System Identification:

An Approach Using LMS and NLMS Algorithms

Basheer Rjoub
Birzeit University
Birzeit, Ramallah, Palestine
1180291@student.birzeit.edu

Ahmad Hawareth
Birzeit University
Al-Irsal, Ramallah, Palestine
1180324@student.birzeit.edu

Abdallah Wohoush
Birzeit University
Abu-Qash, Ramallah, Palestine
1192055@student.birzeit.edu

***Abstract*— This article examines two adaptive filters algorithms, LMS and the normalized version NLMS, introducing the computations and implementation of these two algorithms that are mainly used for unknown system identification. Moreover, comparing the results obtained from the adaptation of these adaptive filters, deriving the coefficients of the filters and using these filters for noise cancellation purposes, and finally compare the results obtained from the two algorithms.**

Keywords—NLMS, LMS, Adaptive Filter, discrete time, noise, convolution, spectrum.

I. Introduction

Adaptive filters are one type of digital signal processing systems, that make use of the feedback system to generate an approximation for the unknown system, in ordinary cases these types of filters are used to specify the coefficients of an unknown system, also they can do the same job even with interfering noise to the input signal.

These types of filters are usually used in many applications such as noise reduction, system identification and equalization, there are many types of adaptive filters algorithms these types include the Least Mean Squares (LMS) algorithm and the Normalized Least Mean Squares (NLMS), these two algorithms are examined and derived in this paper.

II. Problem Specification

Sometimes there are systems, are hard to obtain their transfer function in order to specify its behavior in the phase and amplitude response, or some systems that interfering with noise. So, using adaptive filters can be the solution for this problem in the real-life applications, as the adaptive filter tries to learn from the signal, as it feeds the system an input and used this input in its internal behavior in order to learn the response of the unknown system.

III. Data

Into mimicking the behavioral of the signals in the real-life applications, as known any signal can be formed as a Fourier series as a summation of sinusoidal, so in this paper using a simple sinusoidal as an input to the adaptive filter.

First of all, generating the input signal:

$$x[n] = \cos(0.03\pi n), N = 2000 \text{ samples.} \quad (3)$$

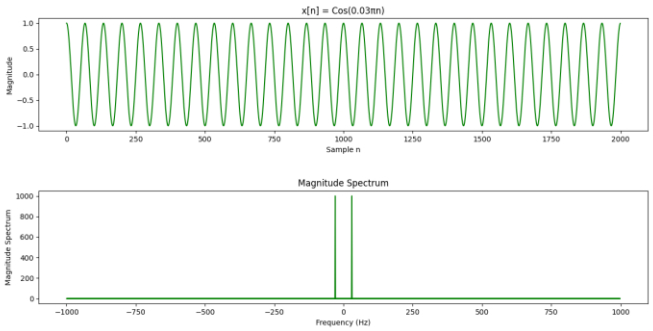

Fig. 1 x[n] and the spectrum of the signal

Finding Finite impulse response of the system, that have the following equation in time domain:

$$d[n] = x[n] - 2x[n-1] + 4x[n-2] \quad (3)$$

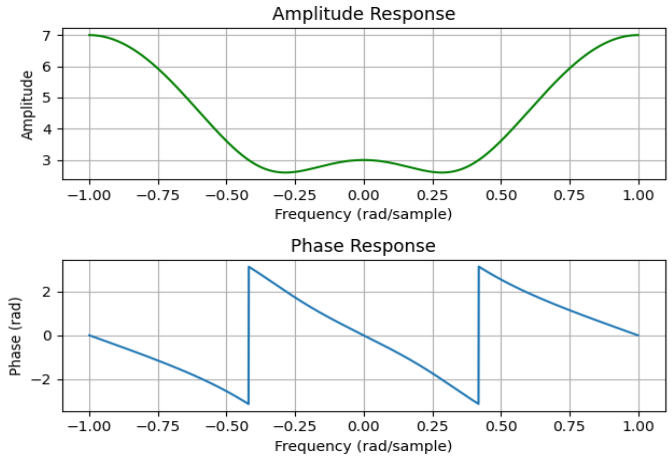

Fig. 2 Amplitude and phase response of d[n]

IV. Evaluation Criteria

To evaluate the results obtained in this paper, the best evaluation techniques can include the error convergence, because the error convergence to zero specifies that the adaptive filter has learned well the signal behavioral, moreover other evaluation metric can be used is to compare and visualize the behavior of the signal in time domain and also in frequency domain to compare the phase and amplitude response of the original system with the adaptive filter.

