5. Fast NLMS-OCF Algorithm

The NLMS-OCF algorithm presented in Chapter 2, which relies on Gram-Schmidt orthogonalization, has a complexity $O\left(NM^2\right)$. The square-law dependence of computational requirements on the number of orthogonal correction factors *M* makes it too complex to be used in practical applications, especially when *M* is larger than about two or three. In this chapter, we derive a fast version of NLMS-OCF, which uses a lattice based forward-backward predictor.

5.1 Fast Algorithm Derivation

The complexity of the NLMS-OCF algorithm can be reduced by calculating $\mathbf{x}_n^k$, which is the component of $\mathbf{x}_{n-kD}$ that is orthogonal to $\mathbf{x}_n, \mathbf{x}_{n-D}, \mathbf{x}_{n-2D}, \ldots, \mathbf{x}_{n-(k-1)D}$, efficiently. As illustrated in Figure 5.1, the sets of $M+1$ vectors used for adaptation at $n-D$ and at n have M vectors in common. This property is exploited to achieve fast orthogonalization [22].

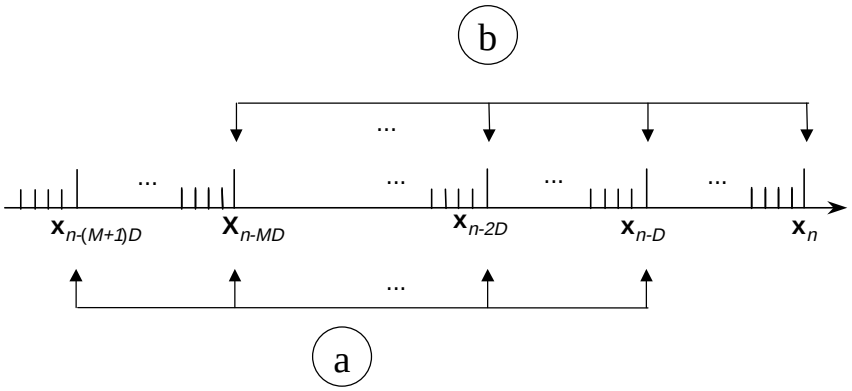

Figure 5.1 Set of Input Vectors Used for Adaptation (a) At $n-D$ and (b) At n.

Using the notation $\mathbf{e} = \mathbf{x}\langle\mathbf{Y}\rangle$ to denote the component of $\mathbf{x}$ that is orthogonal to the span of $\mathbf{Y}$,

$$\mathbf{x}_n^k = \mathbf{x}_{n-kD}\left\langle \mathbf{X}_n^{k-1} \right\rangle, \tag{5.1}$$

where $\mathbf{X}_n^{k-1} = \left[\mathbf{x}_n \quad \mathbf{x}_{n-D} \quad \cdots \quad \mathbf{x}_{n-(k-1)D}\right]$. Invoking $\mathbf{X}_n^{k-1} = \left[\mathbf{x}_n \quad \vdots \quad \mathbf{X}_{n-D}^{k-2}\right]$, (5.1) can be rewritten as

$$\mathbf{x}_n^k = \mathbf{x}_{n-kD}\left\langle \mathbf{x}_n, \mathbf{X}_{n-D}^{k-2} \right\rangle. \tag{5.2}$$

Let us define

$$\mathbf{f}_n^k = \mathbf{x}_n\left\langle \mathbf{X}_{n-D}^{k-1} \right\rangle. \tag{5.3}$$

Using the above definition, we can rewrite (5.2) as

$$\mathbf{x}_n^k = \mathbf{x}_{n-kD}\left\langle \mathbf{f}_n^{k-1}, \mathbf{X}_{n-D}^{k-2} \right\rangle \tag{5.4}$$

since $span\{\mathbf{x}_n, \mathbf{X}_{n-D}^{k-2}\} = span\{\mathbf{f}_n^{k-1}, \mathbf{X}_{n-D}^{k-2}\}$. Recognizing that $\mathbf{x}_{n-kD}\left\langle \mathbf{X}_{n-D}^{k-2} \right\rangle = \mathbf{x}_{n-D}^{k-1}$ and that both $\mathbf{f}_n^{k-1}$ and $\mathbf{x}_{n-D}^{k-1}$ are orthogonal to $\mathbf{X}_{n-D}^{k-2}$, the right hand side of (5.4) can be interpreted as a further orthogonalization of $\mathbf{x}_{n-D}^{k-1}$ with respect to $\mathbf{f}_n^{k-1}$. That is,

$$\mathbf{x}_n^k = \mathbf{x}_{n-D}^{k-1}\left\langle \mathbf{f}_n^{k-1} \right\rangle. \tag{5.5}$$

