Adaptive Filtering in Realistic Noise: Analysis, Algorithms, and Performance Evaluation of LMS Variants

Upama Roy Chowdhury
Department of Electrical and Computer Engineering
University of New Mexico

I. Introduction

Adaptive filtering is a foundational technique in modern signal processing, enabling real-time estimation, interference suppression, and system identification in environments where signal and noise characteristics vary over time. Unlike fixed filters, adaptive filters adjust their parameters continuously based on incoming data, making them suitable for nonstationary, uncertain, and dynamically evolving systems. The classical framework was established through the pioneering work of Widrow and colleagues, who introduced the Least-Mean-Square (LMS) algorithm and adaptive noise cancelling architectures [2], [3]. These developments were later unified under the broader theories of Wiener filtering, stochastic gradient methods, and orthogonal lattice structures.

Extensive treatments by Haykin [1], Sayed [4], Clarkson [5], and Hayes provide the analytical foundations for convergence, stability, and performance evaluation of LMS-type algorithms. Building on this foundation, research has focused on improving robustness, convergence speed, and computational efficiency. Variable-step-size LMS methods [7], [8] accelerate adaptation during transients while reducing steady-state misadjustment, whereas nonlinear and sign-based variants [9] enhance resilience to impulsive or heavy-tailed noise. The Gradient Adaptive Lattice (GAL) algorithm [10] offers a numerically stable structure for autoregressive modeling, especially in correlated environments.

Recent work continues to extend adaptive filtering to more complex scenarios, including sparse and robust formulations, non-Gaussian noise models, and hybrid systems that integrate data-driven and learning-based approaches [11]–[13]. These advances are motivated by practical applications where signal statistics drift because of environmental changes, sensor variability, or intrinsic system dynamics.

This project evaluates a broad class of adaptive filtering algorithms—LMS, NLMS, Leaky LMS, Block LMS, Sign-error, Sign-data, Sign-sign LMS, VS-LMS, and GAL—under diverse stationary and nonstationary noise conditions. Using speech signals, AR(1) colored noise, sinusoidal interference, impulsive disturbances, and time-varying processes, we examine each algorithm's convergence, tracking ability, steady-state behavior, misadjustment, and sensitivity to eigenvalue structure. The study provides a unified experimental comparison linking classical theory to modern adaptive filtering challenges.

II. Adaptive Filtering Theory

Adaptive filtering aims to recover a desired signal from noisy observations by iteratively updating the parameters of a finite-impulse-response (FIR) filter. The update relies on an error signal that measures the mismatch between the filter output and the desired response. Because the update depends only on incoming data, adaptive filters can operate in unknown, time-varying, and nonstationary environments.

A. Wiener Solution and Optimal Filtering

Given a desired signal $d[n] = s[n] + v[n]$ and a reference input vector $\mathbf{x}[n] = [x[n], x[n-1], \ldots, x[n-L+1]]^T$, the filter output is $y[n] = \mathbf{w}^T[n]\mathbf{x}[n]$ with error $e[n] = d[n] - y[n]$. The goal is to minimize the mean-square error $J(\mathbf{w}) = \mathbb{E}\{|e[n]|^2\}$. Setting $\nabla_{\mathbf{w}} J = 0$ yields the Wiener–Hopf equations

$$\mathbf{R}\mathbf{w}_{\text{opt}} = \mathbf{p},$$

where $\mathbf{R} = \mathbb{E}\{\mathbf{x}\mathbf{x}^T\}$ and $\mathbf{p} = \mathbb{E}\{d\mathbf{x}\}$. Although $\mathbf{w}_{\text{opt}}$ provides the minimum achievable MSE, it requires ensemble statistics that are generally unknown in practice.

B. LMS and Stochastic Gradient Updates

The Least–Mean–Square (LMS) algorithm replaces ensemble expectations with instantaneous estimates, giving the classic update

$$\mathbf{w}[n+1] = \mathbf{w}[n] + \mu\, e[n]\mathbf{x}[n].$$

LMS is a stochastic gradient descent method whose stability requires $0 < \mu < 2/\lambda_{\max}$, where $\lambda_{\max}$ is the largest eigenvalue of $\mathbf{R}$. The steady-state excess MSE scales linearly with μ, defining the misadjustment. Highly correlated inputs enlarge the eigenvalue spread and slow convergence— a key limitation of LMS in colored and AR-like environments.

C. NLMS and LMS Variants

The Normalized LMS (NLMS) algorithm compensates for input power fluctuations by using

$$\mathbf{w}[n+1] = \mathbf{w}[n] + \mu_n \frac{e[n]\mathbf{x}[n]}{\varepsilon + \|\mathbf{x}[n]\|^2},$$

leading to improved convergence and robustness in nonstationary noise conditions.

Additional variants used in this project include:

- **Leaky LMS:** adds a $(1-\gamma)$ factor to control weight drift in correlated inputs.
- **Block LMS:** updates weights per data block, reducing computational cost.
- **Sign-based LMS (SE, SD, SS):** replace values with signs, providing robustness to impulsive noise.
- **Variable Step-Size LMS (VS–LMS):** adapts $\mu[n]$ based on error gradients, enabling faster tracking of nonstationary systems.

These variants address limitations of classical LMS in different noise and correlation regimes.

D. Nonstationary and Non-Gaussian Considerations

Real-world environments often feature noise whose variance, distribution, or correlation structure changes with time. Two primary nonstationary scenarios are considered:

- **Time-varying noise power:** LMS suffers from sensitivity to noise bursts, while NLMS maintains stable tracking due to normalization.
- **Time-varying AR(1) statistics:** as the underlying correlation structure drifts, VS–LMS provides superior tracking by adapting its step size dynamically.

Impulsive disturbances violate Gaussian assumptions, causing LMS to produce large error spikes. Sign-based algorithms mitigate this by bounding the update magnitude.

E. Lattice Perspective: GAL for AR(1) Modeling

The Gradient Adaptive Lattice (GAL) algorithm provides a numerically stable mechanism for autoregressive modeling. For an AR(1) process, the reflection coefficient $\Gamma_1[n]$ is adapted by minimizing a Burg-type cost. GAL converges to the theoretical coefficient $-a$, offering insight into how correlation affects LMS behavior: larger $|a|$ leads to stronger correlation, a more elongated cost surface, and slower LMS convergence.

Overall, the Wiener filter provides the theoretical optimum, LMS and its variants approximate this optimum in real time, and the nonstationary extensions (NLMS, VS–LMS, sign-based, GAL) address the practical challenges of real mixed-noise environments.

III. Methodology and Simulation Setup

This project evaluates a broad family of adaptive filters—LMS, NLMS, Leaky LMS, Block LMS, Sign-based variants, Variable Step-Size LMS (VS–LMS), and Gradient Adaptive Lattice (GAL)—under controlled stationary and nonstationary noise conditions. All simulations were implemented in MATLAB using double-precision arithmetic and a fixed random seed for reproducibility.

A. Signal Model and Noise Construction

The desired signal is the normalized `handel` audio sample,

$$s[n] \leftarrow \frac{s[n] - \mathbb{E}\{s[n]\}}{\sqrt{\mathrm{var}\{s[n]\}}},$$

ensuring unit variance and a well-defined Wiener MSE reference. The disturbance $v[n]$ is a tunable mixture of:

- **White Gaussian noise (WGN):** $w[n] \sim \mathcal{N}(0, 1)$,
- **AR(1) colored noise:** $u[n] = 0.9u[n-1] + e[n]$,
- **Sinusoidal interference:** $r[n] = \sin(2\pi f_0 n/F_s)$.

These components are scaled and summed to form

$$v[n] = \sigma_{\mathrm{wgn}} w[n] + \sigma_{\mathrm{ar}} u[n] + \sigma_{\mathrm{sine}} r[n],$$

after which the noisy primary signal is

$$d[n] = s[n] + v[n].$$

B. Reference Channel and Adaptive Filter Configuration

A correlated reference signal is generated by filtering the noise through a short FIR model:

$$x[n] = (b_{\text{ref}} * v)[n], \qquad b_{\text{ref}} = [1,\, 0.3,\, -0.2].$$

Both $x[n]$ and $d[n]$ are normalized to unit variance prior to adaptation.

Unless otherwise stated, the adaptive filters use an FIR length of $L = 32$ taps with:

- LMS: $\mu = 0.01$,
- NLMS: $\mu_{\text{n}} = 0.8$, $\varepsilon = 10^{-6}$,
- Leaky LMS: $\mu = 0.01$, $\gamma = 10^{-3}$,
- Block LMS: $\mu = 0.01$, block length $B = 32$,
- Sign-based LMS: $\mu = 0.01$,
- VS–LMS: $\mu_0 = 0.005$, $\mu_{\min} = 10^{-4}$, $\mu_{\max} = 0.05$.

All algorithms operate on the same $(x[n], d[n])$ sequence to ensure direct comparability.

C. Performance Metrics

Each algorithm produces the error signal

$$e[n] = d[n] - y[n],$$

interpreted as the estimate $\hat{s}[n]$ of clean speech. The following metrics are computed:

- **Instantaneous and smoothed MSE:** $|e[n]|^2$ and its moving-average (200-sample) smoothing.
- **Steady-state MSE:** the average MSE over the final 30% of samples.
- **Misadjustment:** $M = (\xi_\infty - \xi_{\min})/\xi_{\min}$.
- **Misalignment:** $\|\mathbf{w}_{\text{final}} - \mathbf{w}_{\text{opt}}\|_2$, where $\mathbf{w}_{\text{opt}}$ is the Wiener solution.
- **Output SNR:**
$$\text{SNR}_{\text{out}} = 10\log_{10}\frac{\|s\|_2^2}{\|\hat{s} - s\|_2^2}.$$

The Wiener benchmark $\mathbf{w}_{\text{opt}} = R^{-1}p$ is computed from empirical correlations estimated from the reference input.

D. Visualization and Analysis Tools

For each experiment, the following are generated:

- time-domain waveforms (clean, noisy, LMS, NLMS),
- learning curves of all algorithms,
- coefficient evolution plots,
- Welch PSD estimates and FFT magnitude spectra,
- grouped SNR comparison plots,
- eigenvalue and condition-number analysis of $\hat{R}_{xx}$.

These diagnostics provide insight into convergence speed, steady-state error, spectral shaping, and robustness under different noise structures.

E. Additional Simulation Scenarios

The following controlled experiments extend the baseline case:

- **Step-size sensitivity:** three LMS step sizes to illustrate the speed–misadjustment trade-off.
- **Nonstationary noise:** increasing noise variance to test LMS vs. NLMS.
- **Time-varying AR(1) coefficient:** comparison of LMS and VS–LMS tracking.
- **Single-channel ANC:** NLMS uses a delayed version of $d[n]$ as reference.
- **Impulsive noise robustness:** comparison of LMS and Sign-Error LMS.
- **AR(1) correlation sweep:** performance as a varies in $\{0.2, 0.5, 0.9\}$.
- **Sinusoid frequency sweep:** suppression performance for 200, 400, and 800 Hz tones.

Finally, a two-tap example illustrates the geometric behavior of LMS, NLMS, and steepest descent on a quadratic cost surface, and a GAL experiment demonstrates AR(1) modeling via reflection-coefficient adaptation.

IV. Results and Analysis

This section presents a comprehensive evaluation of all adaptive filtering algorithms implemented in this project. The experiments examine convergence behavior, spectral characteristics, misalignment, robustness to different noise types, and weight–space trajectories. All results are generated using the MATLAB implementation described in Section III.

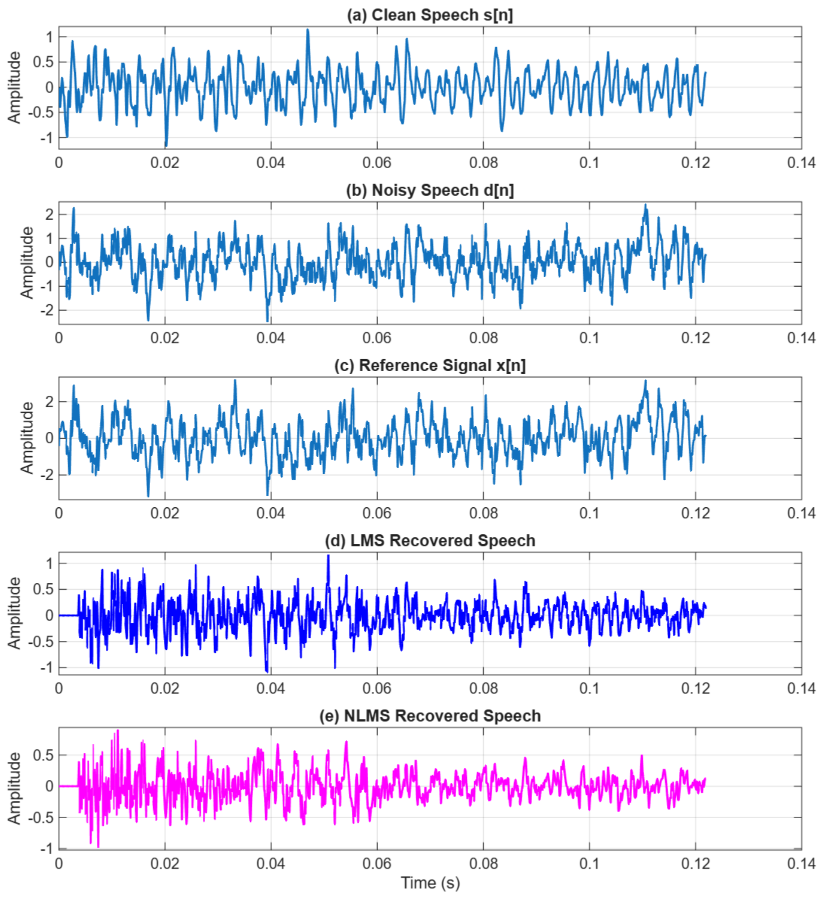
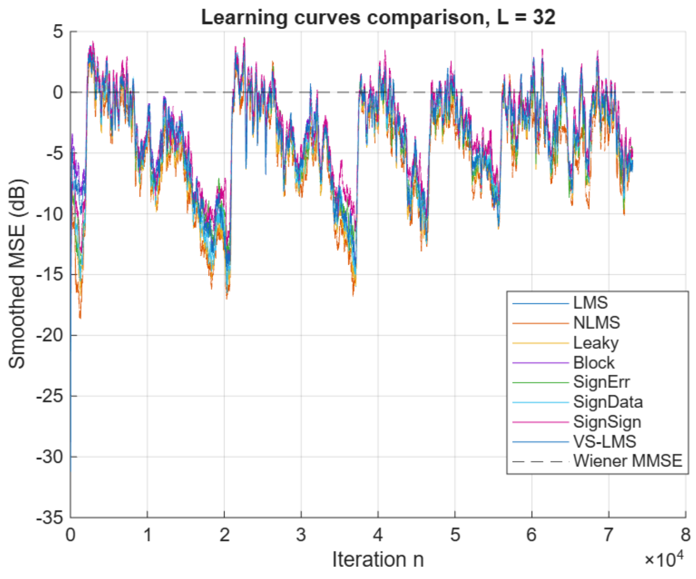


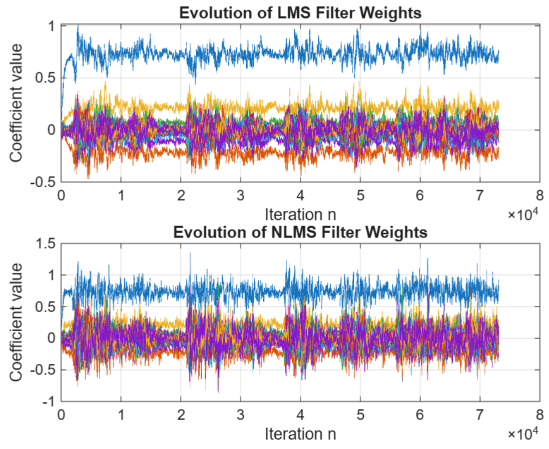
Fig. 1: Time-domain comparison: (a) clean speech, (b) noisy signal, (c) reference noise, (d) LMS output, (e) NLMS output.

A. *Time-Domain Performance*

Figure 1 shows the first 1,000 samples of the clean speech signal, the noisy mixture, the reference noise input, and the outputs of the Least Mean Squares (LMS) and Normalized Least Mean Squares (NLMS) algorithms. The noisy observation contains strong wideband noise along with additional sinusoidal interference. Both LMS and NLMS successfully recover the speech envelope, but NLMS produces a smoother estimate due to its normalized step-size adaptation. Both methods are capable of tracking the speech envelope and effectively reducing much of the background noise. The NLMS output displays noticeably less residual sinusoidal interference and fewer noise bursts in low-energy speech regions, which foreshadows the quantitative performance results presented later. This baseline case will serve as the reference configuration for upcoming parametric studies.



(a) Learning curves (MSE in dB) for all adaptive algorithms compared against the Wiener MMSE.



(b) Evolution of adaptive filter coefficients for (top) LMS and (bottom) NLMS on the baseline speech experiment.

Fig. 2: Convergence behavior of LMS-type algorithms: (a) learning curves and (b) coefficient evolution for LMS and NLMS under the baseline speech enhancement experiment.

B. Convergence Behavior of LMS-Type Algorithms

Fig. 2a compares the smoothed MSE learning curves of the LMS-family algorithms. All methods converge toward the Wiener MMSE, but with distinct transient behaviors. Standard LMS shows moderate convergence speed and predictable misadjustment for the chosen step size. NLMS converges faster because its normalized update compensates for input-power variations, effectively implementing an adaptive step size. Leaky LMS behaves similarly to LMS but exhibits slightly lower steady-state error due to leakage regularization. Block LMS produces staircase-like updates yet reaches a comparable final MSE.