V. Methodology of LMS And An Approach To System Identification.

LMS is one of the most popular and widely used algorithms regarding adaptive filters, it is a gradient error minimization algorithm that is based on a single error value to adjust the coefficients until reaching a good point of accuracy on minimizing the error hence providing a good approximation for the original unknown system. [1]

The algorithm for the LMS adaptive filter goes as in Algorithm 1, first of all initializing the parameters for the LMS function, then repeating for the number of samples of the signal into minimizing gradually the amount of error, then using this error to learn the coefficients and adjusting the coefficients w into the new corresponding values, and so on, until deriving the approximating equation of the unknown systems response, in the case of DSP the approximating signal is a list of impulses for the specified number of samples.

Algorithm 1 LMS Algorithm for Mth-Order filter

Inputs:
```
M: Filter length
N: size of input signal
Mu: Learning rate
x[n]: Input Signal
w: vector of Length M initially 0s
w1: Matrix M x N
```
Repeat:
```
y[n] = w^T[n]x[n]
e[n] = d[n] - y[n]
w[n+1] = w[n] + 2Mu*e[n]*x[n]
w1[n-M:] = w[1,]
```
Output:
```
Lists: y,e,w, w1
```

Fig. 3 LMS Algorithm for system identification

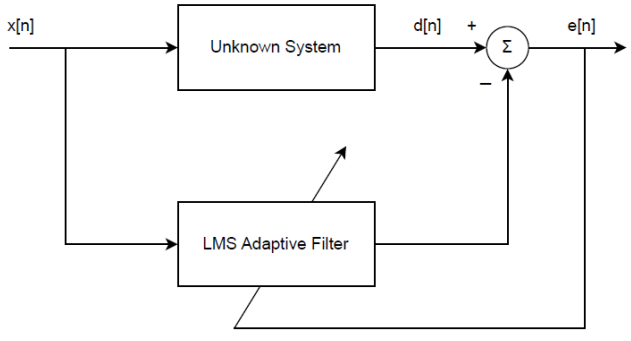

Fig. 4 LMS adaptive filter block diagram

VI. Methodology of NLMS And an approach to System Identication

Regarding LMS, the step size or what mean as the learning rate is always fixed for each iteration, this will make an overhead of understanding the behavior of the unknown system before the real implementation of the adaptive filter, which is hard to maintain. [2]

The solution for the fixed step size in LMS came as the Normalized LMS (NLMS), which avoids this problem by computing the step size in each iteration as in the following equation:

$$Mu = \frac{1}{x[n]^T x[n]} \quad (1)$$

And then the weights are updated as in the following equation:

$$w(n+1) = w(n) + \mathrm{u(n)e(n)x(n)} \quad (2)$$

Each iteration requires N more multiplications than the standard LMS algorithm. However, the increase of complexity isn't something to worry about when getting more stable response and better echo attenuation.

VII. Results of LMS Algorithm

First of all, after implementing the LMS algorithm which is included in appendix, the several figures are used to observe the behavioral of this Algorithm, as in the following figures the algorithm is ran with 2000 iterations and mu = 0.01 with a clear channel, meaning that there is no interfering noise (ideal case).

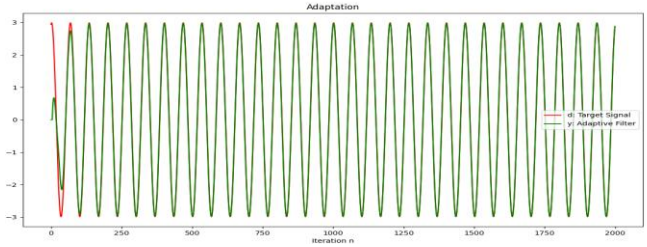

Fig. 5 Adaptation of LMS filter with learning rate = 0.01 and clear channel

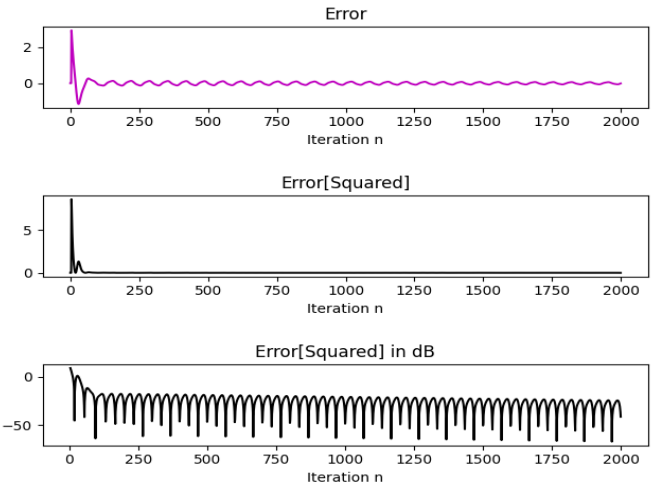

Fig. 6 Error values with different scales for LMS with zero noise and 0.01 learning rate