Hence, the component of $\mathbf{x}_{n-kD}$ orthogonal to $span\{\mathbf{X}_n^{k-1}\}$ can be written as

$$\mathbf{x}_n^k = \mathbf{x}_{n-D}^{k-1} + \beta_n^k \mathbf{f}_n^{k-1}, \tag{5.6}$$

where

$$\beta_n^k = -\frac{\left(\mathbf{f}_n^{k-1}\right)^H \mathbf{x}_{n-D}^{k-1}}{\left\|\mathbf{f}_n^{k-1}\right\|^2}. \tag{5.7}$$

Thus, assuming $\mathbf{f}_n^{k-1}$ is available, (5.6) provides a way to compute $\mathbf{x}_n^k$ recursively, because $\mathbf{x}_{n-D}^{k-1}$ was evaluated during time step $(n-D)$. Now, we derive a procedure to compute $\mathbf{f}_n^{k-1}$. It turns out that $\mathbf{f}_n^k$ can also be computed recursively. To compute $\mathbf{f}_n^k$ recursively, using $\mathbf{X}_{n-D}^{k-1} = \left[\mathbf{X}_{n-D}^{k-2} \quad \vdots \quad \mathbf{x}_{n-kD}\right]$, rewrite (5.3) as

$$\mathbf{f}_n^k = \mathbf{x}_n\left\langle \mathbf{x}_{n-kD}, \mathbf{X}_{n-D}^{k-2} \right\rangle. \tag{5.8}$$

Using the fact that $\mathbf{x}_{n-D}^{k-1} = \mathbf{x}_{n-kD}\left\langle \mathbf{X}_{n-D}^{k-2} \right\rangle$ and hence $span\{\mathbf{x}_{n-kD}, \mathbf{X}_{n-D}^{k-2}\} = span\{\mathbf{x}_{n-D}^{k-1}, \mathbf{X}_{n-D}^{k-2}\}$, we rewrite (5.8) as

$$\mathbf{f}_n^k = \mathbf{x}_n \left\langle \mathbf{x}_{n-D}^{k-1}, \mathbf{X}_{n-D}^{k-2} \right\rangle . \tag{5.9}$$

Noting that $\mathbf{x}_n \left\langle \mathbf{X}_{n-D}^{k-2} \right\rangle = \mathbf{f}_n^{k-1}$, as was done earlier with $\mathbf{x}_n^k$, $\mathbf{f}_n^k$ can be interpreted as a further orthogonalization of $\mathbf{f}_n^{k-1}$ with respect to $\mathbf{x}_{n-D}^{k-1}$. That is

$$\mathbf{f}_n^k = \mathbf{f}_n^{k-1} \left\langle \mathbf{x}_{n-D}^{k-1} \right\rangle . \tag{5.10}$$

Thus, the component of $\mathbf{x}_n$ orthogonal to $span\{\mathbf{X}_{n-D}^{k-1}\}$ can be written as

$$\mathbf{f}_n^k = \mathbf{f}_n^{k-1} + \phi_n^k \mathbf{x}_{n-D}^{k-1} , \tag{5.11}$$

where

$$\phi_n^k = -\frac{\left(\mathbf{x}_{n-D}^{k-1}\right)^H \mathbf{f}_n^{k-1}}{\left\| \mathbf{x}_{n-D}^{k-1} \right\|^2} . \tag{5.12}$$