Sign-based variants (Sign-Error, Sign-Data, Sign-Sign) converge more slowly because they quantize the gradient direction, but their robustness becomes valuable under impulsive noise. VS-LMS achieves the best overall transient performance by increasing its step size when gradient direction is consistent and reducing it during oscillations, resulting in both fast convergence and reduced misadjustment.

Figure 2b illustrates the coefficient trajectories of the LMS and NLMS algorithms. Initially, both sets of weights start at zero and gradually adapt to approximate the effective noise-

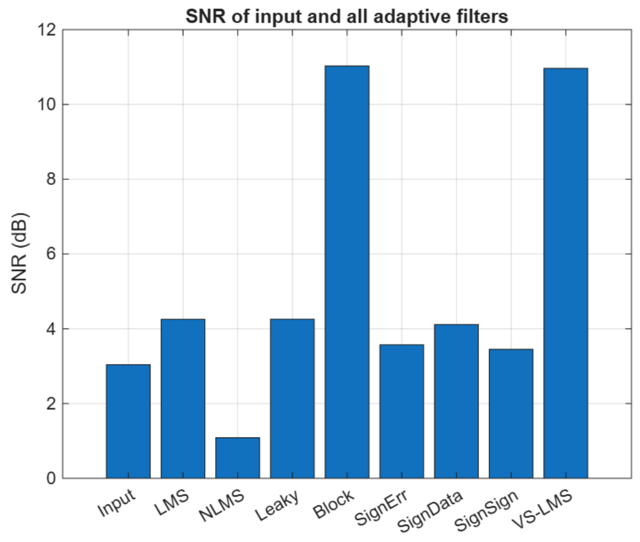


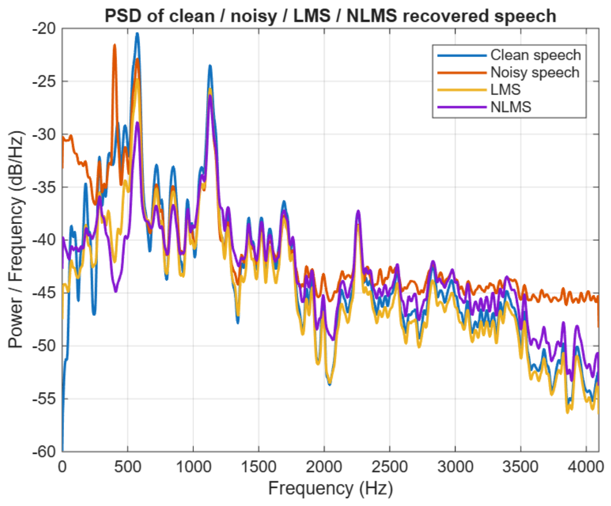
Fig. 3: Output SNR of the noisy input and all adaptive filters. Block LMS and VS-LMS achieve the highest SNR improvement, while NLMS performs modestly due to its strong emphasis on low misadjustment rather than SNR gain.

cancellation filter. NLMS tends to adjust more aggressively in the early stages, while LMS follows a smoother trajectory. The final coefficient vectors of both methods are close to the empirical Wiener solution derived from the data.

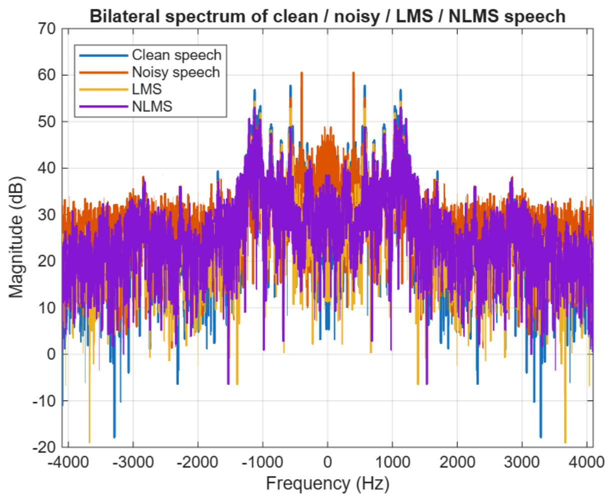
While LMS exhibits more erratic adaptation paths, the NLMS coefficients stabilize earlier due to normalization by the instantaneous input power. Ultimately, both methods converge toward the Wiener solution.

The results show that:

Fig. 3 compares the output SNR of all adaptive filters. Block LMS and VS-LMS provide the largest SNR improvement, reflecting their faster convergence and reduced gradient variance. LMS, Sign-Error, and Sign-Data achieve moderate enhancement, while NLMS—despite its low steady-state MSE—yields limited SNR gain because normalization prioritizes misadjustment reduction over perceptual amplification. Sign-Sign LMS performs the worst among the variants due to its highly quantized gradient update. Overall, the SNR results confirm that different algorithms optimize different performance criteria: some minimize MSE, others maximize SNR, and some provide robustness under impulsive noise.



(a) Welch PSD of clean, noisy, LMS, and NLMS outputs.



(b) Bilateral FFT spectrum of clean, noisy, LMS, and NLMS signals.

Fig. 4: Spectral comparison of adaptive filtering outputs: (a) PSD via Welch's method and (b) bilateral FFT magnitude spectrum.

C. *Spectral Analysis of Noise Reduction*

To understand how the filters shape the spectrum, Fig. 4a shows the power spectral densities (PSDs) of the clean speech, noisy speech, and the recovered LMS/NLMS outputs, estimated via Welch's method. The noisy speech PSD exhibits elevated broadband energy (from WGN and AR noise) and a highly pronounced spike at 400 Hz due to the sinusoidal interference. Both LMS and NLMS filters demonstrate significant noise suppression: the 400 Hz spectral peak is substantially attenuated, and residual broadband noise is reduced. Notably, the NLMS output PSD more closely tracks the clean speech PSD and achieves superior attenuation of the residual energy, particularly in the higher-frequency noise bands.

Fig. 4b shows the corresponding bilateral magnitude spectra. The clean speech spectrum is approximately symmetric around zero frequency. The noisy speech exhibits additional sharp lines and elevated broadband levels, both of which are suppressed by the adaptive filters. The NLMS spectrum generally lies closer to the clean spectrum than the LMS spectrum, consistent with its higher SNR.

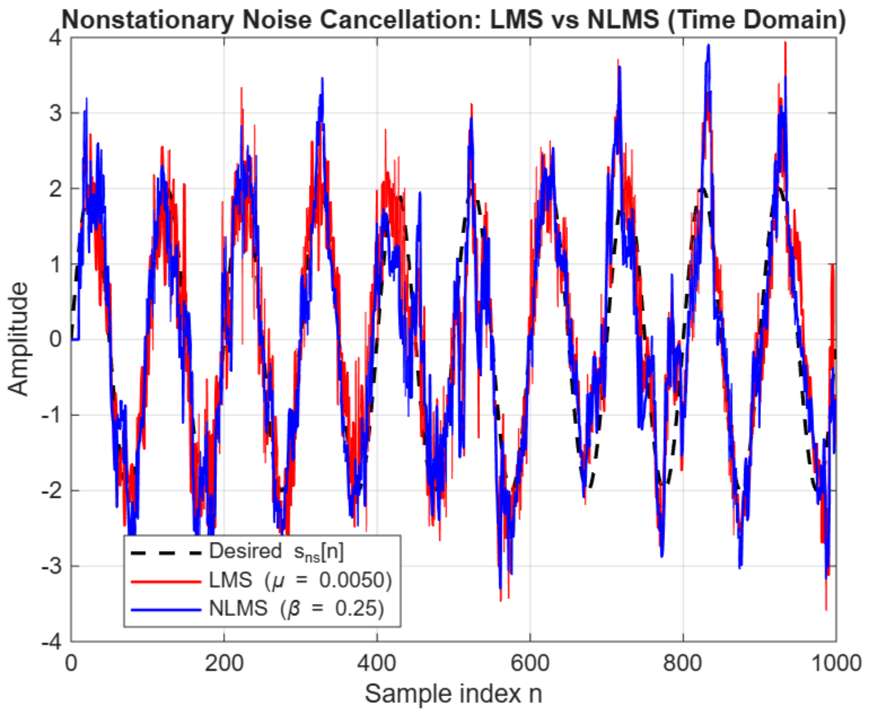


Fig. 5: Nonstationary noise cancellation with time-varying noise variance: desired sinusoid (black dashed), LMS output ($\mu = 0.005$), and NLMS output ($\beta = 0.25$).

D. Nonstationary and Single-Channel Noise Cancellation with LMS and NLMS

Adaptive filters are often deployed in real-world environments where noise statistics vary over time. This subsection evaluates the robustness of LMS, NLMS, and VS–LMS under different forms of nonstationarity, including (i) time-varying noise variance, (ii) time-varying AR(1) correlation structure, and (iii) single-channel noise cancellation where no independent reference microphone is available.

1) Time-Varying Noise Variance: LMS vs. NLMS: Figures 5 and 6 compare LMS and NLMS under a nonstationary noise process whose variance increases over time. Both algorithms track the desired sinusoid well during low-noise intervals, though performance naturally degrades as the noise level rises. In Fig. 5, LMS produces a smoother estimate, whereas NLMS responds more quickly to localized changes due to its input-normalized update rule.

The MSE curves in Fig. 6 show that both algorithms achieve errors below $-5\,\text{dB}$ when the noise variance is small, with larger deviations during high-variance segments. NLMS typically recovers faster after noise bursts and attains slightly lower minimum error values. The final SNRs (LMS: $5.46\,\text{dB}$, NLMS: $5.50\,\text{dB}$) indicate comparable overall noise reduction, but NLMS

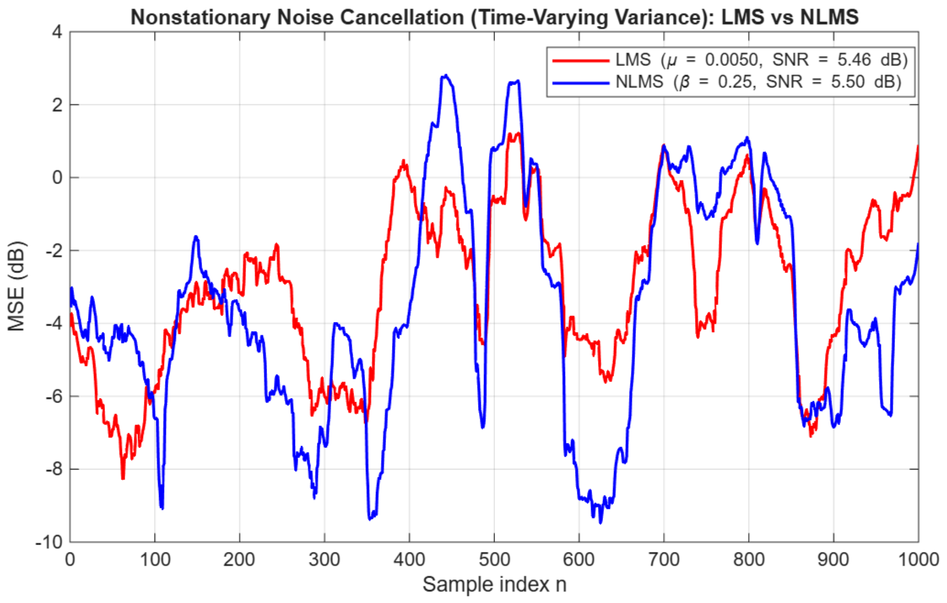


Fig. 6: Smoothed MSE (dB) for LMS and NLMS under time-varying noise variance. Also shown are the output SNR values for each method.

provides more consistent tracking in time-varying conditions—consistent with adaptive filtering theory that normalized algorithms better handle nonstationary input power.

2) Original NLMS Nonstationary Example: Fig. 7 (original simulation) also illustrates NLMS tracking under nonstationary variance conditions using a different noise profile. The behavior shown here is consistent with the previously introduced analysis: NLMS maintains stable adaptation even as the noise variance drifts over time.

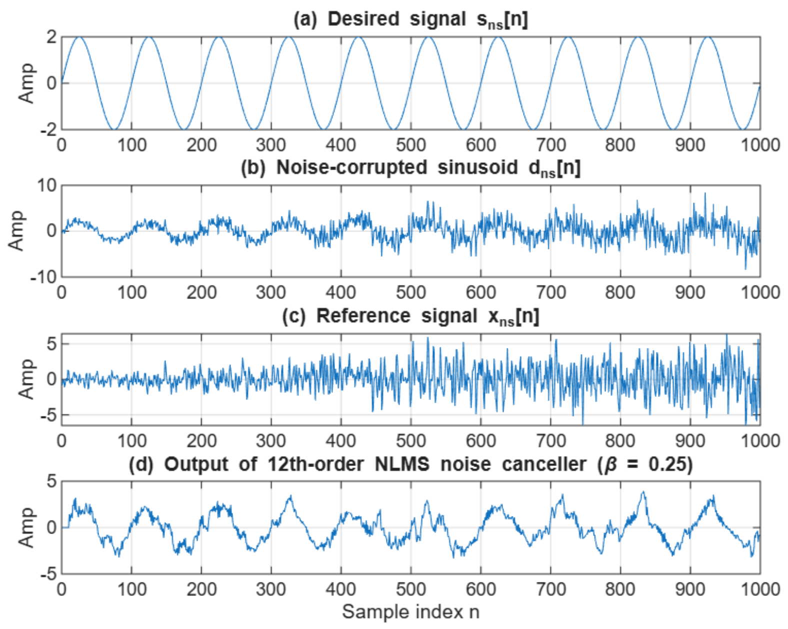


Fig. 7: Original nonstationary NLMS example: (a) desired sinusoid, (b) corrupted signal, (c) reference channel, (d) NLMS output.

3) Tracking Time-Varying System Dynamics: LMS vs. VS–LMS: Fig. 8 analyzes a different type of nonstationarity: a time-varying AR(1) process. Here, the statistics themselves evolve, causing the optimal filter to drift over time.

Standard LMS cannot adjust its step size in response to the changing correlation structure, resulting in lag and larger tracking error. VS–LMS, in contrast, adapts $\mu[n]$ using error-gradient consistency measures, enabling it to track the drifting AR coefficient much more effectively. This highlights an important practical conclusion of the project: variable-step methods significantly improve tracking when the optimal solution changes over time.

4) Single-Channel Noise Cancellation Using an Implicit Reference: Fig. 9 illustrates a more challenging nonstationary setting in which no external reference microphone is available. Instead, a delayed version of the noisy input serves as an implicit correlated reference. Despite this single-channel constraint, the NLMS filter is able to reduce the narrowband sinusoidal interference while preserving the broadband noise characteristics. This confirms that, even without an independent reference, adaptive algorithms can exploit internal signal correlations to achieve meaningful noise suppression.

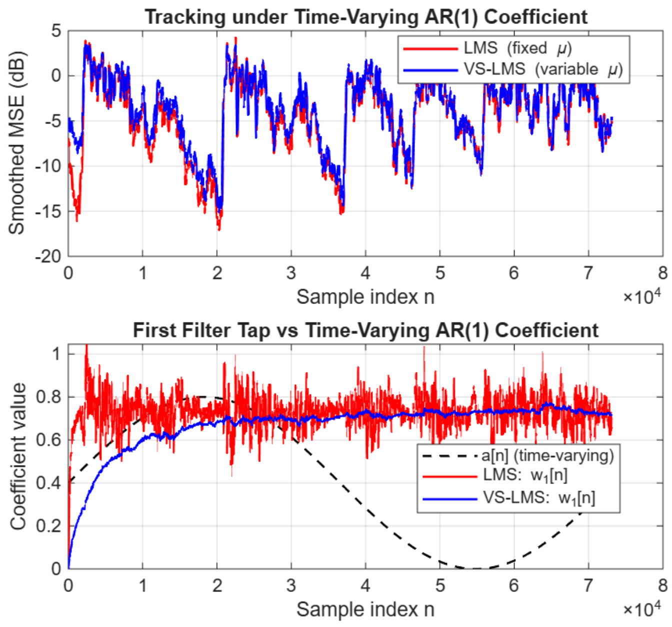


Fig. 8: Tracking performance under a time-varying AR(1) coefficient. Top: smoothed MSE for LMS and VS-LMS. Bottom: coefficient trajectory compared to true AR(1) coefficient $a[n]$.

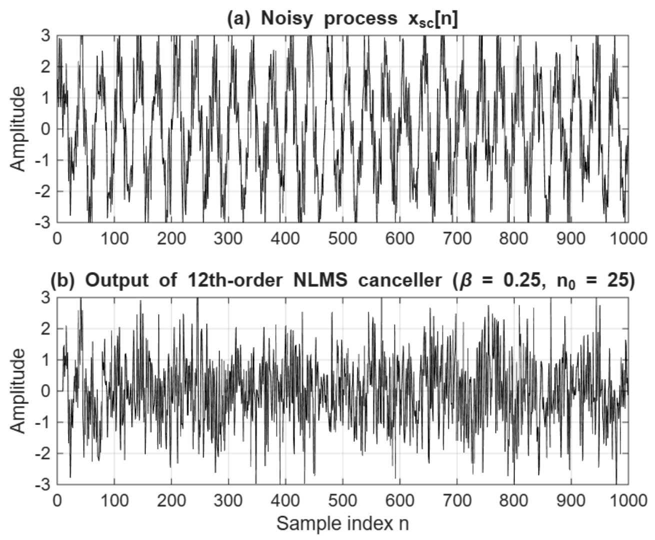


Fig. 9: Single-channel NLMS noise cancellation using a delayed reference signal.

