ADAPTIVE FILTERS IN MATLAB: FROM NOVICE TO EXPERT

Scott C. Douglas[1]   *Ricardo Losada*[2]

[1]Department of Electrical Engineering, Southern Methodist University, Dallas, Texas 75275 USA

[2]DSP Development Group, The MathWorks, Inc., Natick, Massachusetts 01760 USA

ABSTRACT

Adaptive filters are ubiquitous tools for numerous real-world scientific and industrial applications. Many educators and practitioners employ the MATLAB® technical computing environment to implement and study adaptive filters. This paper describes the design and implementation issues regarding a recently-developed set of comprehensive Matlab adaptive FIR filtering tools. In addition to a complete suite of algorithms, the tool set includes analysis functions that enable users to quickly characterize the average performance of selected algorithms when limited data are available. We provide execution speed comparisons for algorithm families to guide users in algorithm selection when MATLAB execution time is most critical.

1. INTRODUCTION

Since the development of the least-mean-square (LMS) adaptive filter over forty years ago, adaptive filters have become well-known signal processing tools for numerous applications in communications, signal processing, and control. The widespread use of adaptive filters has created a strong need for educational materials that allow practicing engineers to learn about and explore their use. Many texts have been written on adaptive filters, including [10, 17, 29]. Many are used in senior and graduate-level courses on adaptive filters in colleges and universities worldwide.

Computer simulation is an important learning tool for understanding how adaptive filters work. Simulations give students an opportunity to understand how signal statistics affect the convergence properties of these methods and to study implementation issues. Many educators and practitioners find the technical computing software package MATLAB to be an ideal environment in which to perform these studies. The "language" of linear algebra that MATLAB uses makes it a natural adaptive filtering simulation environment. The pseudo-random sequence generators included in MATLAB are extremely useful for making synthetic signals for adaptive processing. Some adaptive filtering textbook authors (*c.f.* [29]) have based their computer simulations on MATLAB. The Filter Design Toolbox shipped with Version 6.5 of MATLAB contains only a few adaptive filtering functions, including the LMS, normalized LMS, and conventional RLS algorithms.

This paper describes the design and implementation issues regarding a recently-developed definitive set of comprehensive MATLAB adaptive FIR filtering tools.[1] The

---

[1]To be included in the next release of the Filter Design Toolbox.

31 algorithms in this tool set include gradient and signed LMS variants; recursive least-squares methods in conventional, square root, lattice, and fast transversal filter forms; frequency domain, transform domain, and subband approaches; and fast projection algorithms. The algorithms run efficiently in MATLAB. In some cases, the new algorithm versions are over one hundred times faster than their old versions. Analysis functions enable users to quickly characterize the average performance of selected algorithms when limited data are available. We provide a simple example to show how this analysis functionality can aid in the understanding of an adaptive filter's behavior.

2. ALGORITHM DESIGN ISSUES

The design of any large algorithm toolset in software requires careful planning. Specifications as to which algorithms to implement, their desired features, and the way they are likely to be used must be carefully delineated. Once the algorithmic functionality has been specified, similarities between algorithms should then be exploited to reduce coding time and make future maintenance easier. Enhancements to the tools, such as demonstrations, should also be considered.

The design of the adaptive filter functionality began with a specification for which algorithms to include in the toolset. A taxonomy of adaptive filtering algorithms was developed as shown in Fig. 1. This connected graph depicts all of the algorithms chosen for implementation. The nodes on the right contain stochastic gradient and related methods, whereas the nodes on the left are deterministic (projection and least-squares) methods. The links between the methods indicate similarities between algorithm pairs in terms of implementation structure or cost function. This taxonomy has a number of interesting features:

- The most-popular and widely-used algorithms–LMS, RLS, and NLMS–are located in the center and have numerous links to related methods. Most special-purpose algorithms such as FTF, FAP, adjoint LMS, and partitioned block FDAF appear near the geographical limits of the graph.

- There are two distinct classes of adaptive algorithms (stochastic and deterministic methods) with relatively few links between them. Within these algorithm classes are clusters of algorithms that are similar either in implementation or in the principles underlying their derivation.

