
\end{aligned}
$$

The d_k are the desired signal samples, and the x_k are the reference signal samples. The filter coefficients are contained in $\mathbf{w}_k$, Ξ_k is the reference signal's correlation matrix. The filter length is N, the order of the projection algorithm is P, μ is the step size. Here explicit regularisation is applied by adding δI to the covariance matrix before inverting it.

The complexity of APA is $O(PN)$ where N is the filter length (e.g. 1000 to 2000 taps), and P (e.g. 10) the order of APA. The fast affine projection algorithm (FAP) [1] does not calculate the full error vector, but instead calculates the first component, and uses an approximation for the others :

$$
\mathbf{e}_k^{\text{fap}} = \left[\begin{array}{c} d_k - \mathbf{x}_k^T \mathbf{w}_{k-1} \\ (1-\mu)\mathbf{e}_{k-1}^{\text{fap}}|_1^{P-1} \end{array}\right]
$$

Here, $\mathbf{e}_{k-1}^{\text{fap}}|_1^{P-1}$ denotes the vector that consists of the first $P-1$ components of $\mathbf{e}_{k-1}^{\text{fap}}$. FAP has a complexity of $2N + 20P$. A block exact frequency domain version of FAP derived in [3] and in [6] (BEFAP) brings the complexity further down to less than half of this, namely:

$$
\begin{aligned}
&\frac{6M_1 \log_2 M_1 - 7M_1 - 31}{N_1} + 6N_2 + 15P - 4 + \\
&\frac{P^2 - P}{N_2} + 10P^2 + \frac{6M_1 \log_2 M_2 - 7M_2 - 31}{N_2} \qquad (8)
\end{aligned}
$$

Where N_1 and N_2 are block lengths for the frequency domain algorithm (e.g. $N_1 = N_2 = 128$). Furthermore $M_i = N + N_i - 1$. The term $\frac{P^2-P}{N_2}$ can often be neglected because $P \ll N_2$. A typical example is $P = 3,\ N_1 = N_2 = 128,\ N = 800 \Rightarrow 1654$ flops per sample.

A technique which does not use successive equations for solving the APA–problem, but subsamples a set of PD successive equations, using only one out of every D (e.g. 3), is derived in [5] and in [4]**.** As is shown in the latter paper, this sparse–APA–algorithm provides a valid alternative to explicit regularisation by leaving out correlated equations

(its performance is better than that of a plain APA algorithm of order P). In this paper, a fast implementation of this algorithm (Sparse–BEFAP) is derived, based on [6]. To solve sparse–APA we have to calculate all components of the APA residual vector in each step instead of only the first one. We describe a method which doesn't require a full filtering operation for all of the P equations, but which re-uses quantities that have been calculated in previous steps. The complexity of the new algorithm will be

$$\frac{6M_1 \log_2 M_1 - 7M_1 - 31}{N_1} + 6N_2 + 13P + 2PD - 4 + 10P^{2+} \frac{P^2D^2 - PD}{N_2} + \frac{6M_2 \log_2 M_2 - 7M_2 - 31}{N_2} + P^2D - D$$

This formula shows that even though all P components of the error vector are calculated, the number of flops does not increase significantly. For the setting of $P = 3,\ N_1 = N_2 = 128,\ N = 800,\ D = 3 \Rightarrow 1690$ flops per sample are required.

2 SPARSE–APA

For sparse–APA, the matrix X_k in the equations 1 is replaced by a matrix $\widetilde{X}_k = [\mathbf{x}_k\ \mathbf{x}_{k-D}\ \ldots\ \mathbf{x}_{k-(P-1)D}]$, where D is a distance between the equations. Vector $\mathbf{d}_k$ is replaced by $\tilde{\mathbf{d}}_k = \left[\ d_k \quad d_{k-D} \quad \ldots \quad d_{i-(P-1)D}\ \right]^T$. In [4], we have shown that leaving out equations to obtain sparse–APA has a beneficial effect on the condition number of the covariance matrix, which is indispensable to keep the near end noise amplification of the system as small as possible. Figure 1 shows that the largest and smallest singular values of the correlation matrix are closer together (full line) when sparse equations are used than when appropriate explicit regularisation is applied on successive equations. In [2] it is shown that the smallest singular value has to be large to avoid near end noise amplification.