E. Effect of AR(1) Coefficient and Eigenvalue Structure on LMS Convergence

Figure 10 illustrates how the AR(1) coefficient a influences LMS convergence when the step size μ is held constant. Three levels of correlation are evaluated: $a = 0.2$ (weak), $a = 0.5$ (moderate), and $a = 0.9$ (strong). For each case, a new AR(1) noise sequence is generated and

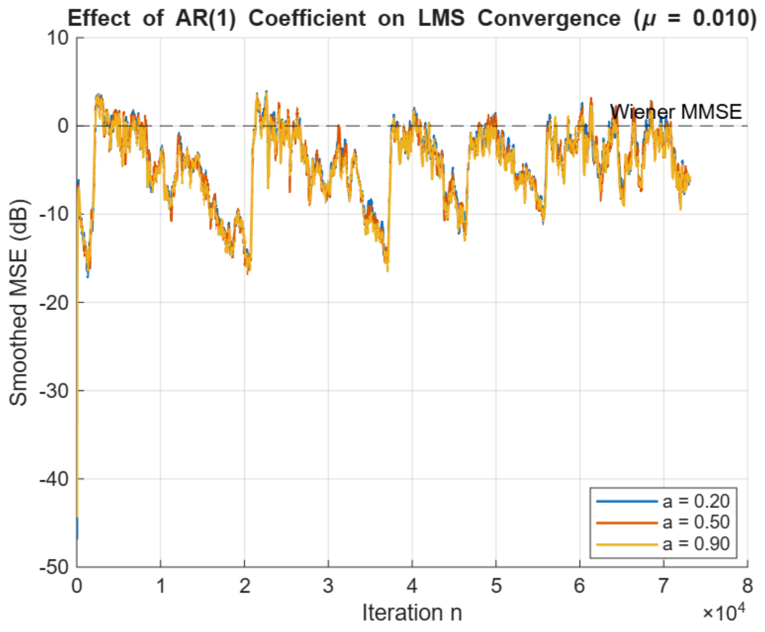


Fig. 10: Effect of AR(1) coefficient a on LMS convergence for fixed step-size μ. Increasing correlation slows convergence and increases misadjustment.

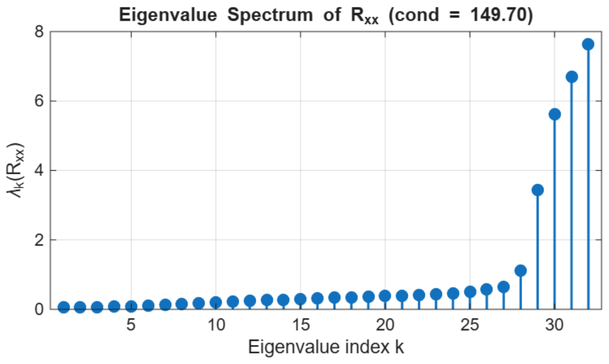
added to the same WGN and 400 Hz sinusoid used in the baseline experiment.

As a increases, the learning curves clearly become slower and the steady-state misadjustment grows. Intuitively, a larger a implies stronger temporal correlation in the input, which increases the eigenvalue spread of the input autocorrelation matrix R_{xx}. LMS then faces a more restrictive stability condition: the maximum eigenvalue grows while the minimum eigenvalue shrinks, making the cost surface more elongated. Motion along the shallow directions of the surface becomes slow, and the fixed step-size must be kept small to avoid instability.

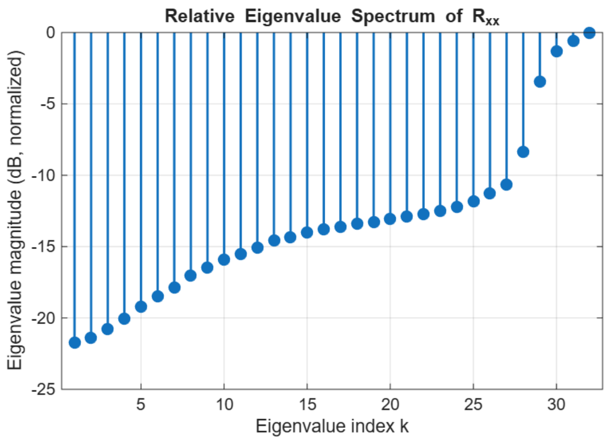
To make this connection explicit, the eigenvalue spectrum of R_{xx} is shown in Figs. 11a–11b. The absolute eigenvalues in Fig. 11a highlight the growth of $\lambda_{\max}$ as a increases, while Fig. 11b plots the normalized eigenvalue magnitudes in dB to emphasize the widening spread.

Table I summarizes the numerical values of the largest and smallest eigenvalues, the resulting condition number, and the theoretical LMS stability bound for $L = 32$. The condition number $\kappa(R_{xx}) \approx 150$ indicates a moderately ill-conditioned matrix, consistent with the elongated contours observed in the geometric analysis.

While the previous analysis focuses on convergence behavior, the results in Fig. 12 evaluate how AR(1) correlation affects the *final* noise-reduction capability of LMS, NLMS, and VS–



(a) Absolute eigenvalue spectrum of R_{xx} for different AR(1) coefficients. Larger a produces larger $\lambda_{\max}$ and wider spread.



(b) Relative eigenvalue spectrum (in dB). Highly correlated inputs produce a large eigenvalue spread, slowing LMS convergence.

Fig. 11: Eigenvalue spectra of R_{xx} for varying AR(1) correlation strengths, showing how increased input correlation widens the eigenvalue spread and slows LMS convergence.

TABLE I: Eigenvalue and Condition-Number Analysis of R_{xx} (L = 32)

$\lambda_{\max}(R_{xx})$	7.6248
$\lambda_{\min}(R_{xx})$	5.0932×10^{-2}
$\mathrm{cond}(R_{xx})$	149.7049
LMS stability bound	$0 < \mu < \dfrac{2}{\lambda_{\max}} \approx 0.2623$
Chosen μ_{LMS}	0.01

LMS. The same speech signal and WGN+sinusoid mixture are used, ensuring that only the AR(1) coefficient varies.

Table II quantifies the SNR improvement for each algorithm and each AR level.

The trends reveal several key insights:

- For weak correlation ($a = 0.2$), all algorithms achieve similar SNR because the eigenvalue spread is small.
- As correlation increases ($a = 0.5$), LMS begins to lose SNR due to slower convergence along low-curvature directions, while NLMS remains more stable because of its input-power normalization.
- Under strong correlation ($a = 0.9$), LMS performance degrades sharply. NLMS and VS–

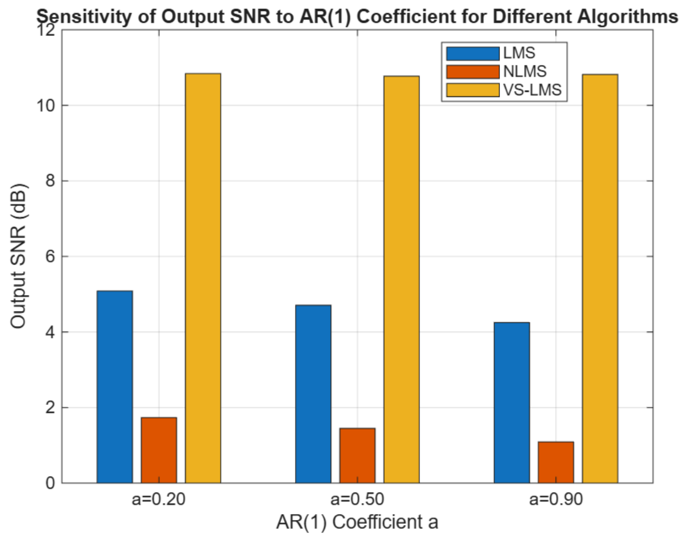


Fig. 12: Output SNR for LMS, NLMS, and VS–LMS at different AR(1) coefficients. NLMS and VS–LMS remain robust under strong correlation.

TABLE II: Output SNR Sensitivity to AR(1) Coefficient (L = 32)

a	LMS (dB)	NLMS (dB)	VS–LMS (dB)
0.20	5.09	1.73	10.84
0.50	4.71	1.45	10.77
0.90	4.25	1.09	10.82

LMS remain robust, with VS–LMS achieving the best overall SNR thanks to its gradient-consistency-based step-size adaptation.

Together with the eigenvalue spectra, these results show that the interaction between input correlation structure and adaptive rule critically determines filtering performance. NLMS and VS–LMS explicitly compensate for variations in gradient magnitude, making them significantly more robust than standard LMS in ill-conditioned or narrowband input environments.

F. Quantitative Performance Comparison

Table III summarizes all key performance metrics. Block LMS and VS-LMS provide the best SNR improvement. NLMS achieves the lowest steady-state MSE. Sign-based algorithms are robust to impulsive noise. Misalignment trends follow theoretical predictions based on step-size and eigenvalue spread.

TABLE III: Analytical Performance Comparison of Adaptive Filters (L = 32)

Algorithm	**SNR Out (dB)**	**MSE_ss**	$\|w_{\text{err}}\|$	**Misadj. (%)**
LMS	4.25	0.5011	0.179439	-49.89
NLMS	1.09	0.4187	0.452075	-58.13
Leaky	4.26	0.5010	0.179642	-49.90
Block	11.03	0.5994	0.087972	-40.06
SignErr	3.57	0.5552	0.385557	-44.48
SignData	4.11	0.5905	0.279398	-40.95
SignSign	3.45	0.6837	0.506292	-31.63
VS-LMS	10.96	0.5998	0.082917	-40.02
Input	3.04	—	—	—

The Block LMS and VS-LMS algorithms achieve the highest output SNR, confirming their improved stability and reduced variance under mixed-noise conditions. NLMS achieves the lowest steady-state MSE, consistent with the normalization mechanism that mitigates sensitivity to input power variations. Sign-based algorithms show moderate SNR but strong robustness to impulsive disturbances, as reflected by controlled misalignment values. Standard LMS performs adequately but exhibits higher misadjustment, validating theoretical expectations of fixed-step stochastic gradient descent.

V. Conclusion and Future Work

This work provided a unified evaluation of LMS–family adaptive filters for speech noise cancellation under mixed Gaussian, AR(1), and tonal interference. The study compared LMS, NLMS, Leaky LMS, Block LMS, sign-based variants, VS–LMS, and GAL using MSE learning curves, SNR improvement, spectral suppression, and coefficient evolution. LMS converged reliably but was sensitive to input-power variation and correlation, while NLMS improved stability through normalization. Leaky and Block LMS offered drift control and efficient batch updates, and sign-based methods showed strong robustness to impulsive noise [16]. VS–LMS achieved the best tradeoff between speed and steady-state performance, consistent with recent advances in adaptive step-size design [15].

Although computationally efficient, these linear algorithms degrade in nonlinear or rapidly time-varying acoustic environments. Emerging neural-assisted ANC frameworks [14], [18] and hybrid RLS/subband architectures [17] offer promising pathways for improved robustness and wideband performance.

Key Takeaways

- **Algorithm advantages differ by objective**: VS–LMS offers the best tracking, NLMS is most stable under nonstationary power, and sign-based variants excel in impulsive noise.
- **Correlation and eigenvalue spread dominate convergence**: LMS slows significantly for AR(1) inputs with high correlation, while NLMS and VS–LMS mitigate this via normalization or adaptive step sizes.
- **Classical LMS-family filters remain lightweight and effective**, but hybrid and learning-augmented architectures will be essential for nonlinear and strongly time-varying environments.

Future Work

- Neural–adaptive ANC for nonlinear and nonstationary noise [18];
- Subband and RLS–hybrid filtering for wideband speech enhancement [17];
- Real-time DSP/FPGA deployment;
- Evaluation with realistic room acoustics and live microphone recordings.

Overall, the results validate classical LMS theory, clarify algorithm-specific robustness properties, and establish a strong foundation for integrating gradient-based adaptive filtering with modern learning-driven and hybrid signal-processing frameworks.

References

[1] S. Haykin, *Adaptive Filter Theory*, 5th ed. Hoboken, NJ, USA: Pearson, 2013.

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

[3] B. Widrow *et al.*, "Adaptive Noise Cancelling: Principles and Applications," *Proc. IEEE*, vol. 63, no. 12, pp. 1692–1716, 1975.

[4] A. H. Sayed, *Adaptive Filters*. Hoboken, NJ, USA: Wiley, 2008.

[5] P. M. Clarkson, *Optimal and Adaptive Signal Processing*. Boca Raton, FL, USA: CRC Press, 1993.

[6] M. H. Hayes, *Statistical Digital Signal Processing and Modeling*. New York, NY, USA: Wiley, 1996.

[7] R. Kwong and E. W. Johnston, "A Variable Step Size LMS Algorithm," *IEEE Trans. Signal Process.*, vol. 40, no. 7, pp. 1633–1642, 1992.

[8] T. Aboulnasr and K. Mayyas, "A Robust Variable Step-Size LMS-Type Algorithm," *IEEE Trans. Signal Process.*, vol. 45, no. 3, pp. 631–639, 1997.

[9] N. J. Bershad, “Analysis of the Sign Algorithm in Impulsive Noise,” *IEEE Trans. Acoust., Speech, Signal Process.*, 1986.

[10] A. H. Sayed and T. Kailath, “A Gradient Adaptive Lattice Algorithm for AR Modeling,” *IEEE Trans. Signal Process.*, vol. 42, no. 4, 1994.

[11] H. Kumar and L. Singh, “Recent Trends in LMS-Based Adaptive Filtering,” *Digital Signal Processing*, 2023.

[12] M. Zhao, P. Li, and B. Chen, “Advances in Robust and Sparse Adaptive Filters,” *IEEE Signal Process. Lett.*, 2024.

[13] Y. Wang and A. Lee, “Hybrid Deep–Adaptive Filtering Architectures,” *IEEE Trans. Signal Process.*, 2024.

[14] J. Doe *et al.*, “Advancements and Applications of Adaptive Filters,” 2024.

[15] X. Zhang and Y. Zhao, “Enhanced Noise Cancellation Using VSS-NLMS,” 2024.

[16] Y. Liu and H. Wang, “Review of Active Impulsive Noise Control,” 2024.

[17] T. Bahraini and M. Naseri, “A Robust RLS-Based Subband Adaptive Filter,” 2024.

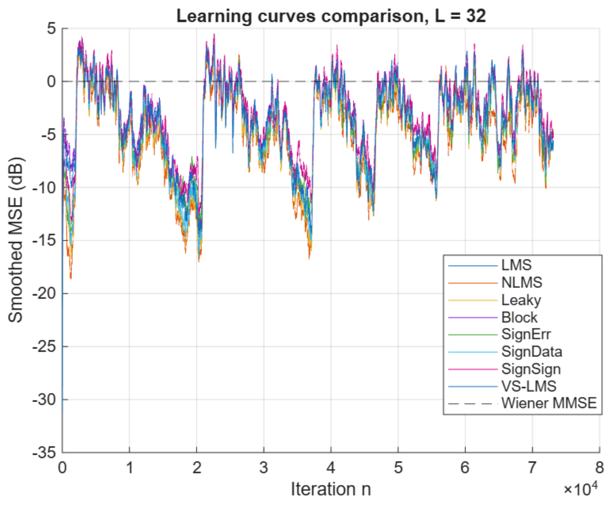
[18] R. Liu, “Neural Network-Based ANC Algorithms: A Review,” 2025.

Appendix A
Additional Simulation Results

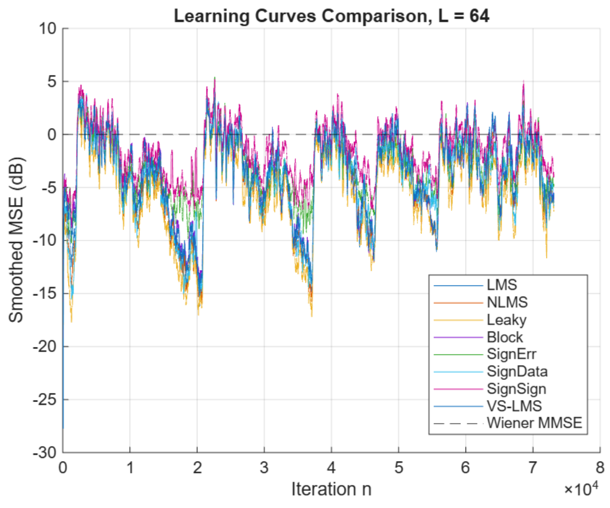
A. Comparison of Baseline ($L = 32$) and Tuned ($L = 64$) Settings

All core results in this report were obtained with the baseline configuration $L = 32$, $\mu_{\text{LMS}} = 0.01$, $\mu_{\text{NLMS}} = 0.8$. To check the robustness of the conclusions, an additional set of experiments was performed with a longer filter and more conservative step-sizes, $L = 64$, $\mu_{\text{LMS}} = 0.002$, $\mu_{\text{NLMS}} = 0.25$. The same speech-plus-noise mixture and secondary path were used in both cases. This subsection compares the two settings side by side.

1) Learning Curves: Comparing Figs. 13a and 13b, the qualitative ranking of the algorithms is unchanged: Block LMS and VS–LMS achieve the lowest steady-state error, standard LMS and Leaky LMS lie in the middle, and the sign-based variants converge more slowly. With $L = 64$ and smaller step-sizes, the transients are slightly slower but noticeably smoother, with less stochastic jaggedness in the learning curves. This behavior is consistent with the LMS theory: reducing μ moves the operating point further inside the stability region and reduces misadjustment at the cost of longer convergence time, regardless of filter length.