We can use the natural clustering provided by this taxonomy to group the various algorithms implemented in this toolset. Table 1 lists all of these adaptive algorithms along

FIG. 1: TAXONOMY OF ADAPTIVE FILTERS

DETERMINISTIC METHODS   STOCHASTIC METHODS
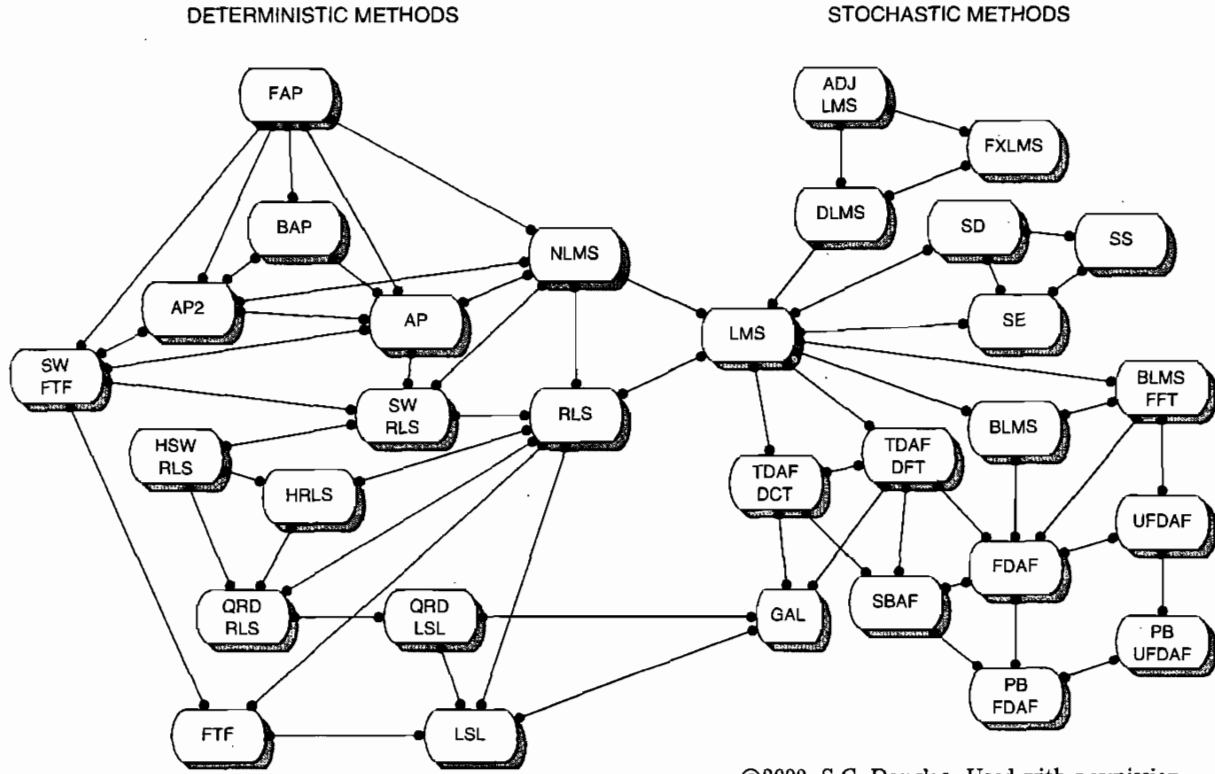
Table 1: Adaptive algorithms within the adaptive filters toolset.