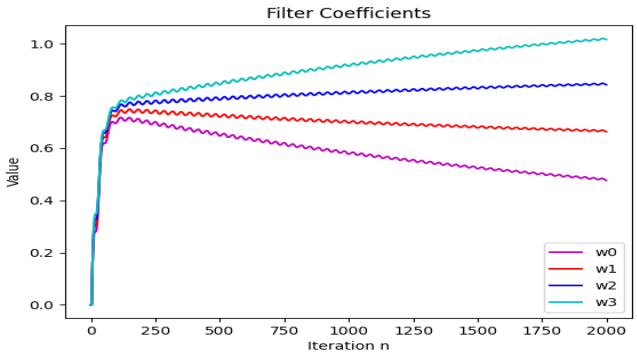

Fig. 7 Convergence of AF coefficients for LMS with mu=0.01 and no noise occurring

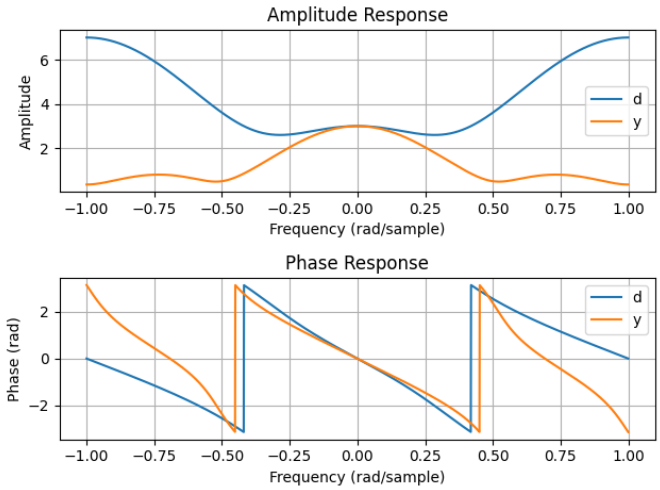

Fig. 8 Am. and Phase response of LMS with mu=0.01 and clear channel

Some results can be obtained from the previous figures is that using the LMS filter with clear channel (which is not the case in the real life) can go well in learning the shape of the channel and approximate the amplitude and phase response of it.

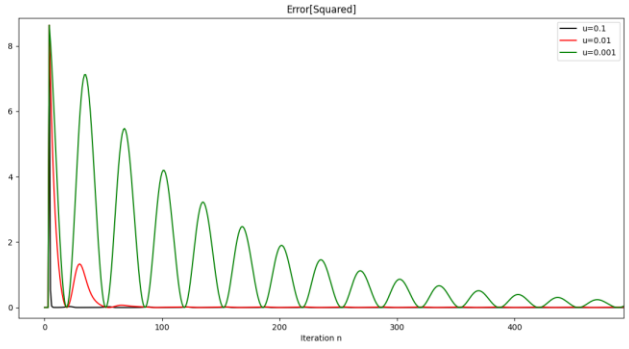

Fig. 9 Effect of mu on LMS algorithm

As from fig. 9, decreasing the learning rate will make it harder to reach to the steady state error. However, with small learning rate the adjustment of the coefficients will be slight, but the accuracy for learning the shape of the unknown signal will be better.