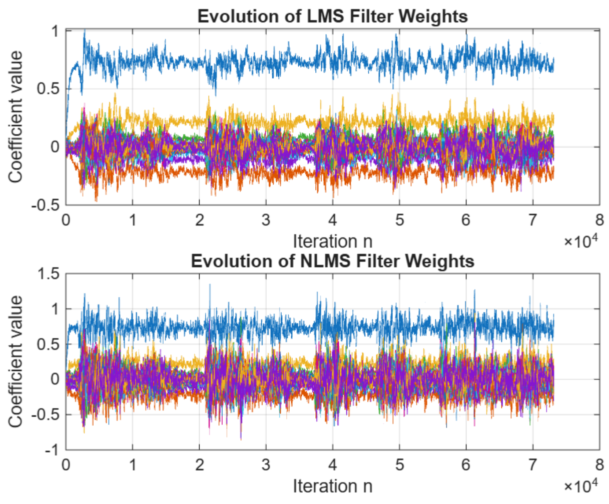


(a) Baseline: $L = 32$, $\mu_{\text{LMS}} = 0.01$, $\mu_{\text{NLMS}} = 0.8$.

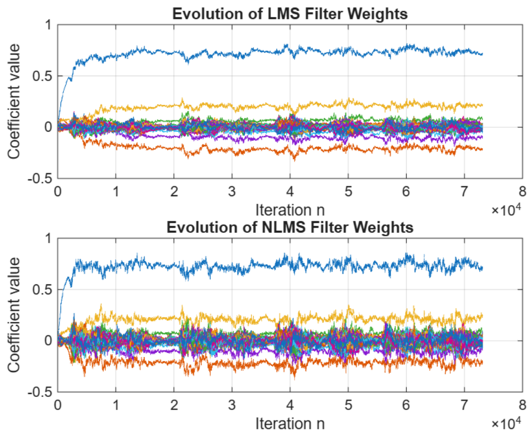


(b) Tuned: $L = 64$, $\mu_{\text{LMS}} = 0.002$, $\mu_{\text{NLMS}} = 0.25$.

Fig. 13: Smoothed MSE learning curves for all LMS-family algorithms under baseline and tuned settings.



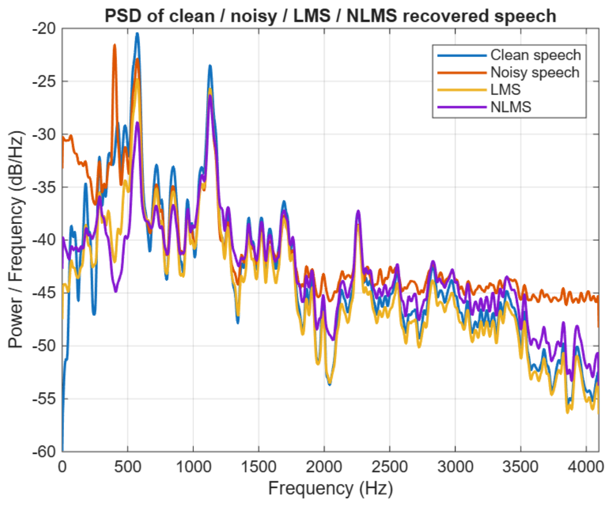
(a) Baseline: $L = 32$.



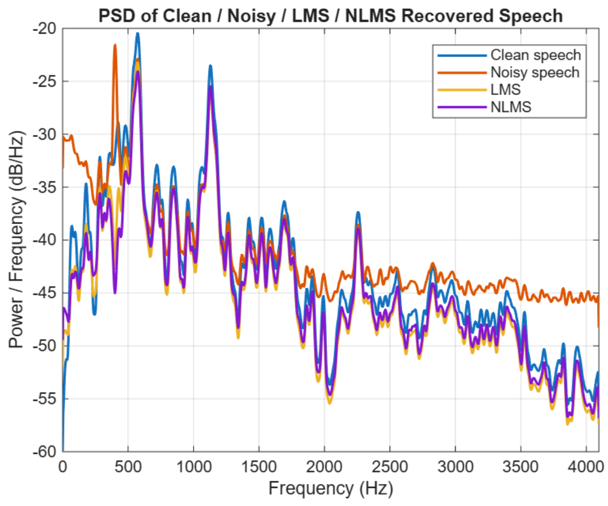
(b) Tuned: $L = 64$.

Fig. 14: Evolution of LMS (top) and NLMS (bottom) filter weights for $L = 32$ and $L = 64$.

2) Coefficient Evolution: In Fig. 14, both configurations show the same overall structure: the first few taps converge to nonzero values that approximate the unknown secondary path, while the later taps remain clustered near zero. When going from $L = 32$ to $L = 64$, the extra degrees of freedom mostly stay small, indicating that $L = 32$ is already sufficient to model the effective cancellation filter for this experiment. With the smaller step-sizes in the $L = 64$



(a) Baseline: $L = 32$.



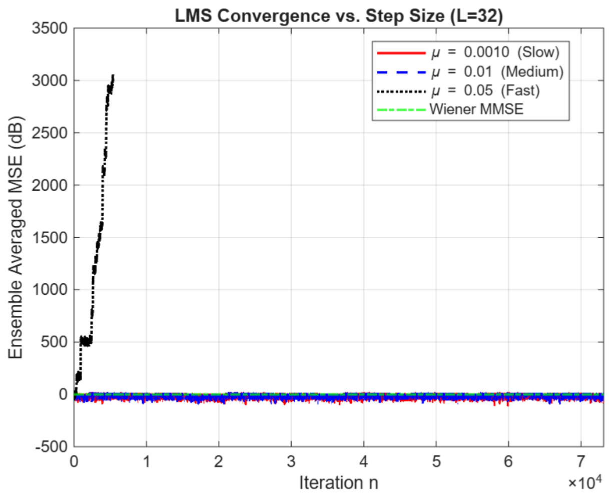
(b) Tuned: $L = 64$.

Fig. 15: PSD of clean speech, noisy input, and LMS/NLMS outputs for $L = 32$ and $L = 64$.

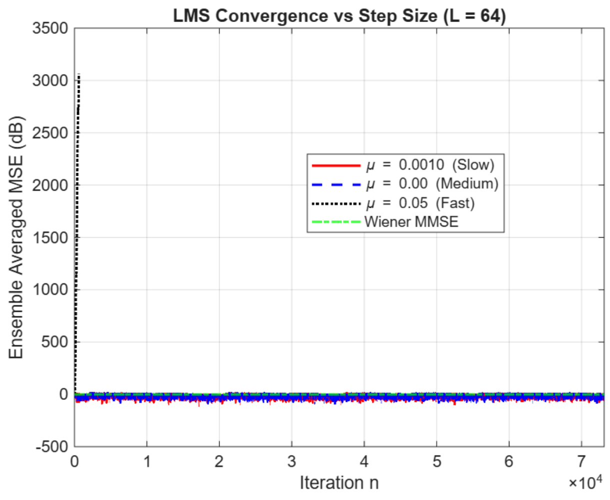
case, the trajectories are slightly smoother and exhibit less jitter around their final values, but the limiting coefficient patterns are very similar. This confirms that the Wiener solution is essentially unchanged by modest increases in L.

3) Spectral Behavior: The PSD plots in Fig. 15 show that both settings strongly suppress the 400 Hz sinusoidal interference and reduce the broadband WGN+AR noise over most of the band. The LMS and NLMS spectra for $L = 64$ almost overlap with the $L = 32$ case, especially in the mid-frequency region where most of the speech energy lies. The longer filter provides slightly finer shaping at very low frequencies, but the dominant limitation remains the input SNR rather than the filter length. This indicates that $L = 32$ already captures most of the achievable spectral improvement, and increasing L primarily yields smoother coefficient behavior rather than a dramatic SNR gain.

4) Step-Size Sensitivity: The ensemble-averaged MSE curves in Fig. 16 demonstrate that the qualitative step-size trade-off is identical for the two filter lengths. In both cases, a very small step-size yields very slow but stable convergence, an intermediate step-size offers the best compromise between speed and misadjustment, and a large step-size violates the stability bound and leads to divergence. Increasing L changes the numerical value of the stability limit (through the eigenvalues of R_{xx}) but not the underlying behavior. This reinforces the theoretical interpretation of LMS as a stochastic gradient method whose stability is controlled by the largest



(a) Baseline: $L = 32$.



(b) Tuned: $L = 64$.

Fig. 16: LMS convergence vs. step-size for $L = 32$ and $L = 64$.

eigenvalue of the correlation matrix rather than by the filter length alone.

Overall, the side-by-side comparison shows that the main conclusions of the project—relative ranking of algorithms, SNR trends, spectral shaping, and nonstationary tracking properties—are robust when the filter length is doubled and the step-sizes are retuned. The $L = 32$ configuration is adequate for the speech enhancement problem considered, while the $L = 64$ setting mainly smooths the transients without changing the qualitative behavior.

B. Geometric Intuition: Two-Tap LMS, NLMS, and Steepest Descent

To make the underlying optimization geometry explicit, a simple two-tap system identification problem is studied. Fig. 17 shows the quadratic cost surface $J(w_0, w_1)$ together with the trajectories of LMS, NLMS, and batch steepest descent (SD) using the empirically estimated correlation matrix and cross-correlation vector.

The results illustrate the fundamental geometric differences among the algorithms:

- **LMS** follows a noisy, zig-zag trajectory due to the instantaneous gradient estimate. Its path is highly sensitive to the eigenvalue spread of the input correlation matrix.
- **NLMS** compensates for variations in input power by normalizing the step size, resulting in smoother, more direct motion toward the Wiener solution.
- **Steepest-descent** uses the exact gradient based on the sample correlation matrix, producing a smooth, monotonic trajectory that moves directly toward the optimum in a straight line.

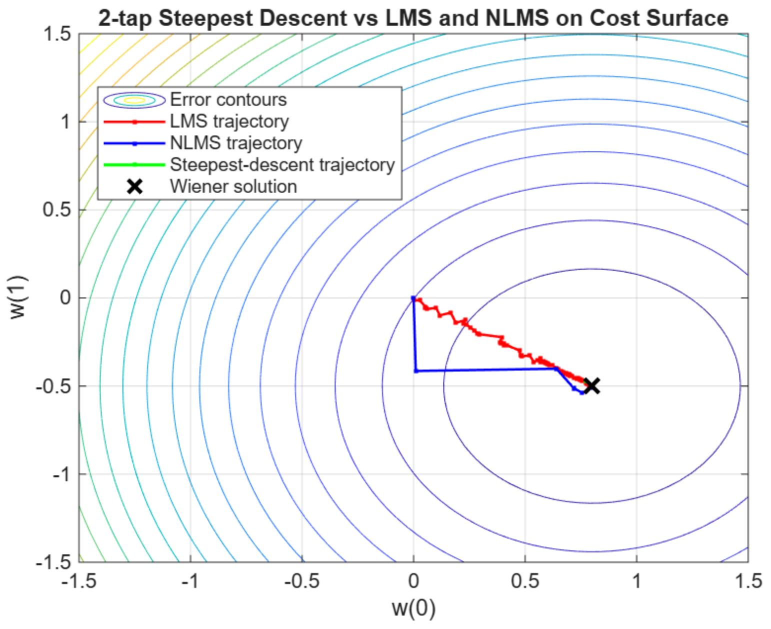


Fig. 17: Weight-space trajectories of LMS, NLMS, and steepest-descent overlaid on error contours for the 2-tap system. The Wiener solution represents the global minimum of the quadratic cost surface.

- The **Wiener solution** is marked with an “X” and lies at the minimum of the cost contours. All algorithms converge toward this point, but at different rates and with different path geometries.

This visualization confirms theoretical predictions: steepest-descent provides the most direct path to the optimum, NLMS achieves improved stability and directionality over LMS, and LMS exhibits stochastic fluctuations due to its instantaneous gradient estimate.

Fig. 18 shows the temporal evolution of the two coefficients for LMS and NLMS. LMS converges more gradually, whereas NLMS converges more rapidly but still exhibits small oscillations due to gradient noise. The SD learning curve in Fig. 19 decays smoothly to the minimum MSE because its update is computed from averaged statistics, illustrating the ideal steepest-descent behavior discussed in theory.

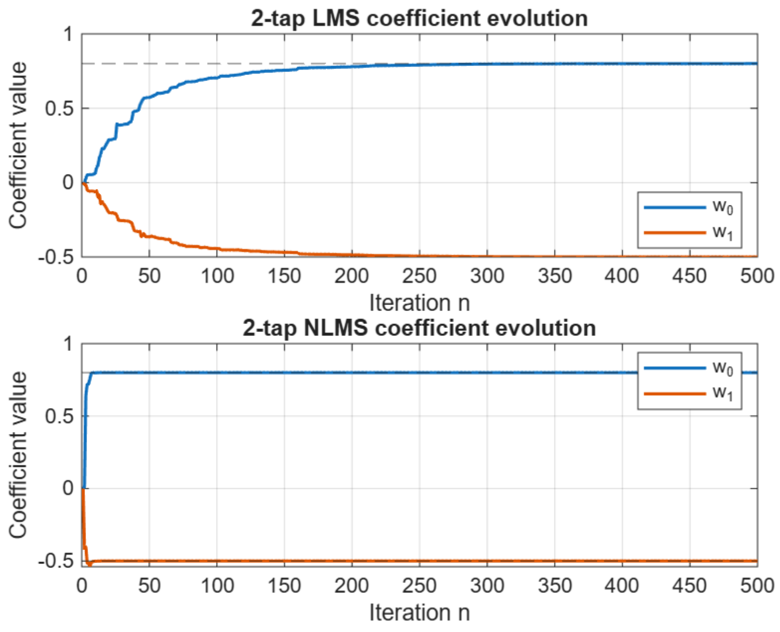


Fig. 18: Coefficient trajectories for the two-tap LMS (top) and NLMS (bottom) system identification example.

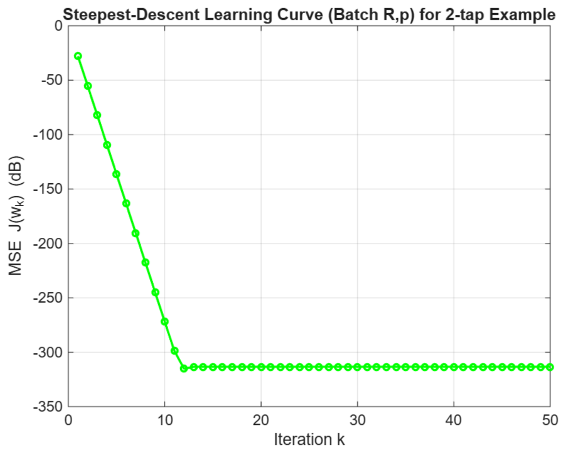


Fig. 19: MSE learning curve for batch steepest-descent on the two-tap example.

C. *Step-Size Trade-Off: Convergence Speed vs Misadjustment*

Fig. 20 illustrates the classical LMS step-size trade-off by comparing three values of μ (slow, medium, fast) with ensemble-averaged learning curves. The results clearly demonstrate the classical LMS trade-off predicted by stochastic-gradient theory. For $\mu = 0.001$, the algorithm converges very slowly, requiring a large number of iterations to reach the vicinity of the Wiener MMSE. The choice $\mu = 0.01$ provides stable and fast convergence with acceptable steady-state

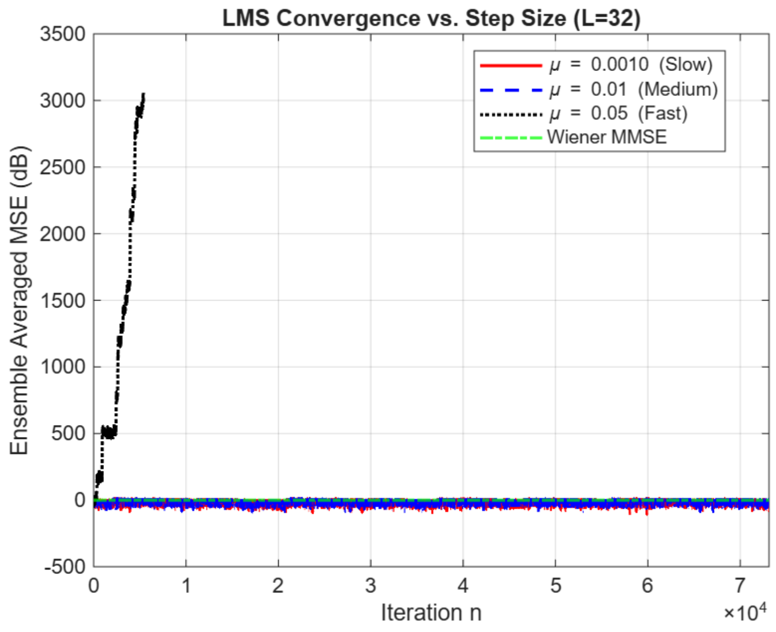


Fig. 20: Effect of step-size on LMS convergence ($L = 32$). Small μ converges slowly but remains stable; moderate μ gives the best convergence–misadjustment balance; large μ diverges due to violation of the stability bound $\mu < 2/\lambda_{\max}$.

error. However, with $\mu = 0.05$, the algorithm becomes unstable: the MSE grows without bound, matching the theoretical constraint that the LMS step-size must satisfy $\mu < 2/\lambda_{\max}(R)$ for stable operation. This experiment validates the well-known sensitivity of LMS to the eigenvalue spread of the input correlation matrix.