3 A FAST IMPLEMENTATION

3.1 Principle

A block exact frequency domain version of FAP, derived in [3] and in [6] (Block Exact FAP), forms the basis of the derivation of our algorithm. In this algorithm a *basis filter* is used which is constant over a block, and thus can be implemented in the frequency domain, and afterwards *corrections* are added to each of the basis filter outputs in the block. The difference with [6] is the fact that here *all* the components of the error vector are calculated, and that the equations are not successive, but spaced by distance D. If the filter vector *were* updated in *each* step and the basis filter vector $\mathbf{w}_{k-1}$ were known at time instant k, we can write $\mathbf{w}_{k+i-1}$ in terms of $\mathbf{w}_{k-1}$.

$$\widetilde{\mathbf{y}}_{k+i} = \widetilde{X}_{k+i}^T \widetilde{\mathbf{w}}_{k+i-1}$$

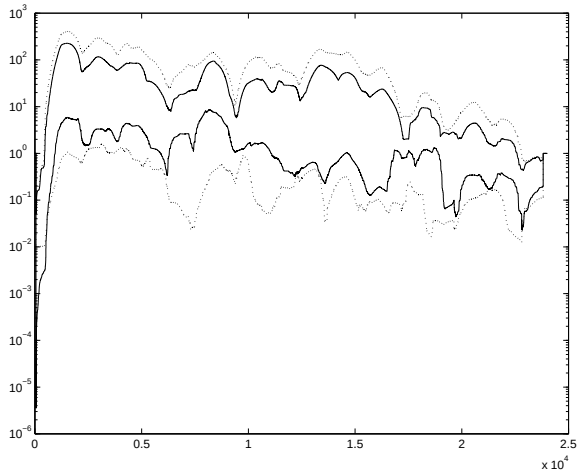

Figure 1: Smallest and largest singular value of input matrix on a logarithmic scale. Dotted line : explicit regularisation ($\delta = 0.01$) with successive equations. Full line : dispersed equations ($D = 5$) without explicit regularisation. Signal peak value = 0.1. Both times, 10 equations were considered.

$$\widetilde{\mathbf{e}}_{k+i} = \widetilde{\mathbf{d}}_{k+i} - \widetilde{\mathbf{y}}_{k+i}$$

$$\widetilde{\mathbf{g}}_{k+i} = (\widetilde{X}_{k+i}^T \widetilde{X}_{k+i})^{-1} \widetilde{\mathbf{e}}_{k+i}$$

$$\widetilde{\mathbf{w}}_{k+i-1} = \widetilde{\mathbf{w}}_{k-1} + \sum_{j=1}^{i} \mu \widetilde{X}_{k+i-j} \widetilde{\mathbf{g}}_{k+i-j} \qquad (9)$$

We let $s_{k,j}$ denote the j'th component of vector $\mathbf{s}_k$. In a similar way as was done for BEFAP in [6], we can write

$$\widetilde{\mathbf{w}}_{k+i-1} = \widetilde{\mathbf{w}}_{k-1} + \sum_{j=1}^{i+PD-1} \widetilde{s}_{k+i-1,j} \mathbf{x}_{k+i-j} - \sum_{j=1}^{PD-1} \widetilde{s}_{k-1,j} \mathbf{x}_{k-j}$$

where the vector $\widetilde{\mathbf{s}}_{k+i-1}$ is recursively obtained from $\widetilde{\mathbf{s}}_{k+i-2}$. In the beginning of each block, the size of vector $\widetilde{\mathbf{s}}_k$ is reset. The recursion is

$$\widetilde{\mathbf{s}}_k = \begin{pmatrix} 0 \\ \widetilde{\mathbf{s}}_{k-1}|_1^{PD-1} \end{pmatrix} + \mu \widetilde{\widetilde{\mathbf{g}}}_k \qquad (10)$$

$$\widetilde{\mathbf{s}}_{k+i-1} = \begin{pmatrix} 0 \\ \widetilde{\mathbf{s}}_{k+i-2} \end{pmatrix} + \mu \begin{pmatrix} \widetilde{\widetilde{\mathbf{g}}}_{k+i-1} \\ 0_{N_2-1} \end{pmatrix} \qquad (11)$$

$$\widetilde{\widetilde{\mathbf{g}}}_{k+i-1} = \begin{pmatrix} \widetilde{g}_{k+i-1,1} \\ 0_{D-1} \\ \widetilde{g}_{k+i-1,2} \\ 0_{D-1} \\ \vdots \\ \widetilde{g}_{k+i-1,P} \\ 0_{D-1} \end{pmatrix}$$