Let us define

$$C_n^k = \mathbf{f}_n^{kH} \mathbf{x}_{n-D}^k , \tag{5.13}$$

$$F_n^k = \left\| \mathbf{f}_n^k \right\|^2 , \text{ and} \tag{5.14}$$

$$B_n^k = \left\| \mathbf{x}_n^k \right\|^2 . \tag{5.15}$$

Now, taking the self inner product of (5.6) we get

$$\left\| \mathbf{x}_n^k \right\|^2 = \left(\mathbf{x}_{n-D}^{k-1}\right)^H \mathbf{x}_n^k + \beta_n^{k*} \left(\mathbf{f}_n^{k-1}\right)^H \mathbf{x}_n^k \tag{5.16}$$

The second term on the right hand side of the above expression vanishes, since by (5.5) $\mathbf{x}_n^k$ is orthogonal to $\mathbf{f}_n^{k-1}$. Substituting the right hand side of (5.6) for $\mathbf{x}_n^k$ in the right hand side of (5.16), we get

$$\left\| \mathbf{x}_n^k \right\|^2 = \left\| \mathbf{x}_{n-D}^{k-1} \right\|^2 + \beta_n^k \left(\mathbf{x}_{n-D}^{k-1}\right)^H \mathbf{f}_n^{k-1} \tag{5.17}$$

Invoking the notations defined in (5.13) and (5.15), (5.17) can be rewritten as

$$B_n^k = B_{n-D}^{k-1} + \beta_n^k \left(C_n^{k-1}\right)^* \tag{5.18}$$

Similarly, the self inner product of (5.11), using the orthogonality expressed in (5.10), and the definitions in (5.13) and (5.14), results in

$$\begin{aligned} F_n^k &= \left(\mathbf{f}_n^{k-1}\right)^H \mathbf{f}_n^k + \phi_n^{k*} \left(\mathbf{x}_{n-D}^{k-1}\right)^H \mathbf{f}_n^k \\ &= \left(\mathbf{f}_n^{k-1}\right)^H \left(\mathbf{f}_n^{k-1} + \phi_n^k \mathbf{x}_{n-D}^{k-1}\right) \\ &= F_n^{k-1} + \phi_n^k C_n^{k-1} \end{aligned} \tag{5.19}$$

Thus, (5.6) and (5.11), along with (5.7) and (5.12), constitute the fast orthogonalization algorithm.

Table 5.1 summarizes the fast NLMS-OCF algorithm along with the number of computations needed for each step. Adding the number of computations for each step, we get the total number of computations per iteration to be

$$2N + 4 + \sum_{k=1}^{M} \left(5N + 6\right) = 5NM + 2N + 6M + 4 \tag{5.20}$$

Summarizing (5.20), the fast algorithm has a complexity of $O\left(NM\right)$.

Table 5.1 Summary of the Fast NLMS-OCF Algorithm.

Choose an arbitrary $\hat{\mathbf{w}}_0$. Choose the number of Orthogonal Correction Factors $M < N$. Pick $\bar{\mu} \in (0,2)$. Repeat the following steps for each n.	Number of Computations
(1) $e_n = d_n - \hat{\mathbf{w}}_n^H \mathbf{x}_n$	N
(2) $\mathbf{x}_n^0 = \mathbf{f}_n^0 = \mathbf{x}_n$	
(3) $B_n^0 = F_n^0 = B_{n-1}^0 + \lvert x_n \rvert^2 - \lvert x_{n-N} \rvert^2$	2
(4) $\mu_0 = \dfrac{\bar{\mu} e_n^*}{B_n^0}$	2
(5) $\hat{\mathbf{w}}_{n+1}^1 = \hat{\mathbf{w}}_n + \mu_0 \mathbf{x}_n$	N
For $k = 1,2,\ldots M$, repeat steps (6)-(15)	
(6) $C_n^{k-1} = \left(\mathbf{f}_n^{k-1}\right)^H \mathbf{x}_{n-D}^{k-1}$	N
(7) $\beta_n^k = -\dfrac{C_n^{k-1}}{F_n^{k-1}}$	1
(8) $\phi_n^k = -\dfrac{\left(C_n^{k-1}\right)^*}{B_{n-D}^{k-1}}$	1
(9) $\mathbf{x}_n^k = \mathbf{x}_{n-D}^{k-1} + \beta_n^k \mathbf{f}_n^{k-1}$	N
(10) $\mathbf{f}_n^k = \mathbf{f}_n^{k-1} + \phi_n^k \mathbf{x}_{n-D}^{k-1}$	N
(11) $F_n^k = F_n^{k-1} + \phi_n^k C_n^{k-1}$	1
(12) $B_n^k = B_{n-D}^{k-1} + \beta_n^k \left(C_n^{k-1}\right)^*$	1
(13) $e_n^k = d_{n-kD} - \hat{\mathbf{w}}_{n+1}^{kH} \mathbf{x}_{n-kD}$	N
(14) $\mu_k = \begin{cases} \dfrac{\bar{\mu} e_n^{k*}}{B_n^k} & \text{if } B_n^k \neq 0 \\ 0 & \text{otherwise} \end{cases}$	2
(15) $\hat{\mathbf{w}}_{n+1}^{k+1} = \hat{\mathbf{w}}_{n+1}^k + \mu_k \mathbf{x}_n^k$	N
(16) $\hat{\mathbf{w}}_{n+1} = \hat{\mathbf{w}}_{n+1}^{M+1}$	