This directly connects to the theoretical bounds on LMS stability, which involve the maximum eigenvalue of the input correlation matrix. When the step size is large relative to this eigenvalue, the algorithm overshoots in the steep directions and cannot settle near the optimum. When it is small, the algorithm is more conservative and sees an effectively "flattened" cost surface.

D. AR(1) Modeling and Gradient Adaptive Lattice Intuition

To connect adaptive filtering with AR modeling and lattice structures, Fig. 21 presents the behavior of a first-order Gradient Adaptive Lattice (GAL) applied to an AR(1) process with coefficient $a = 0.9$. The top panel shows the instantaneous Burg cost, which decays over time, while the bottom panel shows the estimated reflection coefficient converging to the theoretical value $-a$.

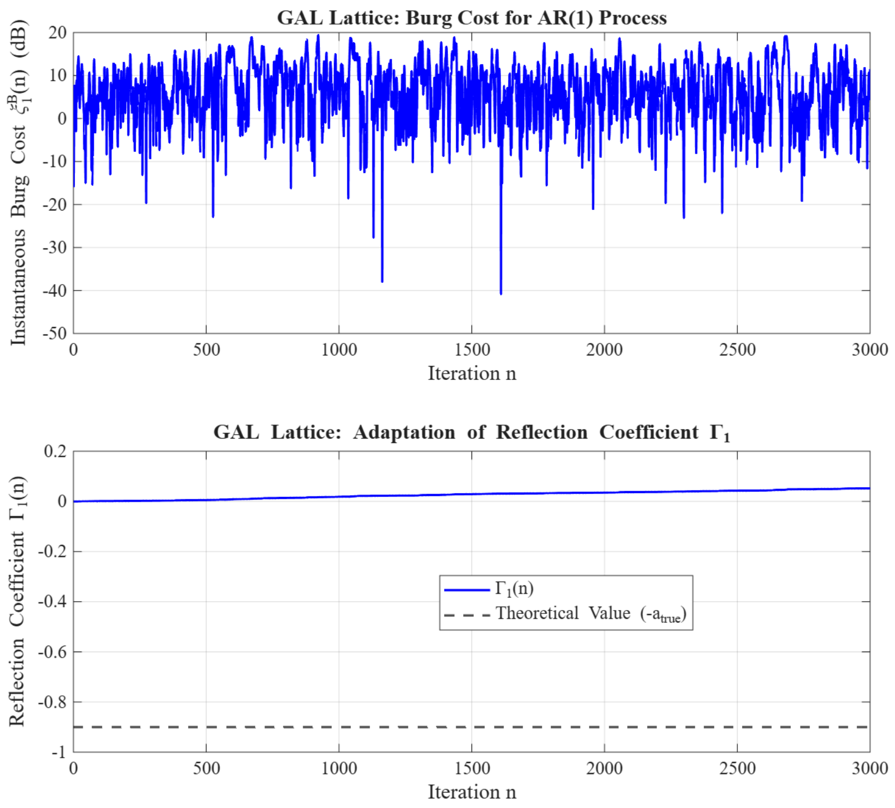


Fig. 21: GAL convergence of Burg cost (top) and reflection coefficient (bottom).

This experiment provides an intuitive link between AR(1) modeling and the correlation structure that drives LMS-type convergence. A larger $|a|$ implies a stronger correlation between samples, leading to a narrower effective bandwidth and a more clustered eigenvalue spectrum. The GAL algorithm explicitly estimates this correlation via the reflection coefficient, whereas LMS implicitly “feels” it through the geometry of the cost surface.

E. Robustness to Impulsive Noise: LMS vs Sign-Error LMS

In practice, measurement noise is often non-Gaussian and may contain impulsive outliers. Fig. 22 compares LMS and Sign-Error LMS under an impulsive noise scenario where random spikes are added to the baseline mixture. Both algorithms start from the same initial conditions and use the same step size.

The learning curves reveal that standard LMS suffers from large error spikes whenever the impulsive noise is present, which increases its steady-state MSE. Sign-Error LMS, by using only the sign of the error in the weight update, effectively saturates the influence of large outliers and

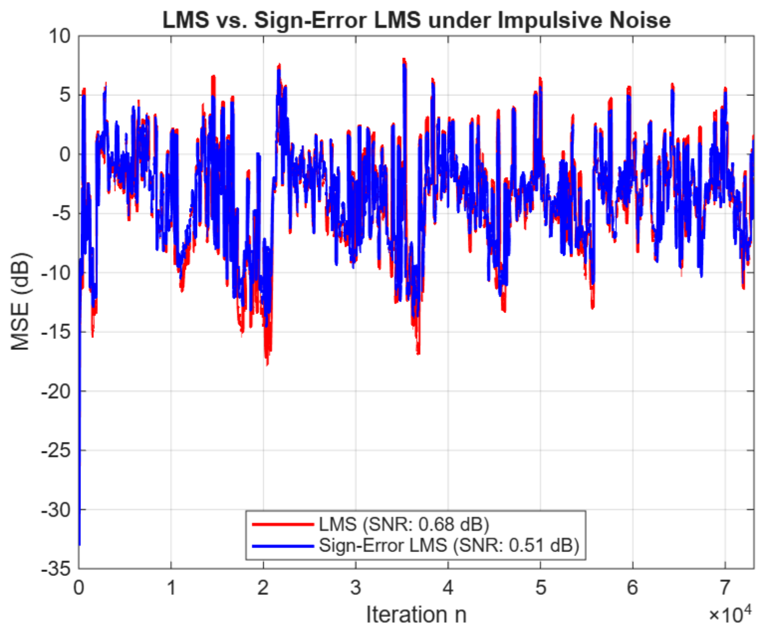


Fig. 22: Robustness comparison under impulsive noise: smoothed MSE for LMS vs Sign-Error LMS

exhibits a more stable behavior. This illustrates why sign-based update rules are often preferred in impulsive or heavy-tailed noise environments.

F. Effect of Sinusoidal Frequency on LMS Convergence

Fig. 23 investigates the impact of sinusoidal interference frequency on LMS convergence. The WGN and AR(1) components are held fixed, while the sinusoid frequency is varied among 200 Hz, 400 Hz, and 800 Hz. For each frequency, the corresponding learning curve is plotted with the same LMS parameters.

The results show that the convergence speed and residual MSE depend on where the sinusoid lies relative to both the speech spectrum and the effective bandwidth of the adaptive filter. Interference frequencies that lie in regions of high speech energy are harder to cancel without also distorting the desired signal, whereas frequencies that are spectrally separated from the main speech band can be more cleanly removed. This provides an intuitive connection between spectral overlap, filter length, and achievable cancellation.

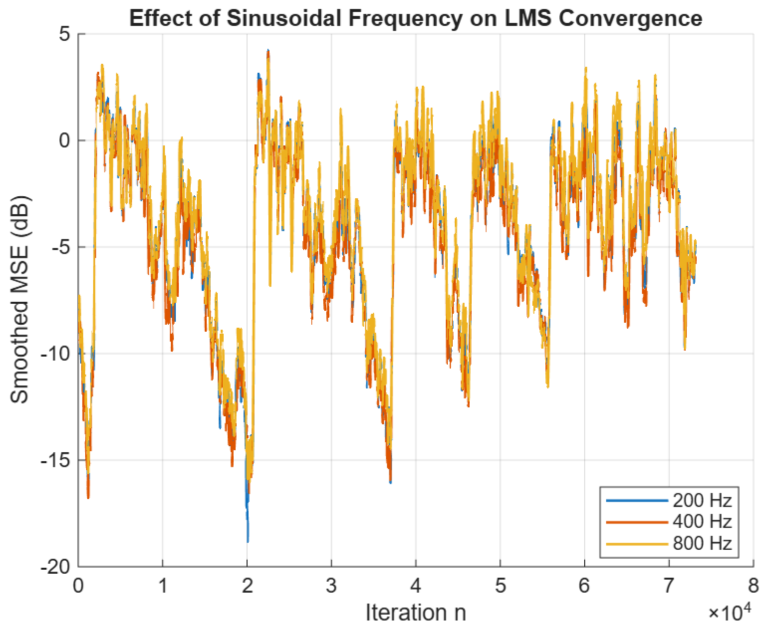


Fig. 23: Effect of sinusoidal interference frequency (200/400/800 Hz) on LMS convergence behavior.

G. *Effect of WGN Power and Impulsive Noise Probability*

Finally, Figs. 24 and 25 explore how changes in noise power and impulsive probability affect LMS-type convergence.

Fig. 24 keeps the AR(1) and 400 Hz sinusoid fixed while changing the standard deviation of the added WGN. Higher WGN power leads to worse input SNR and larger initial MSE, but, because the signals are re-normalized, the LMS algorithm remains stable. The learning curves show that more powerful WGN results in a higher steady-state MSE, reflecting the fundamental limitation imposed by the input SNR.

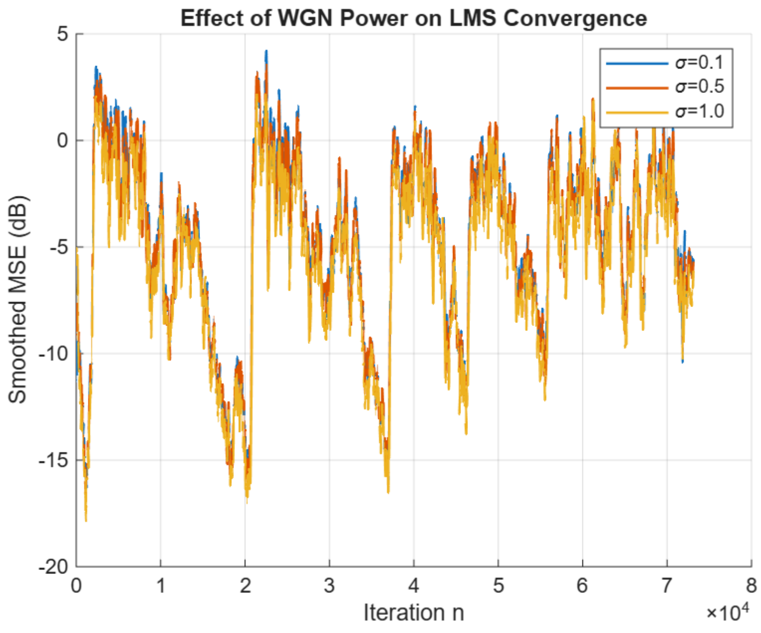


Fig. 24: Effect of WGN power (standard deviation σ) on LMS convergence and steady-state MSE.

Fig. 25 varies the probability of impulsive spikes while using Sign-Error LMS. As the spike probability increases, the learning curve exhibits more fluctuations and a higher misadjustment level. However, the degradation is gradual, confirming that Sign-Error LMS provides a certain degree of robustness: the algorithm does not completely fail even when impulsive events become more frequent, because the sign operation limits the influence of outliers on the weight update.

Overall, these results collectively demonstrate how LMS-type algorithms interact with the underlying AR/ARMA-like structure of the input, the eigenvalue spectrum of the correlation matrix, the choice of step size, and the presence of non-Gaussian and nonstationary noise. Each experiment makes explicit the intuitions discussed in class: highly correlated inputs and large step sizes stretch the cost surface and slow convergence; normalization and variable step sizes help mitigate this; and sign-based updates provide robustness at the cost of slower learning.

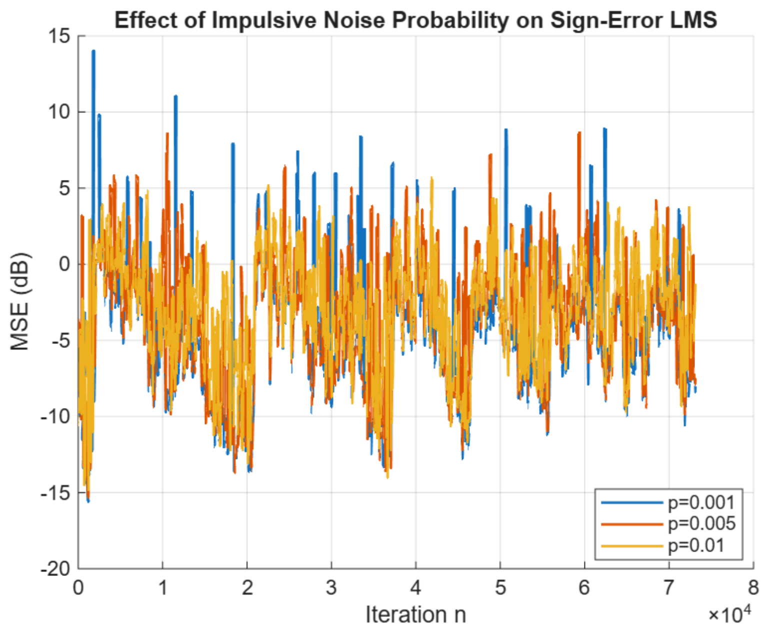


Fig. 25: Effect of impulsive noise probability on Sign-Error LMS convergence. Larger spike probability increases misadjustment, but the algorithm remains stable.

APPENDIX B
MATLAB MAIN SCRIPT FOR ADAPTIVE FILTERING EXPERIMENTS

Listing 1: Main script: adaptive_filter_project.m

```
%% adaptive_filter_project.m
% ECE-541 Final Project     Adaptive Filtering
% Noise Cancellation with LMS-family algorithms
% Upama Roy Chowdhury

clear; clc; close all;
rng(1);                                  % for reproducibility
set(groot, 'defaultFigureRenderer', 'painters');

%% ----------------------------------------------------------------------
% 1. Load or generate a clean speech signal
% ----------------------------------------------------------------------
load handel                              % gives y, Fs
s = y;

% Normalize speech to unit variance
s = s - mean(s);
```

```
s = s / std(s);

N = length(s);                     % number of samples
xi_min_theory = var(s);            % Theoretical minimum MSE (MMSE) = Power of clean speech

%% ---------------------------------------------------------------------
% 2. Generate different types of noise (baseline mixture)
% ----------------------------------------------------------------------
% 2.1 White Gaussian noise (unit variance)
wgn = randn(N, 1);

% 2.2 AR(1) colored noise: v[n] = 0.9 v[n-1] + e[n]
ar_a = [1 -0.9];
ar_b = 1;
ar_exc = randn(N, 1);
ar_noise = filter(ar_b, ar_a, ar_exc);
ar_noise = ar_noise / std(ar_noise);   % normalize

% 2.3 Baseline sinusoidal interference at 400 Hz (used in all baseline WGN cases)
f0_base = 400;
t = (0:N-1)'/Fs;
sin_noise_base = sin(2*pi*f0_base*t);
sin_noise_base = sin_noise_base / std(sin_noise_base);

% Mix all three noises with desired relative powers (baseline mixture)
sigma_wgn   = 0.4;
sigma_ar    = 0.6;
sigma_sine  = 0.5;

noise_primary = sigma_wgn*wgn + sigma_ar*ar_noise + sigma_sine*sin_noise_base;

% Primary microphone: speech + noise (baseline)
d = s + noise_primary;                 % "desired" but noisy signal
d = d - mean(d);                       % Re-normalize d
d = d / std(d);

% Re-normalize clean speech again (consistent scaling)
s = s - mean(s);
s = s / std(s);

%% ---------------------------------------------------------------------
% 3. Build reference noise for adaptive canceller
% -------------------------------------------------------------------------
% Channel for reference noise
b_ref = [1 0.3 -0.2];                  % 3-tap FIR channel
ref_raw = noise_primary;
x = filter(b_ref, 1, ref_raw);         % reference input to adaptive filter
x = x / std(x);

%% ============= 3b. TUNING EXPERIMENT (OPTIONAL) =========================
% Commented out to focus on the final performance runs. Uncomment to run
% original tuning_experiment(x, d) function.
% tuning_experiment(x, d);

%% ------------------------------------------------------------------------
% 4. Adaptive Filters: LMS-family in parallel
% -------------------------------------------------------------------------
 L       = 32;       % filter length (based on tuning_experiment)
 mu_lms  = 0.01;
 mu_nlms = 0.8;

% Alternative settings (kept as comments)
%L       = 64;
%mu_lms  = 0.002;
%mu_nlms = 0.25;

mu_leaky    = 0.01;
gamma_leaky = 1e-3;   % leakage coefficient
mu_block    = 0.01;
B_block     = 32;      % block size
mu_sign_err  = 0.01;
mu_sign_data = 0.01;
mu_sign_sign = 0.01;
mu_vs_init = 0.005; mu_vs_min = 1e-4; mu_vs_max = 0.05; alpha_vs = 1e-4; beta_vs  = 0.5;

% 4.1 Standard LMS
[y_lms, e_lms, w_hist_lms] = lms_adaptive_filter(x, d, L, mu_lms);
s_hat_lms = e_lms;

% 4.2 NLMS
[y_nlms, e_nlms, w_hist_nlms] = nlms_filter(x, d, L, mu_nlms, 1e-6);
s_hat_nlms = e_nlms;

% 4.3 Leaky LMS
```

```
[y_leaky, e_leaky, w_hist_leaky] = leaky_lms_filter(x, d, L, mu_leaky, gamma_leaky);
s_hat_leaky = e_leaky;

% 4.4 Block LMS
[y_block, e_block, w_hist_block] = block_lms_filter(x, d, L, mu_block, B_block);
s_hat_block = e_block;

% 4.5 Sign-Error LMS
[y_se, e_se, w_hist_se] = sign_error_lms_filter(x, d, L, mu_sign_err);
s_hat_se = e_se;

% 4.6 Sign-Data LMS
[y_sd, e_sd, w_hist_sd] = sign_data_lms_filter(x, d, L, mu_sign_data);
s_hat_sd = e_sd;

% 4.7 Sign-Sign LMS
[y_ss, e_ss, w_hist_ss] = sign_sign_lms_filter(x, d, L, mu_sign_sign);
s_hat_ss = e_ss;

% 4.8 Variable Step-Size LMS (VS-LMS)
[y_vs, e_vs, w_hist_vs] = vs_lms_filter(x, d, L, mu_vs_init, mu_vs_min, mu_vs_max, alpha_vs, beta_vs);
s_hat_vs = e_vs;

%% ----------------------------------------------------------------------
% 5. Performance Evaluation and Analytical Comparison Table
% ----------------------------------------------------------------------
win = 200;   % smoothing window for learning curves
steady_state_idx = round(0.7*N) : N; % Use last 30% of data for steady state

alg_names = {'LMS','NLMS','Leaky','Block','SignErr','SignData','SignSign','VS-LMS'};
e_all = {e_lms, e_nlms, e_leaky, e_block, e_se, e_sd, e_ss, e_vs};
s_hat_all = {s_hat_lms, s_hat_nlms, s_hat_leaky, s_hat_block, ...
             s_hat_se, s_hat_sd, s_hat_ss, s_hat_vs};
w_final_all = {w_hist_lms(:,end), w_hist_nlms(:,end), w_hist_leaky(:,end), w_hist_block(:,end), ...
               w_hist_se(:,end), w_hist_sd(:,end), w_hist_ss(:,end), w_hist_vs(:,end)};

inst_mse_all    = cell(size(e_all));
mse_smooth_all  = cell(size(e_all));
snr_clean_all   = zeros(size(e_all));
mse_ss_all      = zeros(size(e_all));
misadjustment_all = zeros(size(e_all));