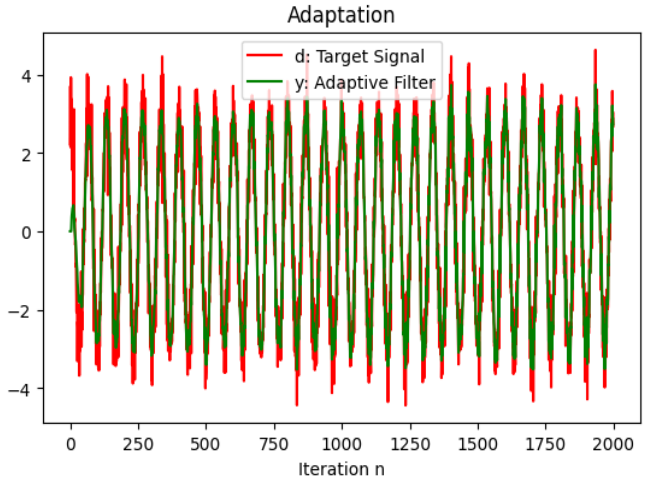

Fig. 10 Adaptation of LMS filter with 30db SNR noisy channel

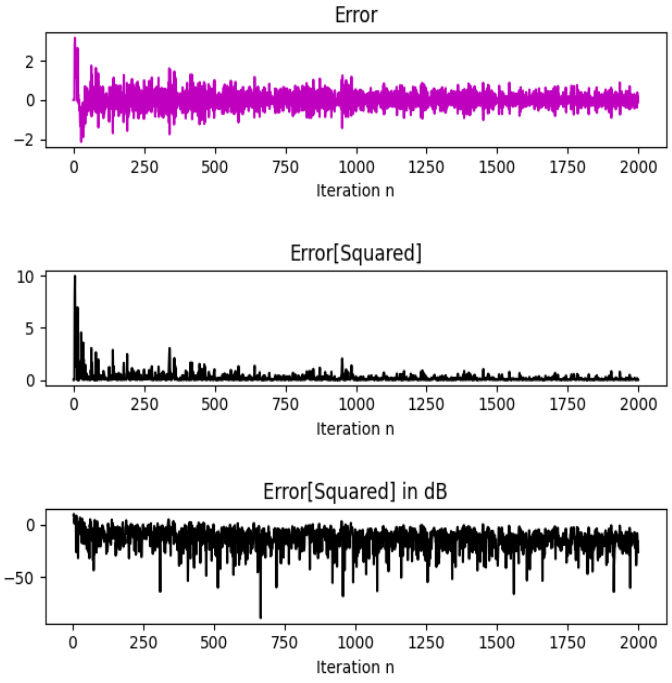

Fig. 11 LMS error values with 40db interfering noise

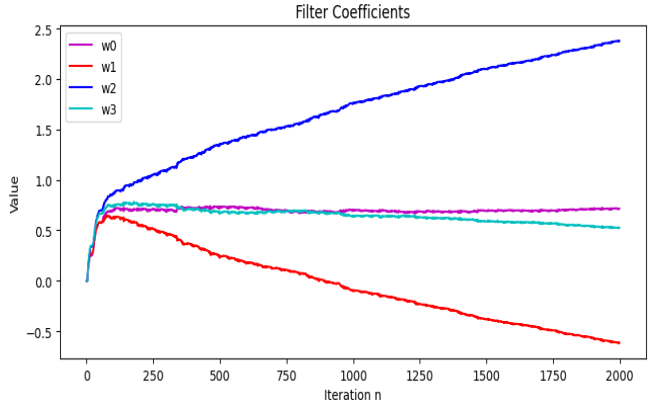

Fig. 12 Convergence of the coefficients for LMS with 30db noise

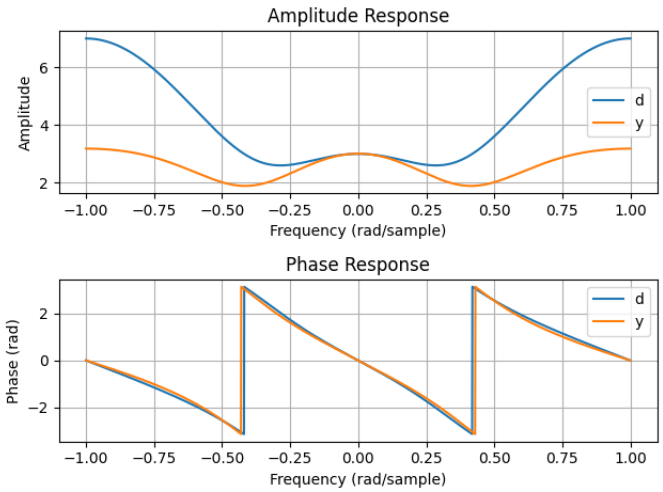

Fig. 13 Amplitude and phase response of AF and the original signal with 30db noise, and learning rate 0.01

To compare the effect of changing the learning rate, and stabilizing the interfering noise to 30db, the same results can be obtained that decreasing the learning rate will make the learning of the filter more accurate but the overhead is that it takes more iterations to reach the steady state error.