Note 1: The recursive computation of B_n^0*, shown in Step 3, involves a unity gain loop. This can lead to numerical problems due to the accumulation of round-off error. However, this problem arises only in floating-point implementations and, interestingly, not in fixed-point implementations* [21]. *In floating-point implementations, a small damping factor is included in the recursive computation to prevent round-off instability. Hence, Step 3 is modified to*

$$B_n^0 = F_n^0 = \gamma B_{n-1}^0 + |x_n|^2 - |x_{n-N}|^2 \tag{5.20}$$

where γ *is the damping factor close to unity.*

Note 2: The divisions needed in Steps 4, 7, 8, and 14 may be performed using series expansion or table look-up techniques to a reasonable degree of accuracy. A small regularization constant can be added to the quantities in the denominator to avoid division by very small numbers. This results in the following equations for Steps 4, 7, 8, and 14, respectively.

$$\mu_0 = \frac{\bar{\mu} e_n^*}{B_n^0 + \alpha} \tag{5.21}$$

$$\beta_n^k = -\frac{C_n^{k-1}}{F_n^{k-1} + \alpha} \tag{5.22}$$

$$\phi_n^k = -\frac{\left(C_n^{k-1}\right)^*}{B_{n-D}^{k-1} + \alpha} \tag{5.23}$$

$$\mu_k = \begin{cases} \dfrac{\bar{\mu} e_n^{k*}}{B_n^k + \alpha} & \text{if } B_n^k \neq 0 \\ 0 & \text{otherwise} \end{cases} \tag{5.24}$$

In the above equations, α is the regularization constant.

5.2 Lattice Analogy

We can interpret $\mathbf{x}_n^k$ as the error in the backward prediction of $\mathbf{x}_{n-kD}$ using $\mathbf{x}_{n-(k-1)D}, \mathbf{x}_{n-(k-2)D}, \ldots, \mathbf{x}_n$ and $\mathbf{f}_n^k$ as the error in the forward prediction of $\mathbf{x}_n$ using $\mathbf{x}_{n-kD}, \mathbf{x}_{n-(k-1)D}, \ldots, \mathbf{x}_{n-D}$. Using this interpretation and the terminology from lattice structures, ϕ_n^k

and β_n^k are the forward and backward reflection coefficients, respectively. C_n^k is the cross-correlation between the forward prediction error $\mathbf{f}_n^k$ and the delayed backward prediction error $\mathbf{x}_{n-D}^k$. F_n^k and B_n^k are the forward and backward residual energies, respectively. Thus, the above fast orthogonalization process is equivalent to forward and backward prediction of a vector process using the lattice structure.

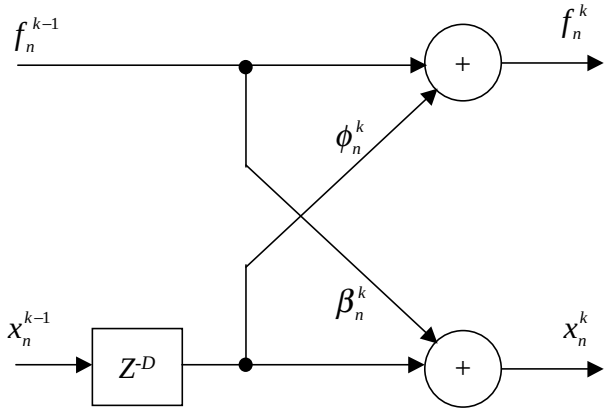

Figure 5.2 Lattice Equivalent of Fast Orthogonalization.