Acronym	Algorithm Name	Ref.	Acronym	Algorithm Name	Ref.
Simple Updates			*Delayed Updates*		
LMS	least-mean-square	[1]	DLMS	delayed LMS	[13]
SE	sign-error	[6]	FXLMS	filtered-X LMS	[10]
SD	sign-data	[4]	ADJLMS	adjoint LMS	[26]
SS	sign-sign	[2]	*Projection Methods*		
Block and Frequency Domain Methods			NLMS	normalized LMS	[3]
BLMS	block LMS	[18]	AP	affine projection $\mathcal{O}(N^3)$	[8]
BLMSFFT	block LMS (FFT)	[18]	AP2	affine projection $\mathcal{O}(N^2)$	[15]
FDAF	frequency-domain adaptive filter	[18]	FAP	fast affine projection	[23, 24]
UFDAF	unconstrained FDAF	[18]	BAP	block affine projection	[21]
PBFDAF	partitioned-block FDAF	[14]	*Conventional Least-Squares Methods*		
PBUFDAF	partitioned-block unconstrained FDAF	[14]	RLS	recursive least-squares	[25]
Transform Domain and Subband Methods			SWRLS	sliding-window RLS	[25]
TDAFDFT	transform-domain adaptive filter (DFT)	[18]	*Square-Root Least-Squares Methods*		
TDAFDCT	transform-domain adaptive filter (DCT)	[18]	QRDRLS	QR-decomposition RLS	[25]
SBAF	subband adaptive filter	[29]	HRLS	Householder RLS	[27, 28]
Lattice Structures			HSWRLS	Householder sliding-window RLS	[28]
GAL	gradient adaptive lattice	[5]	*Fast Transversal Filters*		
LSL	least-squares lattice	[17]	FTF	fast transversal filter	[16]
QRDLSL	QR-decomposition least-squares lattice	[17]	SWFTF	sliding-window fast transversal filter	[19]

with their corresponding references. This taxonomy was also used for the technical documentation within the algorithm files; the "see also" references within the help resources employ this graph's structure.

With the algorithm list defined, the core functionality of each algorithm was then implemented as a single inline .m program in MATLAB. Great care was taken to ensure that the implementations ran as fast as possible under the current operating structure of MATLAB. All of the algorithms employ state variables that allow one to partition signals into blocks to manage memory issues within MATLAB. In addition, all algorithms save for the sign-data and sign-sign algorithms have been tested and work for complex-valued signals and coefficients.

Other important issues regarding the algorithm implementations include the following:

1. BLMSFFT uses overlap-save fast convolution for its implementation, whereas BLMS employs traditional convolution using the filter command in MATLAB. The latter implementation allows an *arbitrary* block size to provide much finer control of the computational speedup for different block sizes and filter lengths.

2. The three affine projection algorithms AP, AP2, and FAP differ in the way the input signal autocorrelation information needed for the N-dimensional projection is updated. AP uses direct inversion of the $(N \times N)$ input signal autocorrelation matrix. AP2 uses a pair of Kalman-gain-type updates to propagate the autocorrelation matrix inverse. Finally, FAP uses least-squares linear prediction via an embedded sliding-window FTF algorithm.

3. The transform-domain adaptive filtering algorithms TDAFDFT and TDAFDCT differ in the choice of sliding transform used. Both algorithms use efficient recursive estimators for the DFT or DCT input signal frequency bin values.

4. The SBAF algorithm is a special-purpose function that can be called with one or more of the other adaptive filtering functions along with prototype analysis and synthesis filters for the filter bank structures. This algorithm then runs individual adaptive filters on the individual input and desired response signals generated from the subband structures.

The rich choice of algorithms means that one has many possibilities if implementation structure is not an issue. For example, six different functions–RLS, HRLS, QRDRLS, FTF, LSL, and QRDLSL–implement an exponentially-windowed recursive least-squares adaptive filter. Three different versions of the affine projection algorithm are also available. Studies describe later provide some guidance in algorithm selection in such cases.

3. ANALYSIS FUNCTIONS

Adaptive filters has been called a "twiddler's paradise," whereby the multitude of algorithm choices is dwarfed only by the near-infinite choices of parameter values within each algorithm. An unaided novice user of adaptive filters will invariably run an adaptive algorithm to divergence with out-of-range parameter values. Alternatively, he or she will choose conservative values, resulting in an adaptive filter that "isn't doing anything." Textbooks provide answers to such difficulties, but they can potentially be avoided with the right assistive tools.

To this end, a set of analysis tools has been designed. Five of the adaptive filters–LMS, NLMS, BLMS, BLMSFFT,
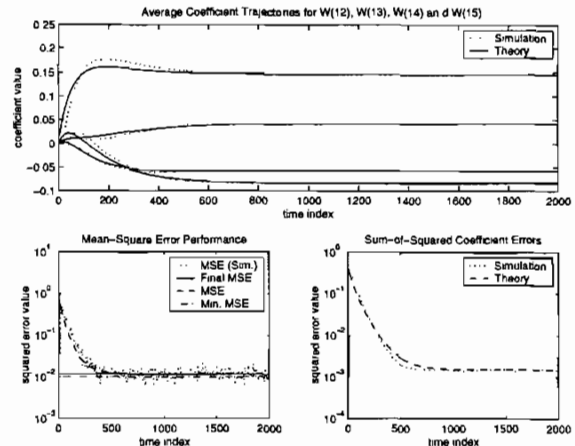