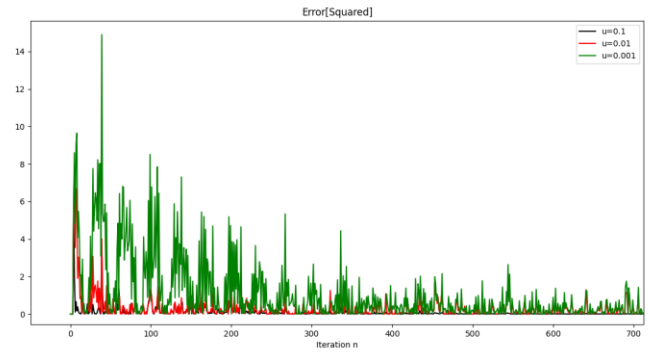

Fig. 14 Different adaptive filter error curves for 30db

Ensemble averaging is a technique in adaptive filtering that combines the outputs of multiple instances of the same algorithm, each operating on a different realization of the input data, in order to improve the overall performance. The basic idea behind ensemble averaging is to average the outputs of multiple instances of the same adaptive filter algorithm, each of which is trained on a different realization of the input data. This averaging can reduce the variance of the output and improve the performance of the filter, in the flowing figure the ensemble averaging is used with 1000 trails.

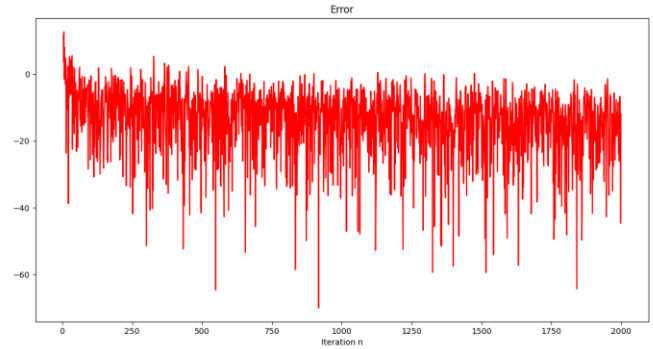

Fig. 15 Averaged Error of the 1000 trails

VIII. Results of NLMS Algorithm

For the implementation of the NLMS algorithm, starting by step size μ = 0.01, with the signal x[n] which is cleared of the noise, hence the following results should be obtained.

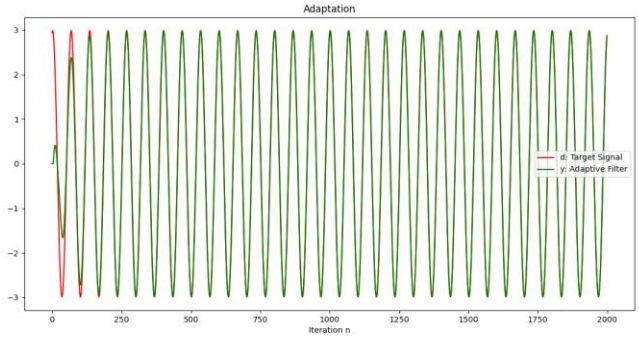

Fig. 16 Adaptation of the adaptive filter to learn the response of d[n]