The filter outputs $y_{k+i,\alpha}$ can now be written as

$$\begin{aligned} y_{k+i,\alpha} &= \mathbf{x}^T_{k+i-(\alpha-1)D}\widetilde{\mathbf{w}}_{k+i-1} \\ &= \mathbf{x}^T_{k+i-(\alpha-1)D}\widetilde{\mathbf{w}}_{k-1} + \\ &\quad \sum_{j=1}^{i+PD-1} \widetilde{s}_{k+i-1,j}\mathbf{x}^T_{k+i-(\alpha-1)D}\mathbf{x}_{k+i-j} - \\ &\quad \sum_{j=1}^{PD-1} \widetilde{s}_{k-1,j}\mathbf{x}^T_{k+i-(\alpha-1)D}\mathbf{x}_{k-j} \\ &= \mathbf{x}^T_{k+i}\widetilde{\mathbf{w}}_{k-1} + \sum_{j=1}^{i+PD-1} \widetilde{s}_{k+i-1,j}r^{\alpha}_{k+i,j} - \\ &\quad \sum_{j=1}^{PD-1} \widetilde{s}_{k-1,j}r^{\alpha}_{k+i,i+j} \end{aligned}$$

The correlations are defined as

$$r^{\alpha}_{k+i,j} \equiv \mathbf{x}^T_{k+i-(\alpha-1)D}\mathbf{x}_{k+i-j}$$

We proceed (similar to [6]) by defining a modified filter vector

$$\widetilde{\mathbf{z}}_{k-1} = \widetilde{\mathbf{w}}_{k-1} - \sum_{j=1}^{PD-1} \widetilde{s}_{k-1,j}\mathbf{x}_{k-j}$$

then the filter output can be written as

$$y_{k+i,\alpha} = \mathbf{x}^T_{k+i-(\alpha-1)D}\widetilde{\mathbf{z}}_{k-1} + \widetilde{\mathbf{s}}^T_{k+i-1}\mathbf{r}^{\alpha}_{k+i} \qquad (12)$$

in which both $\widetilde{\mathbf{s}}_{k+i-1}$ and $\mathbf{r}^{\alpha}_{k+i}$ are vectors of length $i+PD-1$. The autocorrelation vector $\mathbf{r}^{\alpha}_{k+i}$ is needed to calculate the correction for the α'th component of $\mathbf{y}_{k+i}$ (which is needed in turn to calculate the α'th component of the residual vector $\widetilde{\mathbf{e}}_{k+i}$. The first term of this equation is a filtering operation with a filter vector $\widetilde{\mathbf{z}}_{k-1}$ that is fixed over a block of size N_2, an that is independent of α, so it can easily be performed in the frequency domain. The second term is growing inside each block. A recursion for the new filter vector can also be derived along the lines of [6], which gives

$$\begin{aligned} \mathbf{z}_{k+N_2-1} &= \mathbf{z}_{k-1} + \overline{X}_{k+N_2-PD}\mathbf{s}_{k+N_2-1}|^{PD}_{PD+N_2-1} \\ \overline{X}_{k+N_2-PD} &= \begin{bmatrix} \mathbf{x}_{k+N_2-PD} & \mathbf{x}_{k+N_2-PD-1} & \cdots & \mathbf{x}_{k-PD+1} \end{bmatrix} \end{aligned}$$

where $\mathbf{s}|^j_i$ is a sub–vector consisting of the i'th through j'th element of $\mathbf{s}$. Here to, the matrix–vector product can be calculated in the frequency domain with fast convolution techniques (block size N_2).Concerning the computational complexity, the only difference with the original BEFAP–algorithm is that all calculations are repeated P times here, because we want to compute all error vector components. However, we will show how the number of required flops for the second to the P'th component can be drastically reduced.

3.2 Re-using the correlation coefficients

It can be shown that following recursions hold

$$r^{\alpha}_{(k+i),\beta} = r^{\alpha-1}_{(k+i-D),((D+1)-(\beta-1))} \quad \text{for} \quad \beta \leq D+1, \alpha > 1 \qquad (13)$$

$$r^{\alpha}_{(k+i),\beta} = r^{\alpha-1}_{(k+i-D),(\beta-D)} \quad \text{for} \quad \beta \geq D+1, \alpha > 1 \qquad (14)$$

Since memory is as expensive as processing power, this recursion seems to be no big advantage, because there would be $N_2 + PD$ delay lines of length D needed to re–use the correlation vector components that were calculated in the previous time step. But we can build on this recursion to apply a major complexity reduction.