Fig. 2: Analysis of a 32-coefficient LMS adaptive filter. (a) Sample coefficient trajectories computed from analysis and estimated from simulations. (b) Mean-square error performance. (c) Sum-of-squared coefficient errors.

and SE–have optional analysis functions. These tools encapsulate existing knowledge about how these adaptive filters work [7, 9, 11, 12, 20]. Each of these procedures calculates the statistical information needed to evaluate the adaptive filter analysis, such as the input signal autocorrelation matrix, the optimum coefficient vector, and the minimum MSE, from the signals provided. These estimates are then used to calculate maximum step size values, predictions of mean-squared error (MSE) at each time instant, the mean coefficient values at each time instant, predictions of the coefficient error powers at each time instant, and final total and excess MSEs at convergence. Another general-purpose function provided in the toolset allows the user to calculate ensemble-averaged values of the aforementioned quantities for any adaptive filter supported in the toolset. The user can then plot the ensemble-averaged curves against the predicted values to check the accuracy of the analytical results. The analysis tools also have some convenient built-in features that help the novice user avoid mistakes. For example, the analysis functions return a warning if the chosen step size exceeds half of the maximum step size value predicted from the mean or mean-square analysis, depending on which one is called.

Shown in Fig. 2 are the results produced by the LMS adaptive filtering and analysis functionality, where a 32-coefficient LMS adaptive filter with correlated Gaussian input and desired response signals is being studied. All input signals were created such that $E\{x(n)x(n-l)\} = 0.5^{|l|}$. The desired response signals were created using a noisy system identification model corresponding to a low-pass FIR filter with digital cutoff frequency $\omega = 0.5\pi$ and an observation noise variance of 0.01. For the adaptive filter, $\mu = 0.008$ and zero initial coefficients were chosen. The theoretical curves are computed from one pass of the analysis function, whereas the simulation curves are computed by fifty simulation runs using the adaptive filter function. The closeness of the corresponding curves shows that the analysis is accurate for the chosen step size and data statistics. These results can be starting points for further explorations with different signals, step sizes, and filter lengths. This study does not take significant computing time; the generation of these results including all signals and plots took 25.2 seconds on a 700MHz Pentium-based laptop computer.
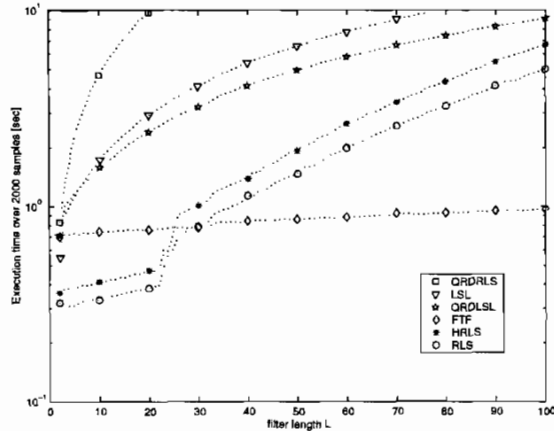
Fig. 3: Execution times for six different RLS algorithms as a function of filter length over 2000-sample signal sets.

Fig. 4: Execution times for the AP, AP2, and FAP algorithms as a function of projection order over 10000-sample signal sets, where $L = 100$.

4. INITIALIZATION ISSUES

The most important internal variables of any adaptive FIR filter are the coefficient values being updated at each time instant and the stored input signal samples used to compute the output and error signals from the coefficients. All but the simplest gradient methods, however, have additional state variables that specify the adaptive filter's current operating point. For example, all direct-form least-squares adaptive filters have state variables associated with the updating of the Kalman gain vector, such as the inverse of the exponentially-weighted input signal autocorrelation matrix. In some cases, the numbers of these internal variables is large; for example, the FAP algorithm has $3L + 10N - 8$ such internal variables for an L-coefficient filter and a projection order of N. Keeping track of these variables and their values can be problematic without a systematic procedure for handling them within the toolset. In addition, certain algorithms, such as the fast transversal filter, require special care in the initialization of their state variables or divergence will occur.