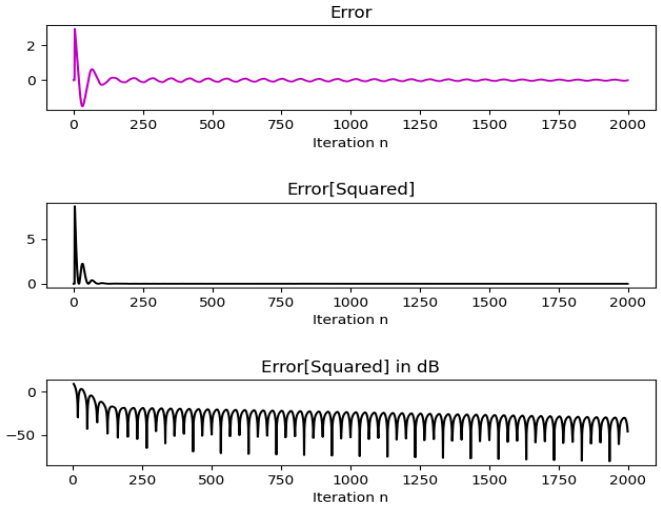

Fig. 17 Error of the adaptation operation

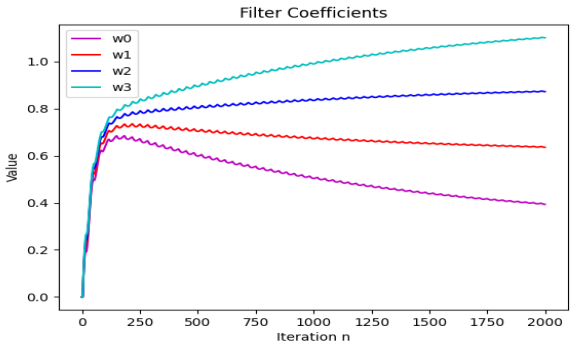

Fig. 18 Convergence of the coefficients of the adaptive filter

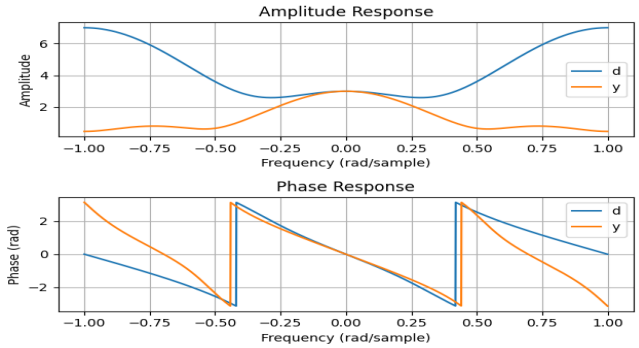

Fig. 19 Amplitude and phase response of both of d[n] and adaptive filter y[n]

From the previous figures, it appears that the 4th order adaptive filter using the NLMS algorithm can quickly adapt to learn the shape of the original unknown system, because of the fast learning and non-occurring noise which is the idea case, then the error term will approach in no time to the original signal so the adjustment for the weights of the adaptive filter would stop with the first terms of samples, so they won't be or shouldn't necessary be as the original filter's coefficient's.

Table 1 Coefficients of adaptive filter y

w0	0.39330899842478034
w1	0.6355345559673138
w2	0.8721190637991499
w3	1.1009625810800514

Another observation can be derived from the results, that the magnitude and the phase response of the two systems are almost exact with a little difference, which means that the algorithm is efficient in learning the shape of the signal in a reasonable time.

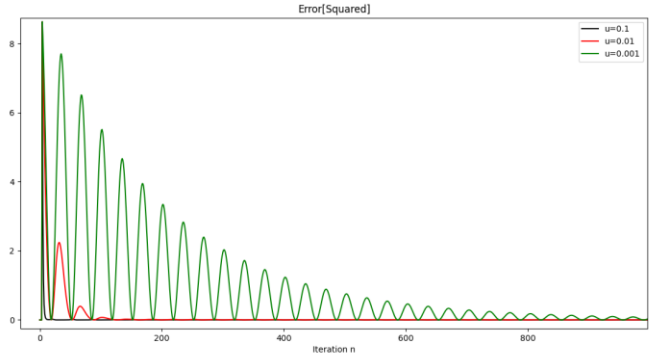

Fig. 20 Effect of using different values of μ

As shown from the previous figure, decreasing the learning rate μ will result in reaching the steady state in longer time. However, this tends to give the algorithm more sensitivity in the adjustment of the coefficients of the filter, but the only draw back that the algorithm needs more iterations to get to a small negligible value of error.

Now with adding noise to the original signal x[n], the algorithm now will have another challenge which is cancelling the affect of noise in the signal alongside with approximating the original signal.

First, let the signal x[n] pass in a noisy channel with SNR 40db, the noise act like a normal distribution with mean 0.

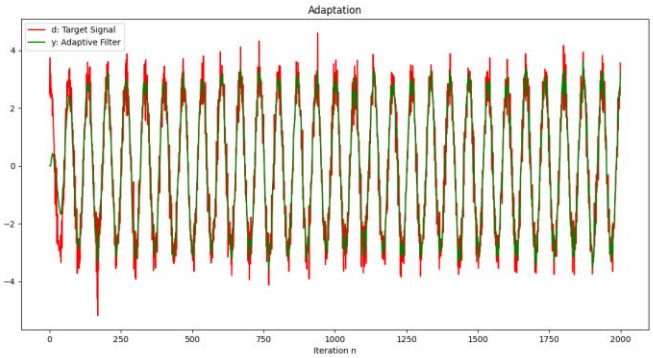

Fig. 21 Adaptation of the AF with 40db noise added to the input signal x[n]

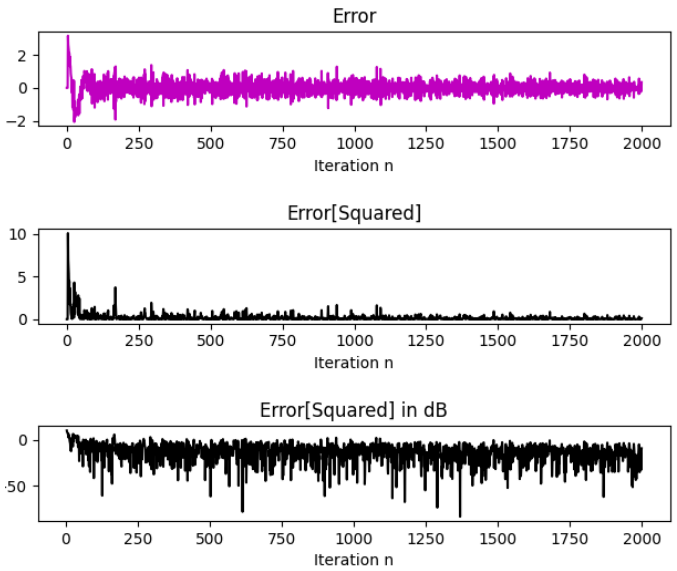

Fig. 22 Increasing of the value of error after adding 40db noise

As predicted, the value of error in the adaptation process increased, moreover it was hard for the filter to adjust itself to the original signal's shape.