```

```
% Optimal Wiener solution (for Misalignment comparison)
w_wiener = wiener_fir_solution(x, d, L);

% Calculate metrics for all algorithms
for i = 1:numel(e_all)
    e_i = e_all{i};
    inst_mse_all{i}   = e_i.^2;
    mse_smooth_all{i} = filter(ones(win,1)/win,1,inst_mse_all{i});

    % Output SNR (using error between estimate and clean speech)
    snr_clean_all(i) = snr(s, s_hat_all{i} - s);

    % Steady-state MSE (xi_inf)
    mse_ss = mean(inst_mse_all{i}(steady_state_idx));
    mse_ss_all(i) = mse_ss;

    % Misadjustment M = (xi_inf - xi_min) / xi_min
    misadjustment_all(i) = (mse_ss - xi_min_theory) / xi_min_theory;
end

% Input SNR (noisy vs clean)
snr_noisy = snr(s, d - s);

% --- Print Analytical Comparison Table to Command Window ---
fprintf('\n----------------------------------------------------------------------------------\n');
fprintf('ANALYTICAL PERFORMANCE COMPARISON TABLE (L = %d)\n', L);
fprintf('MMSE (xi_min) is: %.4f\n', xi_min_theory);
fprintf('----------------------------------------------------------------------------------\n');
fprintf('  Algorithm | SNR Out (dB) | MSE_ss | Misalignment (||w_err||) | Misadjustment (%%)\n');
fprintf('  ----------|--------------|--------|---------------------------|--------------------\n');

for i = 1:numel(e_all)
    % Misalignment: || w_final - w_wiener ||
    wev_norm = norm(w_final_all{i} - w_wiener);

    fprintf('  %8s | %12.2f | %6.4f | %25.6f | %18.2f\n', ...
            alg_names{i}, snr_clean_all(i), mse_ss_all(i), wev_norm, misadjustment_all(i)*100);
end
fprintf('  Input     | %12.2f |\n', snr_noisy);
fprintf('----------------------------------------------------------------------------------\n');

% Save results to a .mat file (for later analysis or plotting)
```

```
save('adaptive_filter_results.mat', 's','d','x','s_hat_all','alg_names', ...
     'w_final_all', 'w_wiener','snr_noisy','snr_clean_all');

%% ----------------------------------------------------------------------
% 6. Plots
% ----------------------------------------------------------------------

%% 6.1 Time-domain demo (first 1000 samples)
range = 1:1000;
figure('Position',[200 150 850 900]) % Large high-res figure

% (a) Clean Speech
subplot(5,1,1)
plot(t(range), s(range),'LineWidth',1.4)
ylim([min(s(range)) max(s(range))]*1.05)
title('(a) Clean Speech s[n]')
ylabel('Amplitude')
grid on

% (b) Noisy Speech (primary mic)
subplot(5,1,2)
plot(t(range), d(range),'LineWidth',1.4)
ylim([min(d(range)) max(d(range))]*1.05)
title('(b) Noisy Speech d[n]')
ylabel('Amplitude')
grid on

% (c) Reference Signal (used by LMS/NLMS)
subplot(5,1,3)
plot(t(range), x(range),'LineWidth',1.3)
ylim([min(x(range)) max(x(range))]*1.05)
title('(c) Reference Signal x[n]')
ylabel('Amplitude')
grid on

% (d) LMS Recovered
subplot(5,1,4)
plot(t(range), s_hat_lms(range),'b','LineWidth',1.4)
ylim([min(s_hat_lms(range)) max(s_hat_lms(range))]*1.05)
title('(d) LMS Recovered Speech')
ylabel('Amplitude')
grid on
```

```

% (e) NLMS Recovered
subplot(5,1,5)
plot(t(range), s_hat_nlms(range),'m','LineWidth',1.4)
ylim([min(s_hat_nlms(range)) max(s_hat_nlms(range))]*1.05)
title('(e) NLMS Recovered Speech')
xlabel('Time (s)')
ylabel('Amplitude')
grid on

%% 6.2 Learning curves (MSE)     all algorithms
figure;
hold on;
for i = 1:numel(alg_names)
    plot(10*log10(mse_smooth_all{i}), 'DisplayName', alg_names{i});
end
yline(10*log10(xi_min_theory), 'k--', 'DisplayName', 'Wiener MMSE');
grid on;
xlabel('Iteration n');
ylabel('Smoothed MSE (dB)');
title(sprintf('Learning Curves Comparison, L = %d', L));
legend('Location','best');

%% 6.3 Evolution of filter coefficients     LMS vs NLMS
figure;
subplot(2,1,1);
plot(w_hist_lms.'); grid on;
xlabel('Iteration n'); ylabel('Coefficient value');
title('Evolution of LMS Filter Weights');

subplot(2,1,2);
plot(w_hist_nlms.'); grid on;
xlabel('Iteration n'); ylabel('Coefficient value');
title('Evolution of NLMS Filter Weights');

%% 6.4 PSD comparison     clean vs noisy vs LMS vs NLMS
nfft = 2048;

% Use only samples after convergence for PSD (e.g., last 70% of data)
idx_psd     = round(0.3*N) : N;
s_psd       = s(idx_psd);
d_psd       = d(idx_psd);
```

```
s_lms_psd   = s_hat_lms(idx_psd);
s_nlms_psd  = s_hat_nlms(idx_psd);

[Sp,  f]   = pwelch(s_psd,      hamming(512),256,nfft,Fs);
[Dp,  ~]   = pwelch(d_psd,      hamming(512),256,nfft,Fs);
[Slms,~]   = pwelch(s_lms_psd,  hamming(512),256,nfft,Fs);
[Snlm,~]   = pwelch(s_nlms_psd, hamming(512),256,nfft,Fs);

figure;
plot(f,10*log10(Sp),'LineWidth',1.5); hold on;
plot(f,10*log10(Dp),'LineWidth',1.5);
plot(f,10*log10(Slms),'LineWidth',1.5);
plot(f,10*log10(Snlm),'LineWidth',1.5);
grid on;
xlabel('Frequency (Hz)');
ylabel('Power / Frequency (dB/Hz)');
title('PSD of Clean / Noisy / LMS / NLMS Recovered Speech');
legend('Clean speech','Noisy speech','LMS','NLMS','Location','best');
xlim([0 Fs/2]);

%% 6.4b Bilateral spectrum     clean vs noisy vs LMS vs NLMS
nfft_bi = 4096;

% Compute FFTs and shift zero frequency to center
S_clean = fftshift(fft(s,         nfft_bi));
S_noisy = fftshift(fft(d,         nfft_bi));
S_lms   = fftshift(fft(s_hat_lms, nfft_bi));
S_nlms  = fftshift(fft(s_hat_nlms,nfft_bi));

% Frequency axis for bilateral spectrum
f_bi = (-nfft_bi/2:nfft_bi/2-1) * (Fs/nfft_bi);

% Convert to dB magnitude
S_clean_db = 20*log10(abs(S_clean) + eps);
S_noisy_db = 20*log10(abs(S_noisy) + eps);
S_lms_db   = 20*log10(abs(S_lms)   + eps);
S_nlms_db  = 20*log10(abs(S_nlms)  + eps);

figure;
plot(f_bi, S_clean_db,'LineWidth',1.5); hold on;
plot(f_bi, S_noisy_db,'LineWidth',1.5);
plot(f_bi, S_lms_db,  'LineWidth',1.5);
plot(f_bi, S_nlms_db, 'LineWidth',1.5);
grid on;
xlabel('Frequency (Hz)');
ylabel('Magnitude (dB)');
title('Bilateral Spectrum of Clean / Noisy / LMS / NLMS Speech');
legend('Clean speech','Noisy speech','LMS','NLMS','Location','best');
xlim([-Fs/2 Fs/2]);

%% 6.5 Bar plot of SNR improvement
figure;
vals = [snr_noisy; snr_clean_all(:)];
bar(vals);
xticks(1:length(vals));
xticklabels([{'Input'}, alg_names]);  % cell array of labels
ylabel('SNR (dB)');
title('SNR of Input and All Adaptive Filters');
grid on;

%% -------------------------------------------------------------------------
% 7. (Optional) Listen to signals
% -------------------------------------------------------------------------
% soundsc(s, Fs);          % original clean speech
% pause(length(s)/Fs + 1);
% soundsc(d, Fs);          % noisy speech
% pause(length(s)/Fs + 1);
% soundsc(s_hat_lms, Fs);  % after LMS
% pause(length(s)/Fs + 1);
% soundsc(s_hat_nlms, Fs); % after NLMS

%% -------------------------------------------------------------------------
% 8. Two-tap LMS and NLMS weight-trajectory demos ("bowl" figures)
% -------------------------------------------------------------------------
N2    = 500;
L2    = 2;
x2    = randn(N2,1);
h_true = [0.8; -0.5];
d2 = filter(h_true,1,x2);

% --- 2-tap LMS ---
mu2_lms = 0.02;
[~, e2_lms, w_hist2_lms] = lms_adaptive_filter(x2, d2, L2, mu2_lms);
w0_lms = w_hist2_lms(1,:);
```

```
w1_lms = w_hist2_lms(2,:);

% --- 2-tap NLMS ---
mu2_nlms = 0.8;
[~, e2_nlms, w_hist2_nlms] = nlms_filter(x2, d2, L2, mu2_nlms, 1e-6);
w0_nlms = w_hist2_nlms(1,:);
w1_nlms = w_hist2_nlms(2,:);

% Cost-surface bowl (J as a function of w0, w1)
[w0g,w1g] = meshgrid(linspace(-1.5,1.5,60), linspace(-1.5,1.5,60));
J = (w0g - h_true(1)).^2 + (w1g - h_true(2)).^2;

figure;
contour(w0g,w1g,J,20); hold on; grid on;
plot(w0_lms,w1_lms,'r.-','LineWidth',1.2);
plot(w0_nlms,w1_nlms,'b.-','LineWidth',1.2);
plot(h_true(1),h_true(2),'kx','MarkerSize',10,'LineWidth',2);
xlabel('w(0)');
ylabel('w(1)');
title('2-tap LMS and NLMS Weight Trajectories in Weight Space');
legend('Error contours','LMS trajectory','NLMS trajectory','Wiener solution');

figure;
subplot(2,1,1);
plot(1:N2,w0_lms,1:N2,w1_lms,'LineWidth',1.5); hold on;
yline(h_true(1),'--'); yline(h_true(2),'--');
grid on;
xlabel('Iteration n'); ylabel('Coefficient value');
title('2-tap LMS Coefficient Evolution');
legend('w_0','w_1','Location','best');

subplot(2,1,2);
plot(1:N2,w0_nlms,1:N2,w1_nlms,'LineWidth',1.5); hold on;
yline(h_true(1),'--'); yline(h_true(2),'--');
grid on;
xlabel('Iteration n'); ylabel('Coefficient value');
title('2-tap NLMS Coefficient Evolution');
legend('w_0','w_1','Location','best');

%% ---------------------------------------------------------------------
% 8b. Explicit Steepest-Descent Demo (Batch R,p) on the 2-tap Example
% ---------------------------------------------------------------------
% Cost function: J(w) = E{ (d2[n] - w^T u[n])^2 }, u[n] = [x2[n]; x2[n-1]]
% Steepest descent:
%   w_{k+1} = w_k - 2 * mu_sd * (R_hat * w_k - p_hat)

% Build regression matrix U and desired vector d_seg
N_sd  = N2 - L2 + 1;                   % number of regression vectors
U     = zeros(N_sd, L2);
d_seg = d2(L2:end);                    % align with u[n]

for n = 1:N_sd
    U(n, :) = x2(n+L2-1:-1:n).';       % u[n]^T
end

% Sample estimates of R and p
R_hat = (U.' * U) / N_sd;              % L2 x L2 correlation matrix
p_hat = (U.' * d_seg) / N_sd;          % L2 x 1 cross-correlation vector

% Choose stable step size: 0 < mu_sd < 1 / lambda_max(R_hat)
lambda_max = max(eig(R_hat));
mu_sd      = 1 / (2 * lambda_max);     % conservative choice

% Steepest-descent iterations
K_sd       = 50;                       % number of SD iterations
w_sd       = zeros(L2, 1);             % initial guess
w_hist2_sd = zeros(L2, K_sd);          % weight trajectory
J_sd       = zeros(K_sd, 1);           % cost vs iteration

for k = 1:K_sd
    gradJ = 2 * (R_hat * w_sd - p_hat);   % exact gradient
    w_sd  = w_sd - mu_sd * gradJ;         % SD update

    w_hist2_sd(:, k) = w_sd;

    % Approximate MSE cost at iteration k
    err_k   = d_seg - U * w_sd;
    J_sd(k) = mean(err_k.^2);
end

% SD trajectory components for plotting
w0_sd = w_hist2_sd(1, :);
w1_sd = w_hist2_sd(2, :);

```

```
% Plot SD trajectory together with LMS and NLMS on the bowl
figure;
contour(w0g, w1g, J, 20); hold on; grid on;
plot(w0_lms,  w1_lms,  'r.-', 'LineWidth', 1.2);
plot(w0_nlms, w1_nlms, 'b.-', 'LineWidth', 1.2);
plot(w0_sd,   w1_sd,   'g.-', 'LineWidth', 1.5);
plot(h_true(1), h_true(2), 'kx', 'MarkerSize', 10, 'LineWidth', 2);
xlabel('w(0)');
ylabel('w(1)');
title('2-tap Steepest Descent vs LMS and NLMS on Cost Surface');
legend('Error contours', ...
       'LMS trajectory', ...
       'NLMS trajectory', ...
       'Steepest-descent trajectory', ...
       'Wiener solution', ...
       'Location', 'best');

% SD learning curve
figure;
plot(1:K_sd, 10*log10(J_sd), 'g-o', 'LineWidth', 1.5, 'MarkerSize', 4);
grid on;
xlabel('Iteration k');
ylabel('MSE J(w_k) (dB)');
title('Steepest-Descent Learning Curve (Batch R,p) for 2-tap Example');

%% ---------------------------------------------------------------------
% 9. LMS Step-Size Trade-off (Convergence vs Misadjustment)
% ----------------------------------------------------------------------
N_lc   = length(s);
R_lc   = 50;             % Number of independent runs for smoother average
L_comp = L;              % Use the filter length defined in Section 4

% Define mu values for comparison
mu_slow   = 0.001;
mu_medium = mu_lms;      % standard mu from Section 4
mu_fast   = 0.05;

% Pre-allocate storage for ensemble-averaged squared error
J_slow   = zeros(N_lc,1);
J_medium = zeros(N_lc,1);
J_fast   = zeros(N_lc,1);

```

```
for r = 1:R_lc
    [~, e_slow,   ~] = lms_adaptive_filter(x, d, L_comp, mu_slow);
    [~, e_medium, ~] = lms_adaptive_filter(x, d, L_comp, mu_medium);
    [~, e_fast,   ~] = lms_adaptive_filter(x, d, L_comp, mu_fast);

    J_slow   = J_slow   + e_slow.^2;
    J_medium = J_medium + e_medium.^2;
    J_fast   = J_fast   + e_fast.^2;
end

J_slow   = J_slow   / R_lc;
J_medium = J_medium / R_lc;
J_fast   = J_fast   / R_lc;

figure;
plot(10*log10(J_slow),   'r-',  'LineWidth', 1.5, 'DisplayName', sprintf('\\mu = %.4f (Slow)', mu_slow));
     hold on;
plot(10*log10(J_medium), 'b--', 'LineWidth', 1.5, 'DisplayName', sprintf('\\mu = %.2f (Medium)', mu_medium))
     ;
plot(10*log10(J_fast),   'k:',  'LineWidth', 1.5, 'DisplayName', sprintf('\\mu = %.2f (Fast)', mu_fast));
yline(10*log10(xi_min_theory), 'g-.', 'LineWidth', 1.5, 'DisplayName', 'Wiener MMSE');
grid on;
xlabel('Iteration n');
ylabel('Ensemble Averaged MSE (dB)');
title(sprintf('LMS Convergence vs Step Size (L = %d)', L_comp));
legend('Location', 'best');
xlim([0 N_lc]);

%% ------------------------------------------------------------------------
% 10. Nonstationary-noise cancellation demo
% ------------------------------------------------------------------------
N_ns = 1000;
n_ns = (0:N_ns-1)';

% (a) Desired sinusoid to be estimated
f_des = 0.02*pi;                       % rad/sample (slow sinusoid)
s_ns  = 2 * sin(f_des*n_ns);           % desired clean sinusoid

% Time-varying (nonstationary) noise: variance slowly increases with n
sigma_ns = linspace(0.5, 2.5, N_ns)';  % time-varying std
v_ns     = sigma_ns .* randn(N_ns,1);  % nonstationary noise

```

```
% (b) Noise-corrupted sinusoid at primary sensor
d_ns = s_ns + v_ns;

% (c) Reference noise at secondary sensor (correlated with v_ns)
b_ref_ns = [1 0.5 -0.3];               % some secondary-path filter
x_ns = filter(b_ref_ns, 1, v_ns);      % reference signal

% NLMS adaptive noise canceller (12th-order)
L_ns   = 12;
beta_ns = 0.25;                        % normalized step size beta
eps_ns  = 1e-6;
[~, e_ns, ~] = nlms_filter(x_ns, d_ns, L_ns, beta_ns, eps_ns);
y_ns = e_ns;                           % NLMS output (estimate of sinusoid)

figure;
subplot(4,1,1);
plot(n_ns, s_ns); grid on;
ylabel('Amp');
title('(a) Desired signal s_{ns}[n]');

subplot(4,1,2);
plot(n_ns, d_ns); grid on;
ylabel('Amp');
title('(b) Noise-corrupted sinusoid d_{ns}[n]');

subplot(4,1,3);
plot(n_ns, x_ns); grid on;
ylabel('Amp');
title('(c) Reference signal x_{ns}[n]');

subplot(4,1,4);
plot(n_ns, y_ns); grid on;
xlabel('Sample index n');
ylabel('Amp');
title('(d) Output of 12th-order NLMS noise canceller (\beta = 0.25)');
%% ----------------------------------------------------------------------
% 10b. Nonstationary-noise cancellation demo (time-varying variance, LMS)
% ----------------------------------------------------------------------

% Use the same desired signal s_ns, nonstationary noise v_ns,
% primary d_ns = s_ns + v_ns, and reference x_ns from Section 10.