As part of the adaptive filters toolset, we have developed a set of initialization routines for each adaptive filter. These programs create a structure variable for the adaptive filter that includes placeholders for all necessary state information for the adaptive filter's operation. Wherever possible, we have also included default values that represent the most-neutral choices for their values. For example, when calling the FAP algorithm, one only needs to specify i) the initial coefficient vector, ii) the step size, iii) the projection offset, and iv) the $(N-1)$th-order input signal cross-correlation vector whose length determines the projection order. The help resources for this function suggest an all-zero vector for this cross-correlation vector as well. The initialization of the sliding-window FTF algorithm embedded within the FAP algorithm is built into the initialization routine. At any time during the adaptive filter's operation, the values of these various states are accessible as well, allowing one to track diagnostic information if needed.

The use of a state variable structure also allows one to handle memory issues with care when local memory is at a premium. Every adaptive filtering routine has been tested to run in a block fashion, where the input and desired response signals are parsed into blocks and the adaptive filter successively processes each block.
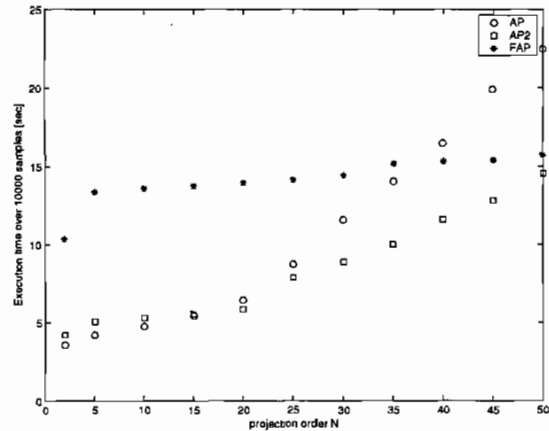
5. PERFORMANCE RELATIONSHIPS

Many signal processing experts use MATLAB to prototype systems and methods. The execution time associated with a typical numerical procedure is a primary concern in such explorations; one doesn't want to waste time waiting for numerical results to become available. Our adaptive filtering tools have been designed to provide the fastest execution time possible without going to extreme steps such as loop unrolling. We also have provided several implementations of a particular approach wherever possible to allow the user a choice of algorithms.

Fig. 3 shows the execution times of six different RLS algorithms as a function of filter length over 2000-sample signal records on the aforementioned 700MHz laptop computer. All algorithms give identical behavior up to numerical errors with an equivalent initialization and a unity forgetting factor. While the $O(L)$ stabilized FTF algorithm is more efficient for longer filter lengths ($L > 26$), both the conventional RLS and (square root) Householder RLS algorithms run faster for shorter filter lengths and are to be preferred over the "faster" FTF algorithm if MATLAB execution time is of primary concern.

Fig. 4 shows the execution times of the AP, AP2, and FAP versions of the affine projection algorithm as a function of projection order N for an $L = 100$-tap filter over 10000-sample records. In this example, the "brute force" $O(N^3)$ version of the algorithm runs the fastest for $N < 15$, and the $O(N^2)$ version of the algorithm runs the fastest for all projection orders between $N = 15$ and $N = 50$. The $O(N)$ FAP algorithm version requires the most execution time in all cases. Thus, unless specific issues regarding the numerical behavior or initialization of the FAP version of the affine projection algorithm are under scrutiny, the "more-complex" versions of the algorithm are to be preferred.