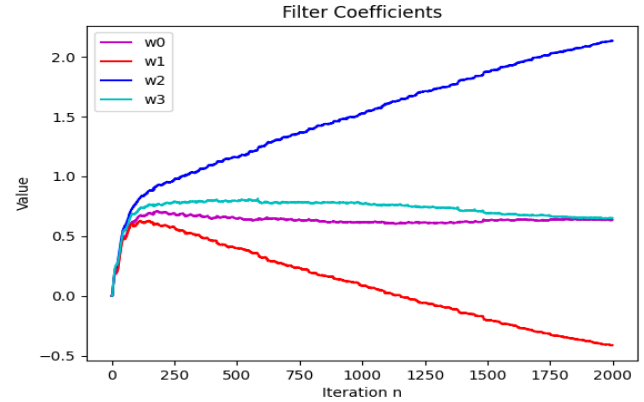

Fig. 23 Convergence of filter's coefficients

Table 2 Filter's coefficients with 40db noise added

w0	0.6360727414163714
w1	-0.4124014515623481
w2	2.135348475032886
w3	0.6522464988672619

The phase response for the system became better with the presence of noise, this can be due to the unpredictable value of error, then the filter tries its best to learn the coefficients of the original signal because of the learning rate μ in the presence of noise is always non-negligible value, hence makes a big difference in the process of the coefficients adjustment

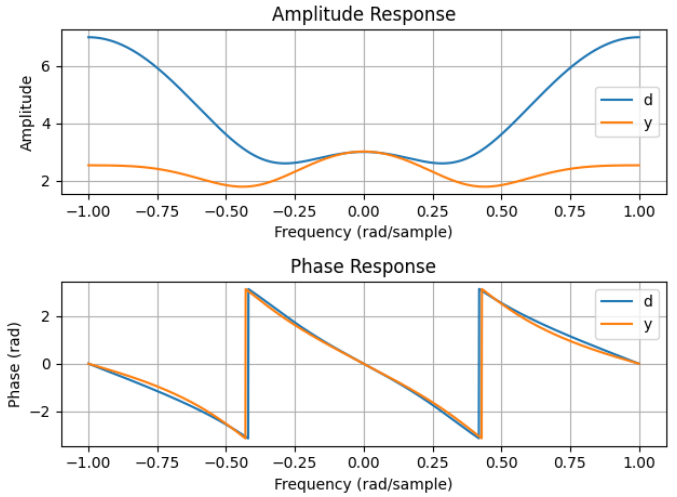

Fig. 24 Phase and amplitude response of d[n] and y[n] in presence of noise in the input signal x[n]

Decreasing the SNR to 30db, with modification to the learning rate, one can observe that as in the previous results decreasing the learning rate will result in a hard adaptation meaning that the AF will modify the coefficients slightly in each iteration, but the overall accuracy will be pleasant after all.

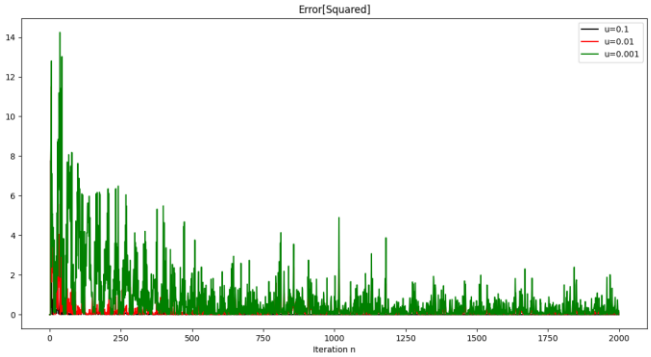

Fig. 25 Error values with 30db SNR and different learning rates.