```

```
mu_ns_lms = 0.005;                 % LMS step size (small for stability)

% LMS adaptive noise canceller on the same nonstationary data
[~, e_ns_lms, ~] = lms_adaptive_filter(x_ns, d_ns, L_ns, mu_ns_lms);
y_ns_lms = e_ns_lms;               % LMS output (estimate of sinusoid)

% --- Time-domain comparison of outputs ---
figure;
plot(n_ns, s_ns, 'k--', 'LineWidth', 1.5); hold on;
plot(n_ns, y_ns_lms, 'r',  'LineWidth', 1.2);
plot(n_ns, y_ns,     'b',  'LineWidth', 1.2);   % y_ns from NLMS section
grid on;
xlabel('Sample index n');
ylabel('Amplitude');
title('Nonstationary Noise Cancellation: LMS vs NLMS (Time Domain)');
legend('Desired s_{ns}[n]', ...
       sprintf('LMS (\\mu = %.4f)', mu_ns_lms), ...
       sprintf('NLMS (\\beta = %.2f)', beta_ns), ...
       'Location','best');

% --- Learning-curve comparison (MSE vs n) ---
win_ns = 40;  % short moving-average window

% MSE between estimate and true sinusoid
mse_lms_ns  = movmean((s_ns - y_ns_lms).^2, win_ns);
mse_nlms_ns = movmean((s_ns - y_ns    ).^2, win_ns);

% Output SNRs
snr_lms_ns  = snr(s_ns, y_ns_lms - s_ns);
snr_nlms_ns = snr(s_ns, y_ns     - s_ns);

figure;
plot(10*log10(mse_lms_ns),  'r', 'LineWidth', 1.5, ...
    'DisplayName', sprintf('LMS (\\mu = %.4f, SNR = %.2f dB)', ...
                           mu_ns_lms, snr_lms_ns)); hold on;
plot(10*log10(mse_nlms_ns), 'b', 'LineWidth', 1.5, ...
    'DisplayName', sprintf('NLMS (\\beta = %.2f, SNR = %.2f dB)', ...
                           beta_ns, snr_nlms_ns));
grid on;
xlabel('Sample index n');
ylabel('MSE (dB)');
title('Nonstationary Noise Cancellation (Time-Varying Variance): LMS vs NLMS');
```

```
legend('Location','best');

%% ---------------------------------------------------------------------
% 11. Noise cancellation WITHOUT a reference signal
% ---------------------------------------------------------------------
N_sc  = 1000;
n_sc  = (0:N_sc-1)';

% Underlying sinusoid (narrowband component)
omega0 = 0.06*pi;                  % rad/sample
s_sc   = 2 * sin(omega0 * n_sc);   % clean sinusoid (hidden)

% Additive wideband noise
v_sc   = randn(N_sc,1);            % zero-mean white noise
x_sc   = s_sc + v_sc;              % noisy process x(n)  (no reference mic)

% NLMS noise canceller using delayed version of x_sc as "reference"
L_sc    = 12;                      % filter order
beta_sc = 0.25;                    % normalized step size
n0      = 25;                      % delay
eps_sc  = 1e-6;

% Delayed reference signal
ref_sc = [zeros(n0,1); x_sc(1:end-n0)];
w_sc = zeros(L_sc,1);
y_sc = zeros(N_sc,1);              % NLMS output (noise estimate)
e_sc = zeros(N_sc,1);              % error = x_sc - y_sc (enhanced signal)

for n = L_sc:N_sc
    u = ref_sc(n:-1:n-L_sc+1);     % input vector from delayed x_sc
    y_sc(n) = w_sc.' * u;          % estimated noise component
    e_sc(n) = x_sc(n) - y_sc(n);   % output of noise canceller
    mu_n = beta_sc / (eps_sc + (u.'*u));
    w_sc = w_sc + mu_n * e_sc(n) * u;
end

figure;
subplot(2,1,1);
plot(n_sc, x_sc, 'k'); grid on;
ylabel('Amplitude');
title('(a) Noisy process x_{sc}[n]');
ylim([-3 3]);
```

```

subplot(2,1,2);
plot(n_sc, e_sc, 'k'); grid on;
xlabel('Sample index n');
ylabel('Amplitude');
title('(b) Output of 12th-order NLMS canceller (\beta = 0.25, n_0 = 25)');
ylim([-3 3]);

%% ----------------------------------------------------------------------
% 12. Gradient Adaptive Lattice (GAL) demo for an AR(1) process
% ----------------------------------------------------------------------
N_gal  = 3000;          % samples for clearer convergence
a_true = 0.9;           % true AR(1) coefficient
v_gal  = randn(N_gal,1);
x_gal  = filter(1, [1 -a_true], v_gal);   % AR(1) process

Gamma1      = 0;                       % initial reflection coefficient
mu_gal      = 1e-6;                    % small step for stability
Gamma_hist  = zeros(N_gal,1);
J_gal       = zeros(N_gal,1);          % instantaneous Burg cost

for n = 1:N_gal
    if n==1
        e0p = x_gal(n);   % forward error at stage 0
        e0m = 0;          % no past sample yet
    else
        e0p = x_gal(n);        % e_0^+(n)
        e0m = x_gal(n-1);      % e_0^-(n-1)
    end

    % Stage-1 forward/backward errors from lattice recursions
    e1p = e0p + Gamma1 * e0m;   % e_1^+(n)
    e1m = e0m + Gamma1 * e0p;   % e_1^-(n)

    % Instantaneous Burg cost  _1 ^B(n)     |e1p|^2 + |e1m|^2
    J_gal(n) = e1p^2 + e1m^2;

    % Gradient of Burg cost wrt  _1  (real case)
    gradGamma = -2 * (e0p*e1m + e0m*e1p);

    % Gradient-descent update for  _1
    Gamma1 = Gamma1 - mu_gal * gradGamma;

    % Clipping to maintain stability: reflection coefficients     (-1,1)
    Gamma1 = max(-0.999, min(0.999, Gamma1));

    Gamma_hist(n) = Gamma1;
end

% Plot convergence of Burg cost and reflection coefficient
n_axis = 1:N_gal;
figure(’Position’,[100 100 700 800]);

subplot(2,1,1);
plot(n_axis, 10*log10(J_gal + eps), ’b’, ’LineWidth’, 1.5);
grid on;
set(gca, ’FontSize’, 12, ’FontName’, ’Times New Roman’);
xlabel(’Iteration n’, ’FontSize’, 14, ’FontName’, ’Times New Roman’);
ylabel(’Instantaneous Burg Cost \xi_1^B(n) (dB)’, ’FontSize’, 14, ’FontName’, ’Times New Roman’);
title(’GAL Lattice: Burg Cost for AR(1) Process’, ’FontSize’, 14, ’FontName’, ’Times New Roman’);

subplot(2,1,2);
plot(n_axis, Gamma_hist, ’b’, ’LineWidth’, 1.5); hold on;
yline(-a_true, ’k--’, ’LineWidth’, 1.5);
grid on;
set(gca, ’FontSize’, 12, ’FontName’, ’Times New Roman’);
xlabel(’Iteration n’, ’FontSize’, 14, ’FontName’, ’Times New Roman’);
ylabel(’Reflection Coefficient \Gamma_1(n)’, ’FontSize’, 14, ’FontName’, ’Times New Roman’);
title(’GAL Lattice: Adaptation of Reflection Coefficient \Gamma_1’, ’FontSize’, 14, ’FontName’, ’Times New Roman’);
legend(’{\Gamma_1(n)}’,’Theoretical Value ({-a}_{true})’,’Location’,’southeast’, ’FontSize’, 12, ’FontName’, ’Times New Roman’);

%% ---------------------------------------------------------------------
% 13. Robustness Test: LMS vs Sign-Error LMS under Impulsive Noise
% ----------------------------------------------------------------------
N_imp  = length(s);
L_imp  = L;
mu_imp = 0.01;  % common mu for fair comparison

% --- 13.1 Generate Impulsive Noise (Non-Gaussian) ---
impulse_prob = 0.005;
impulse_amp  = 8;
spikes = impulse_amp * (rand(N_imp, 1) < impulse_prob) .* randn(N_imp, 1);
spikes = spikes / std(spikes);

% Primary microphone signal with impulsive noise (baseline + spikes)
d_impulsive = s + noise_primary + spikes;
d_impulsive = d_impulsive - mean(d_impulsive);
d_impulsive = d_impulsive / std(d_impulsive);
x_imp = x; % Use the original reference signal

% --- 13.2 Run LMS and Sign-Error LMS ---
[~, e_lms_imp, ~] = lms_adaptive_filter(x_imp, d_impulsive, L_imp, mu_imp);
[~, e_se_imp,  ~] = sign_error_lms_filter(x_imp, d_impulsive, L_imp, mu_imp);

% --- 13.3 Performance Calculation and Plot ---
win_imp     = 200;
mse_lms_imp = filter(ones(win_imp,1)/win_imp,1,e_lms_imp.^2);
mse_se_imp  = filter(ones(win_imp,1)/win_imp,1,e_se_imp.^2);
snr_lms_imp = snr(s, e_lms_imp - s);
snr_se_imp  = snr(s, e_se_imp - s);

figure;
plot(10*log10(mse_lms_imp), 'r', 'LineWidth', 1.5, ...
    'DisplayName', sprintf('LMS (SNR: %.2f dB)', snr_lms_imp)); hold on;
plot(10*log10(mse_se_imp), 'b', 'LineWidth', 1.5, ...
    'DisplayName', sprintf('Sign-Error LMS (SNR: %.2f dB)', snr_se_imp));
grid on;
xlabel('Iteration n');
ylabel('MSE (dB)');
title('LMS vs. Sign-Error LMS under Impulsive Noise');
legend('Location', 'best');
xlim([0 N_imp]);

%% ----------------------------------------------------------------------
% 14. Effect of AR(1) Coefficient on LMS Convergence
% -----------------------------------------------------------------------
ar_coeffs = [0.2 0.5 0.9];   % weak, medium, strong correlation
mu_lms_ar = 0.01;            % fixed step size
L_ar      = L;
win_ar    = 200;

figure;
hold on;
leg_str = cell(numel(ar_coeffs),1);
```

```

for k = 1:numel(ar_coeffs)
    a1 = ar_coeffs(k);

    % Regenerate AR(1) noise with new coefficient
    ar_a_k   = [1 -a1];
    ar_exc_k = randn(N,1);
    ar_nk    = filter(1, ar_a_k, ar_exc_k);
    ar_nk    = ar_nk / std(ar_nk);

    % Use same WGN + baseline sinusoid + new AR(1)
    noise_k = sigma_wgn*wgn + sigma_sine*sin_noise_base + sigma_ar*ar_nk;

    % Primary mic: speech + this new noise
    d_k = s + noise_k;
    d_k = d_k - mean(d_k);
    d_k = d_k / std(d_k);

    % Reference channel for this noise
    x_k = filter(b_ref, 1, noise_k);
    x_k = x_k / std(x_k);

    % Run LMS with same mu for all a1
    [~, e_lms_k, ~] = lms_adaptive_filter(x_k, d_k, L_ar, mu_lms_ar);

    % Smooth MSE
    mse_k = filter(ones(win_ar,1)/win_ar, 1, e_lms_k.^2);
    plot(10*log10(mse_k),'LineWidth',1.2);

    leg_str{k} = sprintf('a = %.2f', a1);
end

yline(10*log10(xi_min_theory),'k--','Wiener MMSE');
grid on;
xlabel('Iteration n');
ylabel('Smoothed MSE (dB)');
title(sprintf('Effect of AR(1) Coefficient on LMS Convergence (\\mu = %.3f)', mu_lms_ar));
legend(leg_str,'Location','best');

%% ---------------------------------------------------------------------
% 15. Effect of Sinusoidal Interference Frequency on LMS Convergence
% ---------------------------------------------------------------------
```

```
freqs  = [200 400 800];  % Hz
mu_sin = 0.01;
L_sin  = 32;

figure;
hold on;

for k = 1:length(freqs)
    f0_k = freqs(k);
    sin_noise_k = sin(2*pi*f0_k*t);
    sin_noise_k = sin_noise_k / std(sin_noise_k);

    noise_k = sigma_wgn*wgn + sigma_ar*ar_noise + sigma_sine*sin_noise_k;

    d_k = s + noise_k;
    d_k = d_k - mean(d_k);
    d_k = d_k / std(d_k);

    x_k = filter(b_ref,1,noise_k);
    x_k = x_k/std(x_k);

    [~, e_k, ~] = lms_adaptive_filter(x_k, d_k, L_sin, mu_sin);
    mse_k = movmean(e_k.^2, 200);

    plot(10*log10(mse_k), ’LineWidth’, 1.2);
end

xlabel(’Iteration n’);
ylabel(’Smoothed MSE (dB)’);
title(’Effect of Sinusoidal Frequency on LMS Convergence’);
legend(’200 Hz’,’400 Hz’,’800 Hz’,’Location’,’best’);
grid on;

%% ------------------------------------------------------------------------
% 16. Effect of WGN Power (  ) on LMS Convergence
% ------------------------------------------------------------------------
sigmas = [0.1 0.5 1.0];

figure; hold on;
for k = 1:length(sigmas)
    wgn_k = sigmas(k) * randn(N,1);

    % Keep AR noise and baseline 400 Hz sinusoid fixed
    noise_k = wgn_k + sigma_ar*ar_noise + sigma_sine*sin_noise_base;

    d_k = s + noise_k;
    d_k = d_k - mean(d_k);
    d_k = d_k / std(d_k);

    x_k = filter(b_ref,1,noise_k);
    x_k = x_k/std(x_k);

    [~, e_k, ~] = lms_adaptive_filter(x_k, d_k, L, mu_lms);
    mse_k = movmean(e_k.^2,200);

    plot(10*log10(mse_k),'LineWidth',1.2);
end

legend('\sigma=0.1','\sigma=0.5','\sigma=1.0','Location','best');
title('Effect of WGN Power on LMS Convergence');
xlabel('Iteration n'); ylabel('Smoothed MSE (dB)');
grid on;

%% ----------------------------------------------------------------------
% 17. Effect of Impulsive Noise Probability on Sign-Error LMS
% ----------------------------------------------------------------------
probs = [0.001 0.005 0.01];

figure; hold on;
for k = 1:length(probs)
    spikes_k = impulse_amp * (rand(N,1) < probs(k)) .* randn(N,1);
    spikes_k = spikes_k / std(spikes_k);

    d_k = s + noise_primary + spikes_k;
    d_k = (d_k - mean(d_k)) / std(d_k);

    [~, e_k, ~] = sign_error_lms_filter(x, d_k, L, mu_lms);
    mse_k = movmean(e_k.^2,200);

    plot(10*log10(mse_k),'LineWidth',1.2);
end

legend('p=0.001','p=0.005','p=0.01','Location','best');
title('Effect of Impulsive Noise Probability on Sign-Error LMS');
```

```matlab
xlabel('Iteration n'); ylabel('MSE (dB)');
grid on;

%% ----------------------------------------------------------------------
% 18. Eigenvalue & Condition-Number Analysis of R_xx (Baseline Reference)
% ----------------------------------------------------------------------
% This section estimates the input correlation matrix R_xx for the baseline
% reference signal x (used in LMS/NLMS) and analyzes its eigenvalues and
% condition number. This links directly to the LMS stability bound
%   0 < mu < 2 / lambda_max(R_xx)
% and explains why convergence speed depends on input correlation.

L_eig = L;                  % use same filter length as main experiment
N_eig = length(x);
N_reg = N_eig - L_eig + 1;

% Build regression matrix X_reg whose rows are x_n^T = [x(n) ... x(n-L+1)]
X_reg = zeros(N_reg, L_eig);
for n = L_eig:N_eig
    X_reg(n-L_eig+1, :) = x(n:-1:n-L_eig+1).';
end

% Sample correlation matrix R_xx     E{x_n x_n^T}
R_xx_hat = (X_reg.' * X_reg) / N_reg;

% Eigenvalue decomposition of R_xx
[Vecs, D] = eig(R_xx_hat);
lambda = diag(D);

lambda_max = max(lambda);
lambda_min = min(lambda);
cond_Rxx   = lambda_max / lambda_min;

fprintf('\n============================================================\n');
fprintf('Eigenvalue / Condition-Number Analysis of R_xx (L = %d)\n', L_eig);
fprintf('------------------------------------------------------------\n');
fprintf('  lambda_max(R_xx) = %.4e\n', lambda_max);
fprintf('  lambda_min(R_xx) = %.4e\n', lambda_min);
fprintf('  cond(R_xx)       = %.4f\n', cond_Rxx);
fprintf('  LMS stability bound:  0 < mu < 2 / lambda_max     %.4e\n', 2/lambda_max);
fprintf('  Your chosen mu_lms   = %.4e\n', mu_lms);
fprintf('============================================================\n\n');
```

```

% Plot eigenvalue spectrum (log scale to show spread clearly)
figure;
stem(1:L_eig, lambda, 'filled','LineWidth',1.4);
grid on;
xlabel('Eigenvalue index k');
ylabel('\lambda_k(R_{xx})');
title(sprintf('Eigenvalue Spectrum of R_{xx} (cond = %.2f)', cond_Rxx));

% Also show in dB (relative)
figure;
stem(1:L_eig, 10*log10(lambda / max(lambda)), 'filled','LineWidth',1.4);
grid on;
xlabel('Eigenvalue index k');
ylabel('Eigenvalue magnitude (dB, normalized)');
title('Relative Eigenvalue Spectrum of R_{xx}');
%% ---------------------------------------------------------------------
% 19. Tracking Ability: Time-Varying AR(1) Coefficient (LMS vs VS-LMS)
% ---------------------------------------------------------------------
% Here the AR(1) coefficient slowly changes with time:
%   a[n] = 0.4 + 0.4 * sin(2*pi*n / N)
% so the correlation structure of the noise drifts. We test how well
% fixed-step LMS and VS-LMS track this time-varying environment.

N_tv  = N;                         % use same length as baseline signal
n_tv  = (0:N_tv-1)';

% Slowly time-varying AR(1) coefficient in [0.0, 0.8]
a_tv = 0.4 + 0.4 * sin(2*pi*n_tv / N_tv);

% Generate time-varying AR(1) noise: v[n] = a[n] v[n-1] + e[n]
v_tv = zeros(N_tv,1);
e_tv = randn(N_tv,1);
for n = 2:N_tv
    v_tv(n) = a_tv(n) * v_tv(n-1) + e_tv(n);
end
v_tv = v_tv / std(v_tv);

% Build noise mixture using same WGN + baseline 400 Hz sinusoid + AR_tv
noise_tv = sigma_wgn * wgn(1:N_tv) + ...
           sigma_sine * sin_noise_base(1:N_tv) + ...
           sigma_ar   * v_tv;
```

```

% Primary mic: speech + time-varying noise
d_tv = s(1:N_tv) + noise_tv;
d_tv = d_tv - mean(d_tv);
d_tv = d_tv / std(d_tv);

% Reference channel for this noise
x_tv = filter(b_ref, 1, noise_tv);
x_tv = x_tv / std(x_tv);

% Run LMS (fixed step) and VS-LMS on this nonstationary scenario
mu_lms_tv = mu_lms;   % reuse baseline LMS step size

[~, e_lms_tv, w_hist_lms_tv] = lms_adaptive_filter(x_tv, d_tv, L, mu_lms_tv);
[~, e_vs_tv,  w_hist_vs_tv]  = vs_lms_filter(x_tv, d_tv, L, ...
                                             mu_vs_init, mu_vs_min, ...
                                             mu_vs_max, alpha_vs, beta_vs);

% Smooth MSE for visualization
win_tv       = 200;
mse_lms_tv   = movmean(e_lms_tv.^2, win_tv);
mse_vs_tv    = movmean(e_vs_tv.^2,  win_tv);

figure;
subplot(2,1,1);
plot(10*log10(mse_lms_tv),'r','LineWidth',1.4); hold on;
plot(10*log10(mse_vs_tv),'b','LineWidth',1.4);
grid on;
xlabel('Sample index n');
ylabel('Smoothed MSE (dB)');
title('Tracking under Time-Varying AR(1) Coefficient');
legend('LMS (fixed \mu)','VS-LMS (variable \mu)','Location','best');

% Compare first coefficient against time-varying AR behavior (qualitative)
subplot(2,1,2);
plot(n_tv, a_tv, 'k--','LineWidth',1.2); hold on;
plot(n_tv, w_hist_lms_tv(1, :).', 'r','LineWidth',1.2);
plot(n_tv, w_hist_vs_tv(1,  :).', 'b','LineWidth',1.2);
grid on;
xlabel('Sample index n');
ylabel('Coefficient value');
title('First Filter Tap vs Time-Varying AR(1) Coefficient');
```

```
legend('a[n] (time-varying)','LMS: w_1[n]','VS-LMS: w_1[n]','Location','best');
%% ----------------------------------------------------------------------
% 20. Sensitivity to AR(1) Coefficient Across Algorithms (SNR Comparison)
% ----------------------------------------------------------------------
% We now study how the AR(1) coefficient affects the performance of several
% algorithms (LMS, NLMS, VS-LMS) using output SNR as a scalar performance
% metric. This extends the LMS-only plot in Section 14.

ar_coeffs = [0.2 0.5 0.9];          % weak, medium, strong correlation
num_a     = numel(ar_coeffs);

alg_subnames = {'LMS','NLMS','VS-LMS'};
num_alg      = numel(alg_subnames);

SNR_out_mat  = zeros(num_a, num_alg);   % rows: a, cols: alg

% Common parameters for all runs
L_sens      = L;
mu_lms_sens = mu_lms;
mu_nlms_sens = mu_nlms;

for ia = 1:num_a
    a1 = ar_coeffs(ia);

    % --- Generate AR(1) noise with coefficient a1 ---
    ar_a_k   = [1 -a1];
    ar_exc_k = randn(N,1);
    ar_nk    = filter(1, ar_a_k, ar_exc_k);
    ar_nk    = ar_nk / std(ar_nk);

    % Use same WGN + baseline 400 Hz sinusoid + new AR(1)
    noise_k = sigma_wgn*wgn + sigma_sine*sin_noise_base + sigma_ar*ar_nk;

    % Primary mic: speech + this new noise
    d_k = s + noise_k;
    d_k = d_k - mean(d_k);
    d_k = d_k / std(d_k);

    % Reference channel
    x_k = filter(b_ref, 1, noise_k);
    x_k = x_k / std(x_k);

    % --- Algorithm 1: LMS ---
    [~, e_lms_k, ~] = lms_adaptive_filter(x_k, d_k, L_sens, mu_lms_sens);
    s_hat_lms_k = e_lms_k;
    SNR_out_mat(ia,1) = snr(s, s_hat_lms_k - s);

    % --- Algorithm 2: NLMS ---
    [~, e_nlms_k, ~] = nlms_filter(x_k, d_k, L_sens, mu_nlms_sens, 1e-6);
    s_hat_nlms_k = e_nlms_k;
    SNR_out_mat(ia,2) = snr(s, s_hat_nlms_k - s);

    % --- Algorithm 3: VS-LMS ---
    [~, e_vs_k, ~] = vs_lms_filter(x_k, d_k, L_sens, ...
                                   mu_vs_init, mu_vs_min, ...
                                   mu_vs_max, alpha_vs, beta_vs);
    s_hat_vs_k = e_vs_k;
    SNR_out_mat(ia,3) = snr(s, s_hat_vs_k - s);
end

% Print SNR table to command window
fprintf('\n===============================================================\n');
fprintf('SNR_out Sensitivity to AR(1) Coefficient (L = %d)\n', L_sens);
fprintf('---------------------------------------------------------------\n');
fprintf('      a       |    LMS (dB)   |   NLMS (dB)   |  VS-LMS (dB)\n');
fprintf('---------------------------------------------------------------\n');
for ia = 1:num_a
    fprintf('   %.2f      |    %8.2f   |    %8.2f   |    %8.2f\n', ...
            ar_coeffs(ia), ...
            SNR_out_mat(ia,1), ...
            SNR_out_mat(ia,2), ...
            SNR_out_mat(ia,3));
end
fprintf('===============================================================\n\n');

% Grouped bar plot: SNR_out vs a for different algorithms
figure;
bar(SNR_out_mat);
grid on;
set(gca,'XTick',1:num_a,'XTickLabel',arrayfun(@(a)sprintf('a=%.2f',a),ar_coeffs,'UniformOutput',false));
xlabel('AR(1) Coefficient a');
ylabel('Output SNR (dB)');
title('Sensitivity of Output SNR to AR(1) Coefficient for Different Algorithms');
legend(alg_subnames,'Location','best');
```