Fig. 5 shows the execution times of the LMS, BLMS, and BLMSFFT adaptive filters as a function of filter length on 81920-sample signal records. Here, we have chosen block sizes equal to the filter lengths, such that BLMS and BLMSFFT perform the same. The implementations, however, allow for block sizes that differ from the filter lengths. In the case of the BLMS algorithm, the choice of filter length L and block size N are completely arbitrary, whereas the BLMSFFT implementation requires that $L + N$ be a power of two. For small step sizes and stationary signals, the con-
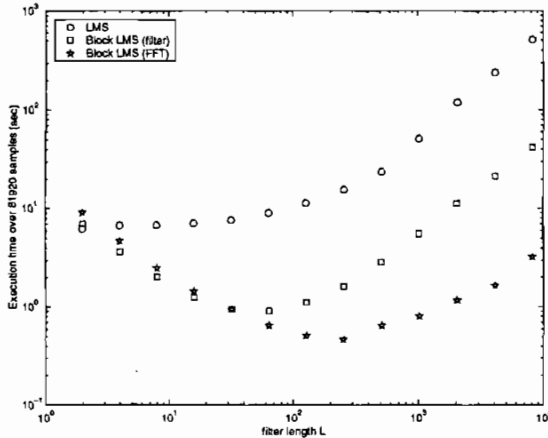
Fig. 5: Execution times for the LMS, BLMS, and BLMSFFT algorithms as a function of filter (equal to block) length over 81200-sample signal sets.
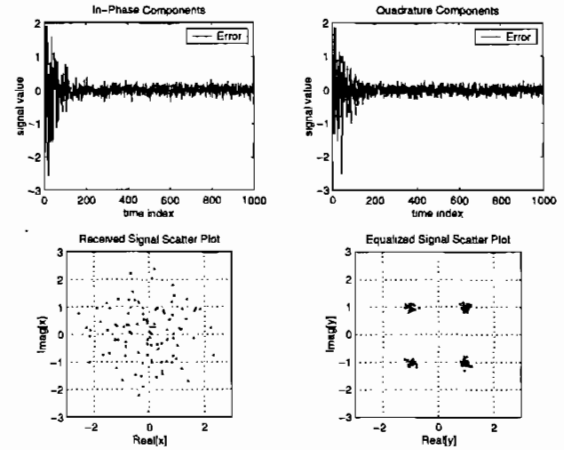

Fig. 6: Adaptive equalization example for the TDAFDFT algorithm. (a) Real-valued component of the error signal. (b) Imaginary-valued component of the error signal. (c) Scatter-plot of the last 101 samples of the input signal. (d) Scatter-plot of the last 101 samples of the output signal.

vergence behaviors of all three algorithms are similar [22]. Except for $L = N = 2$, the block algorithms run faster. Moreover, the speedup is significant as L and N are increased. The speedup of the BLMS algorithm is particularly useful given the freedom with which L and N can be chosen, allowing the user to employ BLMS in a "fast prototyping" design procedure. In this procedure, the BLMS algorithm is used to choose approximate values of L and the algorithm step size given candidate input and desired response signals and a small block length N. Then, the LMS algorithm can be employed to fine-tune the algorithm's behavior and to verify the adaptation performance.

6. DEMONSTRATIONS AND EXAMPLES

When first learning about adaptive filters, it is useful to have a set of practical examples that illustrate the main application areas of adaptive filters. These examples are essentially specified by the choice of the input and desired response signals, as any adaptive FIR filter is defined by these signals and the chosen parameters. In this adaptive filter toolset, two types of examples have been incorporated:

- Every adaptive filter function has an associated simple example contained within the help resources for the adaptive filter. This example consists of a series of commands that when run within MATLAB (1) generate the input and desired response signals, (2) specify the adaptive FIR filter parameters, (3) apply the adaptive filter to the generated signals, and (4) plot quantities related to the performance of the adaptive filter.

- Demonstrations illustrating the use of the adaptive filter toolset in well-known signal processing tasks are also provided. At the time of publication, these demonstrations included (i) acoustic echo cancellation, (ii) adaptive equalization, (iii) adaptive noise cancellation, (iv) adaptive line enhancement of speech, and (v) active noise control.

Fig. 6 shows one of the simple examples contained within the help resources for the adaptive filters toolset. This example depicts a baseband adaptive equalization task, in which the input signal is a complex-valued 4-QAM signal with pseudo-random real and quadrature components. The complex-valued channel used in the example is described by

the transfer function

$$H(z) = e^{-j\pi/4}\frac{-0.7 + z^{-1}}{1 - 0.7z^{-1}}z^{-16}, \qquad (1)$$

in which a 16-tap delay has been introduced to improve the equalizer's convergence performance. The desired signal is generated as the output of the above channel when corrupted by complex-valued uncorrelated Gaussian noise whose real and imaginary components have identical variances of 0.01. Shown in the upper two windows of this figure are the errors generated from the TDAFDFT algorithm on one realization of these signals, where a 32-tap filter has been used. As can be seen, both the real and imaginary components of the error signal converge to small values. The lower two windows show scatter-plots of the input and output signals of the adaptive filter, respectively, for the last 101 samples of both signals in the simulation. The ordered nature of the output signal scatter-plot illustrates that equalization has been obtained.