5.3 Simulation Results

One of the main advantages of NLMS-OCF (and its fast version) is the flexibility offered in choosing the input vectors used for adaptation by the selection of delay *D*. In this section, we present simulation results that illustrate the improvement in performance achieved by using values of *D* larger than unity. Simulation results corresponding to two different types of signals, viz. reasonably- and highly-colored signals, are shown. The reasonably- and highly-colored signals are generated as a complex Gaussian first-order autoregressive process with a pole at 0.3 and 0.97, respectively. In all simulations, the system to be identified has a 32-point long impulse response that satisfies the maximum entropy assumption. The delay line of the adaptive filter is initialized with true data values (soft initialization) in all simulations and $\mathbf{w}_0 = \mathbf{0}$ is used as the initial estimate for the weights. The measurement noise is assumed to be absent, $\varepsilon_n \equiv 0$.

Figures 5.3, 5.4, and 5.5 show the experimental learning curves, based on ensemble averaging 100 independent realizations, obtained by using the highly colored signal as input for step sizes $\bar{\mu}$ of 1.0, 0.5, and 0.01 respectively. We observe that there is an improvement in

convergence rate as the delay *D* is increased when the step size is 1.0 and 0.5. However, when the step size is 0.01, the convergence rate decreases as the delay *D* is increased. We observe from Figure 5.3 that the time-to-reach steady state is cut in half by increasing *D* from unity to 8. However, the improvement is not as significant when the delay is increased further, to 32. This suggests that the improvement in convergence rate diminishes as *D* increases. A similar effect is observed in Figure 5.4 also.

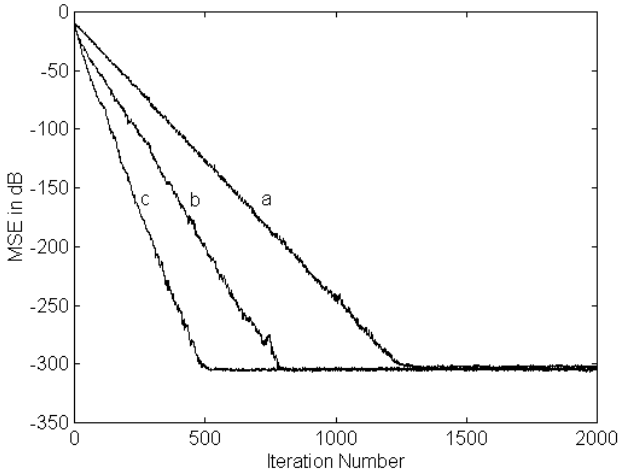

Figure 5.3 Dependence of Convergence Rate on *D* (a) $D = 1$**, (b)** $D = 8$**, and (c)** $D = 32$**. (Input: AR(1), pole at 0.97, FIR(31),** $\bar{\mu} = 1.0$**, and *M* = 9)**

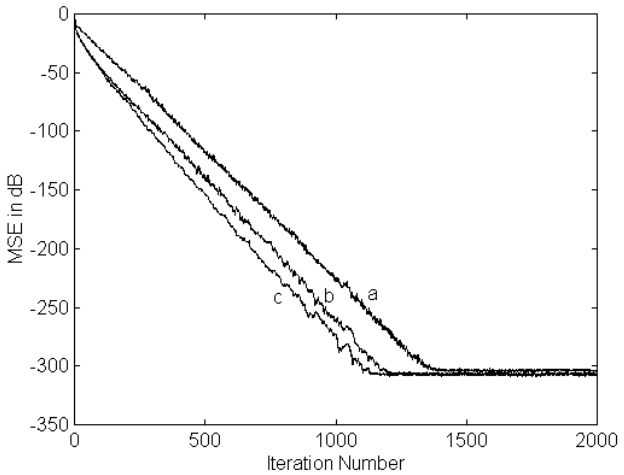

Figure 5.4 Dependence of Convergence Rate on *D* (a) $D = 1$**, (b)** $D = 64$**, and (c)** $D = 128$**. (Input: AR(1), pole at 0.97, FIR(31),** $\bar{\mu} = 0.5$**, and *M* = 9)**