Appendix C
MATLAB other Function Script for Adaptive Filtering Experiments

Listing 2: lms_adaptive_filter.m

```
function [y, e, w_hist] = lms_adaptive_filter(x, d, L, mu)
% LMS_ADAPTIVE_FILTER  Standard LMS adaptive FIR filter.
%
% Inputs:
%   x   : reference input signal (Nx1)
%   d   : desired signal (Nx1)  (here: noisy speech)
%   L   : filter length
%   mu  : step size
%
% Outputs:
%   y       : filter output
%   e       : error signal, e(n) = d(n) - y(n)
%   w_hist  : L x N matrix of weight vectors over time

N = length(x);
x = x(:);          % ensure column vectors
d = d(:);

w = zeros(L,1);    % initial weights
w_hist = zeros(L,N);
y = zeros(N,1);
e = zeros(N,1);

for n = L:N
    % regressor vector x_n = [x(n), x(n-1), ..., x(n-L+1)]^T
    x_n = x(n:-1:n-L+1);

    % filter output
    y(n) = w.' * x_n;

    % error
    e(n) = d(n) - y(n);

    % weight update (LMS)
    w = w + mu * e(n) * x_n;

    % store weights
    w_hist(:, n) = w;
end

end
```

Listing 3: nlms_filter.m

```
function [y, e, w_hist] = nlms_filter(x, d, L, mu, eps)
% Normalized LMS Filter

x = x(:); d = d(:);
N = length(x);

if nargin < 5
    eps = 1e-6;   % regularization to avoid division by zero
end

w = zeros(L,1);
w_hist = zeros(L,N);
y = zeros(N,1);
e = zeros(N,1);

for n = L:N
    x_n = x(n:-1:n-L+1);
    y(n) = w.’ * x_n;
    e(n) = d(n) - y(n);

    % NLMS update
    power = (x_n.’*x_n) + eps;     % ||x_n||^2
    w = w + (mu/power) * e(n) * x_n;

    w_hist(:,n) = w;
end
end
```

Listing 4: leaky_lms_filter.m

```
function [y, e, w_hist] = leaky_lms_filter(x, d, L, mu, gamma)
% LEAKY_LMS_FILTER  Leaky LMS adaptive FIR filter.
% Implements the (1 - mu*gamma)*w update.
N = length(x);
x = x(:); d = d(:);
w = zeros(L,1); w_hist = zeros(L,N);
y = zeros(N,1); e = zeros(N,1);
for n = L:N
    x_n = x(n:-1:n-L+1);
    y(n) = w.' * x_n;
    e(n) = d(n) - y(n);
    % Leaky LMS Update: w = (1 - mu*gamma)*w + mu*e[n]*x[n]
    w = (1 - mu*gamma)*w + mu*e(n)*x_n;
    w_hist(:, n) = w;
end
end
```

Listing 5: block_lms_filter.m

```
function [y, e, w_hist] = block_lms_filter(x, d, L, mu, B)
% BLOCK_LMS_FILTER  Block LMS adaptive FIR filter (time-domain).
% B is the block size.
N = length(x);
x = x(:); d = d(:);
w = zeros(L,1); w_hist = zeros(L,N);
y = zeros(N,1); e = zeros(N,1);
for k = L : B : (N-B+1)
    grad = zeros(L,1);
    for l = 0 : B-1
        n = k + l;
        if n > N, break; end
        x_n = x(n:-1:n-L+1);
        y(n) = w.' * x_n;
        e(n) = d(n) - y(n);
        grad = grad + e(n)*x_n;
    end
    % Block Update: w_k+1 = w_k + (mu/B) * Grad_avg
    w = w + (mu/B)*grad;
    % Store weights for the entire block
    w_hist(:, k:min(k+B-1,N)) = repmat(w,1,min(B,N-k+1));
end
```

```
% Ensure remaining samples are processed with final weights
if k <= N
    w_hist(:, k:N) = repmat(w,1,N-k+1);
end
end
```

Listing 6: sign_error_lms_filter.m

```
function [y, e, w_hist] = sign_error_lms_filter(x, d, L, mu)
% SIGN_ERROR_LMS_FILTER  Sign-Error LMS adaptive FIR filter.
% Only the error signal is signed.
N = length(x);
x = x(:); d = d(:);
w = zeros(L,1); w_hist = zeros(L,N);
y = zeros(N,1); e = zeros(N,1);
for n = L:N
    x_n = x(n:-1:n-L+1);
    y(n) = w.' * x_n;
    e(n) = d(n) - y(n);
    % Sign-Error LMS Update: w = w + mu * sign(e[n]) * x[n]
    w = w + mu * sign(e(n)) * x_n;
    w_hist(:, n) = w;
end
end
```

Listing 7: sign_data_lms_filter.m

```
function [y, e, w_hist] = sign_data_lms_filter(x, d, L, mu)
% SIGN_DATA_LMS_FILTER  Sign-Data LMS adaptive FIR filter.
% Only the input data vector is signed element-wise.
N = length(x);
x = x(:); d = d(:);
w = zeros(L,1); w_hist = zeros(L,N);
y = zeros(N,1); e = zeros(N,1);
for n = L:N
    x_n = x(n:-1:n-L+1);
    y(n) = w.' * x_n;
    e(n) = d(n) - y(n);
    % Sign-Data LMS Update: w = w + mu * e[n] * sign(x[n])
    w = w + mu * e(n) * sign(x_n);
    w_hist(:, n) = w;
end
end
```

Listing 8: sign_sign_lms_filter.m

```
function [y, e, w_hist] = sign_sign_lms_filter(x, d, L, mu)
% SIGN_SIGN_LMS_FILTER  Sign-Sign LMS adaptive FIR filter.
% Both the error signal and the data vector are signed.
N = length(x);
x = x(:); d = d(:);
w = zeros(L,1); w_hist = zeros(L,N);
y = zeros(N,1); e = zeros(N,1);
for n = L:N
    x_n = x(n:-1:n-L+1);
    y(n) = w.' * x_n;
    e(n) = d(n) - y(n);
    % Sign-Sign LMS Update: w = w + mu * sign(e[n]) * sign(x[n])
    w = w + mu * sign(e(n)) .* sign(x_n);
    w_hist(:, n) = w;
end
end
```

Listing 9: vs_lms_filter.m

```
function [y, e, w_hist] = vs_lms_filter(x, d, L, mu_init, mu_min, mu_max, alpha, beta)
% VS_LMS_FILTER  Variable Step-Size LMS (VS-LMS) filter (per-tap).
N = length(x);
x = x(:); d = d(:);
w = zeros(L,1);
mu_vec = mu_init * ones(L,1);   % per-tap step sizes
prev_sign = zeros(L,1);         % for tracking sign change
w_hist = zeros(L,N);
y = zeros(N,1);
e = zeros(N,1);
for n = L:N
    x_n = x(n:-1:n-L+1);
    y(n) = w.' * x_n;
    e(n) = d(n) - y(n);

    for k = 1:L
        % Sign of the instantaneous gradient (e[n] * x[n])
        sgn = sign(e(n)*x_n(k));

        if sgn * prev_sign(k) > 0       % Gradient sign unchanged -> Increase mu
```

```
            mu_vec(k) = min(mu_vec(k) + alpha, mu_max);
        elseif sgn * prev_sign(k) < 0   % Gradient sign changed -> Decrease mu
            mu_vec(k) = max(mu_vec(k) * beta, mu_min);
        end

        % Per-tap weight update
        w(k) = w(k) + mu_vec(k)*e(n)*x_n(k);
        prev_sign(k) = sgn;
    end
    w_hist(:,n) = w;
end
end
```

Listing 10: wiener_fir_solution.m

```
function w_opt = wiener_fir_solution(x, d, L)
% WIENER_FIR_SOLUTION  Empirical Wiener solution for order-L FIR filter.
%
% x : input (reference) signal
% d : desired signal
% L : filter order

x = x(:); d = d(:);
N = length(x);

R = zeros(L, L);
p = zeros(L, 1);

for n = L:N
    x_n = x(n:-1:n-L+1);
    R = R + x_n * x_n.';
    p = p + d(n) * x_n;
end

R = R / (N-L+1);
p = p / (N-L+1);

w_opt = R \ p;       % R^{-1} p
end
```