Fig. 7 shows the behavior of the sign-error algorithm as applied to an adaptive line enhancement task. The desired signal for this example is generated as

$$d(n) = \sin(0.1\pi n) + \eta(n), \qquad (2)$$

where $\eta(n)$ is a real-valued pseudo-random Gaussian noise sequence with unit variance. A 32-tap adaptive filter in a one-step linear prediction structure is employed to extract an estimate of the sinusoidal signal in the adaptive filter's output. Shown in the upper portion of the figure are the original desired response signal, the adaptive filter's output signal, and the clean sinusoidal signal over the last 101 samples of the 5000-sample simulation. The similarity between the adaptive filter's output and the clean sinusoid indicate that the adaptive line enhancer is functioning properly. The lower portion of the figure shows the power spectra of the desired and output signals, respectively, over the last 1000 samples of the simulation. Like its namesake, the adaptive line enhancer enhances the sinusoidal "line" within the signal spectrum, effectively reducing the noise power in the system's output.
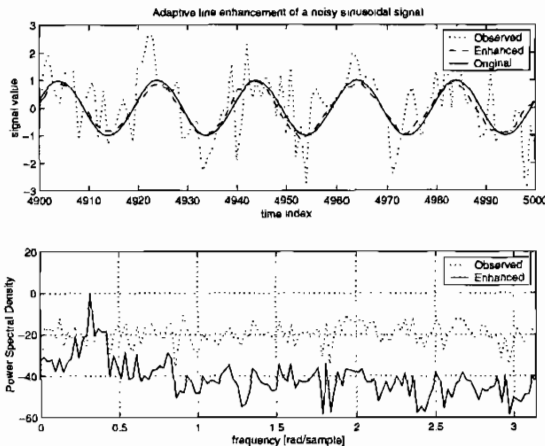
172

Fig. 7: Adaptive line enhancement example for the sign-error algorithm. (a) The original noisy sinusoidal signal, the output of the adaptive filter, and the noiseless sinusoidal signal. (b) Power spectra of the desired response and adaptive filter output signals.

7. CONCLUSIONS

Adaptive filters are important signal processing tools. This paper describes a comprehensive adaptive filter toolset developed for the MATLAB technical computing software package. The toolset includes adaptive filter functions, analysis functions, examples, and demonstrations. Great care has been taken to make the toolset easy to use for the novice yet powerful and flexible enough for the expert. It is hoped that the capabilities contained in this toolset will enrich on-going developments in the adaptive filtering field.

REFERENCES

[1] B. Widrow and M.E. Hoff, Jr., "Adaptive switching circuits," *IRE Wescon Conv. Rec.*, pt. 4, pp. 96-104, 1960.

[2] R.W. Lucky, "Techniques for adaptive equalization of digital communication systems," *Bell Syst. Tech. J.*, vol. 45, pp. 255-286, Feb. 1966.

[3] J. Nagumo and A. Noda, "A learning method for system identification," *IEEE Trans. Automatic Control*, vol. AC-12, pp. 282-287, June 1967.

[4] J.L. Moschner, "Adaptive filter with clipped input data," Ph.D. thesis, Stanford Univ., Stanford, CA, June 1970.

[5] L.J. Griffiths, "A continuously adaptive filter implemented as a lattice structure," *Proc. IEEE Int. Conf. Acoust., Speech, Signal Processing*, Hartford, CT, pp. 683-686, 1977.

[6] A. Gersho, "Adaptive filtering with binary reinforcement," *IEEE Trans. Inform. Theory*, vol. IT-30, pp. 191-199, Mar. 1984.

[7] W.A. Gardner, "Learning characteristics of stochastic-gradient-descent algorithms: A general study, analysis, and critique," *Signal Processing*, vol. 6, no. 2, pp. 113-133, Apr. 1984.