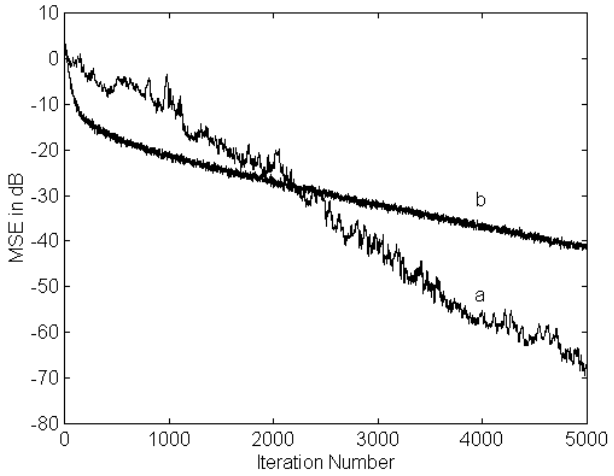

Figure 5.5 Dependence of Convergence Rate on *D* (a) $D = 1$ **and (b)** $D = 32$**.**
(Input: AR(1), pole at 0.97, FIR(31), $\overline{\mu} = 0.01$**, and *M* = 9)**

Results for the reasonably colored input are shown in Figures 5.6, 5.7, and 5.8, respectively. Here also we observe that the convergence rate improves as *D* is increased, except for "small" values of step size. We see from Figures 5.6 and 5.7 that the improvement in time-to-steady state diminishes as *D* increases. A comparison of learning curves corresponding to a reasonably colored input signal (shown in Figures 5.6 and 5.7) with learning curves corresponding to a highly colored input signal (shown in Figures 5.3 and 5.4) reveals that the improvement in convergence rate, obtained by increasing the delay, is larger with a reasonably colored signal as input than with a highly colored signal as input.

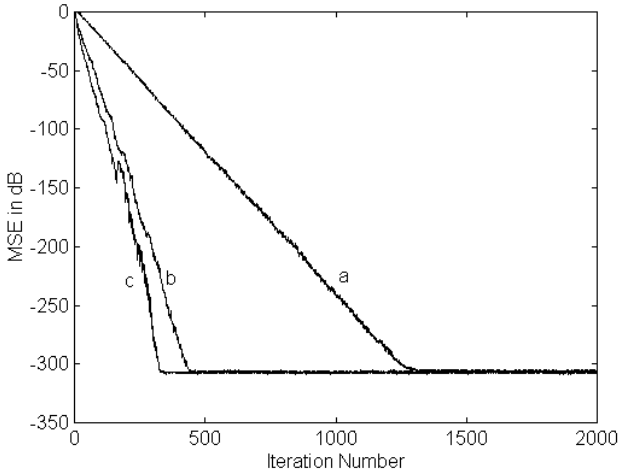

Figure 5.6 Dependence of Convergence Rate on *D* (a) $D = 1$**, (b)** $D = 4$**, and (c)** $D = 8$**.**
(Input: AR(1), pole at 0.3, FIR(31), $\overline{\mu} = 1.0$**, and *M* = 9)**

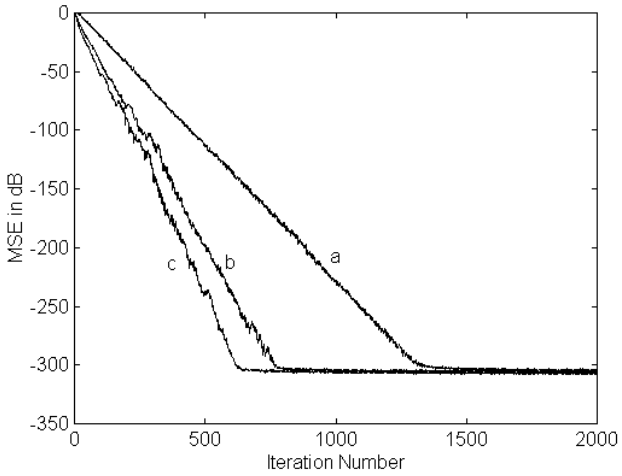

Figure 5.7 Dependence of Convergence Rate on *D* (a) $D=1$, (b) $D=4$, and (c) $D=8$. (Input: AR(1), pole at 0.3, FIR(31), $\bar{\mu}=0.5$, and *M* = 9)

