### *Affine Projection Algorithms (APA) or Generalized NLMS (GNLMS)*

**Comments**

1) Due to the possibility of a near-singularity of $\mathbf{U}^H(n)\mathbf{U}(n)$, in practice we use

$$\boxed{\hat{\mathbf{w}}(n+1) = \hat{\boldsymbol{\omega}}(\nu) + \tilde{\mu}\mathbf{Y}(\nu)[\mathbf{Y}^H(\nu)\mathbf{Y}(\nu) + \beta\mathbf{I}]^{-1}\boldsymbol{\varepsilon}^H(\nu)}$$

where β is a small constant.

2) Choose $0 < \tilde{\mu} < 2$ for convergence in the mean square (Sankaran and Beex).

3) Algorithm has a fundamental performance advantage by which rapid convergence is obtained for correlated signals that can be modeled by an AR process of order less than or equal to *N*. Intuitively, we see that our correction takes into account *N* previous input signal *vectors*.

4) APA converges faster than NLMS; as more inputs vectors are used (larger *N*), the convergence rate itself improves. If the input is white, the learning curse is linear, and the MSE drops by 20dB in about 5 *M* / (*N* + 1) iterations.

5) GNLMS requires an $N \times N$ matrix inverse which is determined by the projection order.

6) Originally derived geometrically by projecting the misalignment vector onto a subspace of dimension $M - N$ determined by the *N* input signal vectors and minimizing the a posteriori estimation error. See Haykin 4th Ed for example. In the literature, *N* is called the projection order and typically $N < M$.

7) Little performance is sacrificed if $\hat{\mathbf{w}}(n)$ is updated only every *N* samples instead of every sample (called a block update). In this case we need only perform matrix inversion every *N* samples thereby realizing significant computational savings.

8) Assuming a block update, we have a per sample complexity of $\boldsymbol{O}(NM)$ vs. $\boldsymbol{O}(M^2)$ as would be the case in computing the Wiener solution for each sample. As *N* varies from 1 to *M* we have a graceful transition from NLMS (low complexity, slow convergence) to full-rank methods i.e. Wiener (high complexity, fast convergence).

9) A "fast" version of GNLMS, known as fast APA (FAP), exists. Complexity is approximately $\boldsymbol{O}(2M)$ assuming a block update, and computational load is uniformly distributed in time. Load distribution provides efficient implementation on signal processor architectures.

---

### *Children of the LMS Algorithm*

There are many variation on the basic LMS algorithm each enhanced for better convergence in a particular application or reduced complexity. The NLMS and GNLMS algorithms are two offshoots of LMS.

---

### *Algorithms with Better Convergence Properties*

**LMS**

$$e(n) = d(n) - \hat{\mathbf{w}}^H(n)\mathbf{u}(n)$$
$$\hat{\mathbf{w}}(n+1) = \hat{\mathbf{w}}(n) + \mu\mathbf{u}(n)e^*(n)$$

**NLMS**

$$e(n) = d(n) - \hat{\mathbf{w}}^H(n)\mathbf{u}(n)$$

$$\hat{\mathbf{w}}(n+1) = \hat{\mathbf{w}}(n) + \frac{\tilde{\mu}\mathbf{u}(n)e^*(n)}{a + \left\|\mathbf{u}(n)\right\|^2}$$

Comment: Update is not sensitive to input power variations.

**GNLMS**

$$e(n-k) = d(n-k) - \hat{\mathbf{w}}^H(n)\mathbf{u}(n-k)$$

$$\mathbf{e}(n) = [e(n) \quad e(n-1) \quad \text{L} \quad e(n-M+1)]$$

$$\mathbf{U}(n) = [\mathbf{u}(n) \quad \mathbf{u}(n-1) \quad \text{L} \quad \mathbf{u}(n-N+1)]$$

$$\hat{\mathbf{w}}(n+1) = \hat{\mathbf{w}}(n) + \tilde{\mu}\mathbf{U}(n)[\mathbf{U}^H(n)\mathbf{U}(n) + \beta\mathbf{I}]^{-1}\mathbf{e}^H(n)$$

Comment: Increased rate of convergence when modeling an AR process of order *N*.