[8] K. Ozeki and T. Umeda, "An adaptive filtering algorithm using an orthogonal projection to an affine subspace and its properties," *Electron. Commun. Japan*, vol. 67-A, no. 5, pp. 19-27, May 1984.

[9] A. Feuer and E. Weinstein, "Convergence analysis of LMS filters with uncorrelated Gaussian data," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 33, no. 1, pp. 222-230, Feb. 1985.

[10] B. Widrow and S.D. Stearns, *Adaptive Signal Processing* (Englewood Cliffs, NJ: Prentice-Hall, 1985).

[11] V.J. Mathews and S.H. Cho, "Improved convergence analysis of stochastic gradient adaptive filters using the sign algorithm," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-35, pp. 450-455, Apr. 1987.

[12] N.J. Bershad, "Behavior of the ϵ-normalized LMS algorithm with Gaussian inputs," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-35, pp. 636-644, May 1987.

[13] G. Long, F. Ling, and J.G. Proakis, "The LMS algorithm with delayed coefficient adaptation," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-37, pp. 1397-1405, Sept. 1989.

[14] J.S. So and K.K. Pang, "Multidelay block frequency domain adaptive filter," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-38, pp. 373-376, Feb. 1990.

[15] Y. Maruyama, "A fast method of projection algorithm," *Proc. 1990 IEICE Spring Conf.*, B-744.

[16] D.T.M. Slock and T. Kailath, "Numerically stable fast transversal filters for recursive least squares adaptive filtering," *IEEE Trans. Signal Processing*, vol. 38, no. 1, pp. 92-114, Jan. 1991.

[17] S. Haykin, *Adaptive Filter Theory*, 2nd ed. (Englewood, Cliffs, NJ: Prentice Hall, 1991).

[18] J.J. Shynk, "Frequency-domain and multirate adaptive filtering," *IEEE Signal Processing Mag.*, vol. 9, no. 1, pp. 14-37, Jan. 1992.

[19] D.T.M. Slock and T. Kailath, "A modular prewindowing framework for covariance FTF RLS algorithms," *Signal Processing*, vol. 28, pp. 47-61, 1992.

[20] S.C. Douglas and T.H.-Y. Meng, "Normalized data nonlinearities for LMS adaptation," *IEEE Trans. Signal Processing*, vol. 42, pp. 1352-1365, June 1994.

[21] M. Montazeri and P. Duhamel, "A set of algorithms linking NLMS and block RLS algorithms," *IEEE Trans. Signal Processing*, vol. 43, pp. 444-453, Feb. 1995.

[22] S.C. Douglas, "Analysis of the multiple-error and block least-mean-square adaptive algorithms," *IEEE Trans. Circuits Syst. II: Analog Digital Signal Processing*, vol. 42, pp. 92-101, Feb. 1995.

[23] S.L. Gay and S. Tavathia, "The fast affine projection algorithm," *Proc. IEEE Int. Conf. Acoust., Speech, Signal Processing*, Detroit, MI, vol. 5, pp. 3023-3026, May 1995.

[24] M. Tanaka, Y. Kaneda, S. Makino, and J. Kojima, "Fast projection algorithm for adaptive filtering," *IEICE Trans. Fund. of Electron., Commun., Comput. Sci.* vol. E78-A, no. 10, pp. 1355-1361, Oct. 1995.

[25] M. Hayes, *Statistical Digital Signal Processing and Modeling* (New York: Wiley, 1996).

[26] E.A. Wan, "Adjoint LMS: An efficient alternative to the filtered-X LMS and multiple error LMS algorithms," *Proc. IEEE Int. Conf. Acoust., Speech, Signal Processing*, Atlanta, GA, vol. 3, pp. 1842-1845, May 1996.

[27] A.A. Rontogiannis and S. Theodoridis, "Inverse factorization adaptive least-squares algorithms," *Signal Processing*, vol. 52, no. 1, pp. 35-47, July 1996.

[28] S.C. Douglas, "Numerically-robust $\mathcal{O}(N^2)$ RLS algorithms using least-squares prewhitening," *Proc. IEEE Int. Conf. Acoust., Speech, Signal Processing*, Istanbul, Turkey, vol. I, pp. 412-415, June 2000.

[29] B. Farhang-Boroujeny, *Adaptive Filters: Theory and Applications* (New York: Wiley, 2000).