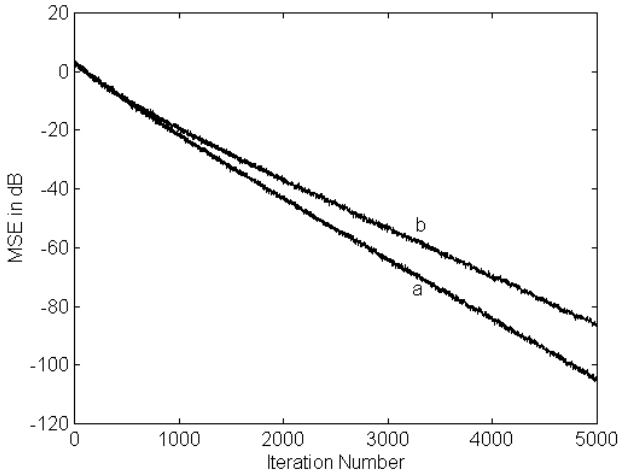

Figure 5.8 Dependence of Convergence Rate on *D* (a) $D=1$ and (b) $D=32$. (Input: AR(1), pole at 0.3, FIR(31), $\bar{\mu}=0.01$, and *M* = 9)

The following summarizes the observations based on the above results:

1. An improvement in convergence rate can be obtained by using delays larger than unity as long as the step size is not too small.
2. While there is further improvement in time-to-steady state as *D* increases, the improvement itself diminishes.

3. The improvement in time-to-steady-state obtained by increasing *D* is larger for large values of $\bar{\mu}$ than for small values of $\bar{\mu}$. This seems reasonable, since for large values of $\bar{\mu}$ we make large corrections along the input vector, thereby utilizing most of the "information" available in the input vector. For small values of *D*, due to their correlation with the current input vector, the past input vectors used for adaptation do not provide "much" additional information. Hence, there is less improvement in convergence rate. However, for large values of *D*, the correlation between the vectors used for adaptation is less and we get "significant" additional information, which results in a "significant" improvement in convergence rate.
4. For small values of $\bar{\mu}$ we make small corrections along the input vector, thereby not utilizing all the information available in the input vector. That is, we can improve our estimates even by reusing the same vector or by using a vector correlated to the input vector. Hence, we do not gain appreciably by adding new information to the adaptation vectors by choosing large *D*, since there is some useful information to be extracted from the original input vector itself.

5.4 Conclusion

We derived a fast version of the NLMS-OCF algorithm. The fast algorithm has a complexity of $O(NM)$, compared to the complexity of $O(NM^2)$ of the Gram-Schmidt based NLMS-OCF. Table 5.2 compares the number of multiply adds, multiplies, and divides per iteration needed for NLMS-OCF and Fast NLMS-OCF. It is evident from Table 5.2 that Fast NLMS-OCF needs much less computation than NLMS-OCF.

Table 5.2 Comparison of Computational Complexities of NLMS-OCF and Fast NLMS-OCF.

	NLMS – OCF	Fast NLMS-OCF
Multiply Adds	$NM^2 + 4NM + 2N + 2$	$5NM + 2N + 2M + 2$
Multiplies	$M + 1$	$M + 1$
Divides	$(M^2 + 3M + 2)/2$	$3M + 1$

The flexibility in the choice of the input vectors used for adaptation allowed by NLMS-OCF can be used to realize a faster convergence rate. This is particularly evident when the input signal

is strongly colored. The price paid for the improvement comes in the form of increased memory requirements and a moderate, selectable increase in the computational complexity. These are not serious drawbacks since present day DSPs have a large amount of on-chip memory and also have a high throughput.