3.3 Re-using vector products from previous steps

The vector product $\sum_{j=1}^{i+PD-1} s_{(k+i-1),j}r^{\alpha}_{(k+i),j}$ can be written as

$$\begin{aligned} \mathbf{s}^T_{(k+i-1)}\mathbf{r}^{\alpha}_{(k+i)} &= \overline{\mathbf{s}}^T_{(k+i-1)}\widetilde{\mathbf{r}}^{\alpha}_{(k+i)} \\ &+ \mathbf{s}^T_{(k+i-D-1)} \begin{pmatrix} \mathbf{r}^{\alpha-1}_{(k+i-D),1} \\ \mathbf{r}^{\alpha-1}_{(k+i-D),2} \\ \vdots \\ \mathbf{r}^{\alpha-1}_{(k+i-D),i+PD} \end{pmatrix} \end{aligned} \qquad (15)$$

The vector $\widetilde{\mathbf{r}}^{\alpha}_{(k+i)}$ is formed from the first $D+PD-1$ components of $\mathbf{r}^{\alpha}_{(k+i)}$. The last term of equation 15 (a scalar) has already been calculated in step $k+i-D$. So in each step, one still needs to calculate the 'large' vector product for the correction to the first component of the error vector as in BEFAP, and $P-1$ small vector products (size $PD+D$) since the second term from equation 15 can be fed through a delay line. For the calculations of the corrections for the first error vector component, this gives an average of $\frac{(PD+N_2-1)(PD+N_2)}{2N_2}$flops per sample, where N_2 is the block size in the BEFAP–algorithm. To this, $(PD+D)$ flops per sample for each of the $P-1$ remaining components of the error vector must be added. If not too large values are chosen for P and D (e.g. setting $P=3$ and $D=3$ already result in an important performance increase) this is a lot less complex than calculating all the components.

In this setting, one is free to choose if the $\widetilde{\mathbf{r}}^{\alpha}_{(k+i)}$ are taken from previous steps as described in equations 13 and 14, or if they are recalculated at the time that they are needed (by up– and downdates). The latter method requires less memory (for delay lines), but more flops.

4 IMPLEMENTATION

A complete algorithm specification is given below. We let $\mathbf{u} = [x_1, x_2, ...]^T$be the input signal, and $\mathbf{v} = [d_1, d_2, ...]^T$ the desired signal, in both of which the order of the samples is different from the definitions of $\mathbf{x}$ and $\mathbf{d}$. A right to left arrow above a vector flips the order of the components. The full algorithm then is as follows