---

### *Other LMS-like Algorithms*

**Leaky LMS**

Instead of minimizing the MSE (cost function) we instead minimize

$$J(n) = \left|\varepsilon(\nu)\right|^2 + \alpha\left\|\hat{\boldsymbol{\omega}}(\nu)\right\|^2$$

with respect to $\hat{\mathbf{w}}(n)$ where $|\alpha| < 1$. This leads to the update equation

$$\hat{\mathbf{w}}(n+1) = (1 - \mu\alpha)\hat{\boldsymbol{\omega}}(\nu) + \mu\boldsymbol{\upsilon}(\nu)\varepsilon^*(\nu)$$

Comments:
1) Prevents filter coefficients from growing large (and hence algorithm instability) by "leaking" some of the coefficient away.
2) Useful in adaptive sidelobe cancelers, i.e. adaptive antennae arrays which can null out an interference or jamming signal.

**Block LMS**

Instead of updating the weight vector every sample as in the LMS algorithm, we may choose to update every *L* samples. In this case rather than base our correction on $\mathbf{u}(n)e^*(n)$ as in LMS, we'll base the correction on $\mathbf{u}(n)e^*(n) + \mathbf{u}(n-1)e^*(n-1) + \ldots + \mathbf{u}(n-L+1)e^*(n-L+1)$ and hold the weight vector fixed over each block of data.

This leads to the update equation for the block LMS

$$\hat{\mathbf{w}}(k+1) = \hat{\boldsymbol{\omega}}(\kappa) + \mu\sum_{\iota=0}^{M-1}\boldsymbol{\upsilon}(\kappa\Lambda + \iota)\varepsilon^*(\kappa\Lambda + \iota)$$

where the sample index $n = \kappa\Lambda + \iota$, $i = 0, 1, \ldots, M-1$ and *k* is the block number.

Comments: Rather than use an instantaneous gradient vector estimate (LMS) we use an averaged gradient vector estimate (averaged over *L* samples). The averaged gradient vector estimate is more accurate and accuracy of course increases with increasing *L*. Increased accuracy does not increase convergence rate but leads to a "smoother" update and error signal.

**Skewed Block LMS**

In the case of block LMS most of our computational effort is expended every $L$ samples. In order to distribute the computational load (as would be necessary in a real-time application) we may choose to partition the weight vector coefficients into $L$ subsets and update each subset every $L$ samples. In this case our computation load is evenly distributed over time and the entire weight vector is updated every $L$ samples as in the block LMS.

Consider the following $p^{\text{th}}$ weight vector partition

$$\hat{\mathbf{w}}_p = [\, \omega_\pi \quad \omega_{\pi+\Lambda} \quad \Lambda \,]^T$$

where $p = 0, 1, ..., L-1$. The update equation for the skewed block LMS is

$$\hat{\mathbf{w}}_p(k+1) = \hat{\boldsymbol{\omega}}_\pi(\kappa) + \mu \sum_{\iota=0}^{M-1} \boldsymbol{\upsilon}_\pi(\kappa\Lambda+\iota)\, \varepsilon^*(\kappa\Lambda+\iota)$$

where the sample index

, $i = 0, 1, ..., M - 1$ and $k$ is the block number.

**Sign-Error Algorithm**

We modify the LMS algorithm by basing our update on the sign of the $e(n)$. We have

$$\hat{\mathbf{w}}(n+1) = \hat{\boldsymbol{\omega}}(\nu) + \mu \boldsymbol{\upsilon}(\nu)\,\mathrm{sgn}[\varepsilon(\nu)]$$

where

$$\mathrm{sgn}[x] = \begin{cases} 1, & \xi > 0 \\ 0, & \xi = 0 \\ -1, & \xi < 0 \end{cases}.$$

Comments: Used to simplify hardware implementation. If we assume μ is a power of two then there are no required multipliers in the update, i.e. we have only bit shift and addition operations.

**Sign-Data/Sign-Sign Algorithms**

Same as above except

$$\hat{\mathbf{w}}(n+1) = \hat{\boldsymbol{\omega}}(\nu) + \mu\,\mathrm{sgn}[\boldsymbol{\upsilon}(\nu)]\varepsilon(\nu) \quad \text{(sign-data)}$$

$$\hat{\mathbf{w}}(n+1) = \hat{\boldsymbol{\omega}}(\nu) + \mu\,\mathrm{sgn}[\boldsymbol{\upsilon}(\nu)]\,\mathrm{sgn}[\varepsilon(\nu)] \quad \text{(sign-sign)}$$

Comments: These algorithms can diverge.