for j = 1 to $N_2 + PD$
$r_j^1 = (\mathbf{u}|_{k-L+1}^{k})^T \mathbf{u}|_{k-j-L+1}^{k-j}$
endfor
for α = 2 to P
for j = 1 to $D + PD$
$r_j^\alpha = (\mathbf{u}|_{k-L-\alpha D+1}^{k-\alpha D})^T \mathbf{u}|_{k-j-L+1}^{k-j}$
endfor
endfor
loop
for i = 0 to $N_2 - 1$
if (i modulo N_1 == 0)
<fill next part of **y** with convolution of next block of N_1 samples from **u** with **z**>
endif
$\mathbf{r}^1 = \mathbf{r}^1 + u_k \overleftarrow{\mathbf{u}|_{k+i+1}^{k+i+1-N_2-PD+1}} - u_{k-L+i+1} \overleftarrow{\mathbf{u}|_{k-L+i+1}^{k-L+i+1-N_2-PD+1}}$
for α = 2 to P
$\mathbf{r}^\alpha = \mathbf{r}^\alpha + u_{k+i-\alpha D+1} \overleftarrow{\mathbf{u}|_{k+i-1}^{k+i+1-(D+PD)+1}} - u_{k-L+i-\alpha D+1} \overleftarrow{\mathbf{u}|_{k-L+i+1}^{k-L+i+1-(D+PD)+1}}$
endfor
$A_{k+i+1+D}^1 = (\mathbf{s}|_1^{PD+i-1})^T \mathbf{r}^1|_2^{i+PD}$
$e_{k+i+1} = v_{k+i+1} - y_{k+i+1} - A_{k+i+1+D}^1$
$E_1 = e_{k+i+1}$
for α = 2 to P
$A_{k+i+1+D}^\alpha = A_{k+i+1}^{\alpha-1} + (\overline{\mathbf{s}}^D)^T \mathbf{r}^\alpha|_2^{D+PD}$
$E_\alpha = v_{k+i-\alpha D+1} - y(k+i-\alpha D+1) - A_{k+i+1+D}^\alpha$
endfor
<update S^{-1}, the $P \times P$ inverse covariance matrix>
<spread the result of $S^{-1}E$ into $\widetilde{\mathbf{g}}$>
for m = D downto 2
$\overline{\mathbf{s}}^m = \begin{bmatrix} 0 \\ \overline{\mathbf{s}}^{m-1}|_1^{D+PD-2} \end{bmatrix} + \begin{bmatrix} \mu\widetilde{\mathbf{g}} \\ 0_{D-1} \end{bmatrix}$
endfor
$\overline{\mathbf{s}}^1 = \begin{bmatrix} \mu\widetilde{\mathbf{g}} \\ 0_{D-1} \end{bmatrix}$
$\mathbf{s} = \begin{bmatrix} 0 \\ \mathbf{s}|_1^{N_2+PD-2} \end{bmatrix} + \begin{bmatrix} \mu\widetilde{\mathbf{g}} \\ 0_{N_2-1} \end{bmatrix}$
endfor
$\mathbf{z} = \mathbf{z}$+<convolution of $\mathbf{s}|_{PD}^{PD+N_2-1}$ with $\mathbf{u}|_{k-PD+N_2+1-L+1+1}^{k-PD+N_2+1}$>
$k = k + N_2$
endloop

5 CONCLUSION

In this paper, we have derived a fast version of the sparse–APA algorithm, with a resulting complexity roughly equal to the BEFAP–complexity [6]. To be able to implement this sparse–BEFAP–algorithm, we had to calculate all components of the error vector instead of only the first one as is done in [1]. Our algorithm can do this, with only a small number of extra flops.

6 ACKNOWLEDGEMENTS

Marc Moonen is a Research Associate with the F.W.O. (Fund for Scientific Research Flanders). Geert Rombouts is a Research Assistant with the I.W.T. (Flemish Institute for Scientific and Technological Research in Industry). This research work was carried out at the ESAT laboratory of the Katholieke Universiteit Leuven, in the frame of the Belgian State, Prime Minister's Office - Federal Office for Scientific, Technical and Cultural Affairs - Interuniversity Poles of Attraction Programme - IUAP P4-02 (1997-2001): Modeling, Identification, Simulation and Control of Complex Systems, the Concerted Research Action GOA-MEFISTO-666 (Mathematical Engineering for Information and Communication Systems Technology) of the Flemish Government, Research Project FWO nr. G.0295.97 ('Design and implementation of adaptive digital signal processing algorithms for broadband applications'), IWT project AUT/970517: MUSETTE ('Multimicrophone Signal Enhancement Techniques for handsfree telephony and voice controlled systems'), and was partially sponsored by Philips-ITCL. The scientific responsibility is assumed by its authors.

REFERENCES

[1] S. L. Gay and S. Tavathia. The fast affine projection algorithm. In *ICASSP*, pages 3023–3026. IEEE, 1995.

[2] Steven Gay. *Fast projection algorithms with application to voice echo cancellation*. PhD thesis, Graduate School, New Brunswick, 1994.

[3] K. Maouche and D. T. M. Slock. The fast subsampled–updating fast affine projection (fsu fap) algorithm. Research report, Institut EURECOM, 2229, route des Cretes, B.P.193, 06904 Sophia Antipolis Cedex, December 1994.

[4] G. Rombouts and M. Moonen. Avoiding explicit regularisation in affine projection algorithms for acoustic echo cancellation. In *Proceedings of ProRISC99, Mierlo, The Netherlands*, 1999.

[5] S. G. Sankaran and A. A. (Louis) Beex. Normalized lms algorithm with orthogonal correction factors. In *Proceedings of the Thirty–First annual Asilomar Conference on Signals, Systems, and Computers*, pages 1670–1673, November 1997.

[6] M. Tanaka and S. Makino. A block exact fast affine projection algorithm. *IEEE Transactions on Speech and Audio Processing*, 7(1):79–86, January 1999.