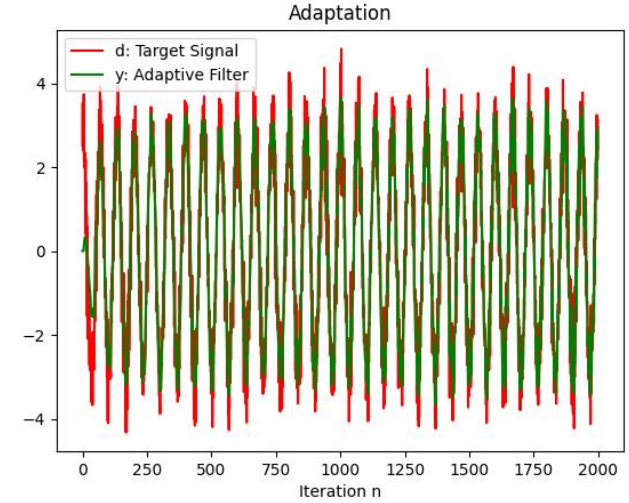

Fig. 26 Adaptation of AF with 30db SNR μ =0.01

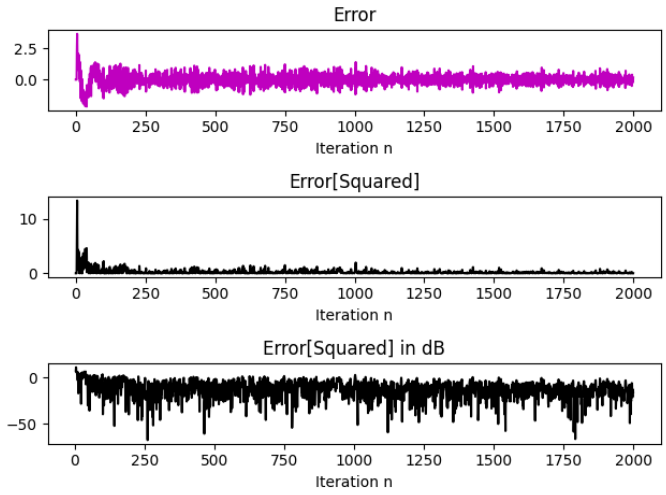

Fig. 27 Error values with 30db SNR and μ=0.01

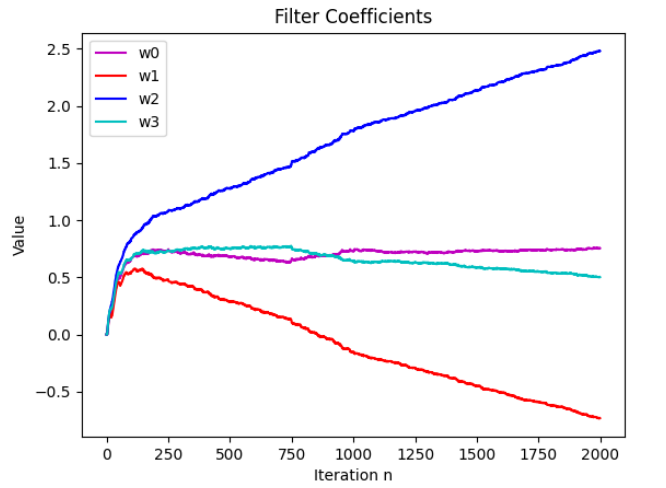

Fig. 28 Coefficients convergence with 30db SNR and μ=0.01

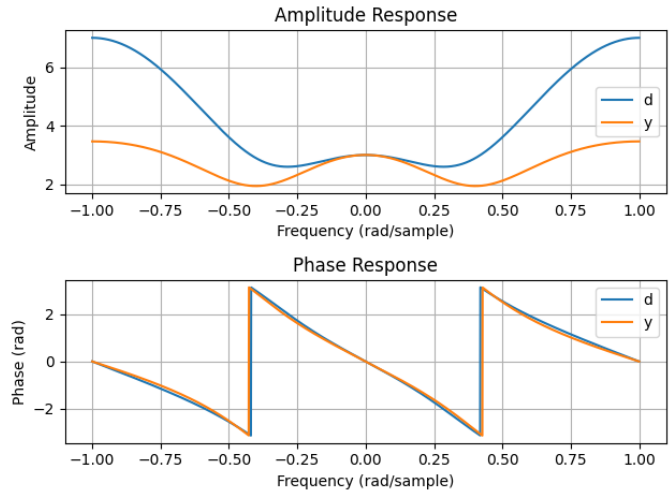

Fig. 29 Phase and Amplitude response of Adaptive Filter with interfering 30db noise and μ=0.01

Some times when having very noisy channel it would be very hard for the adaptive filter to obtain the original signal, the solution for this can be obtained trivially with a large number of iterations then averaging these results to give the mean, because of the large number of iterations the mean will approximately be in the zero (middle) which is the original signal.

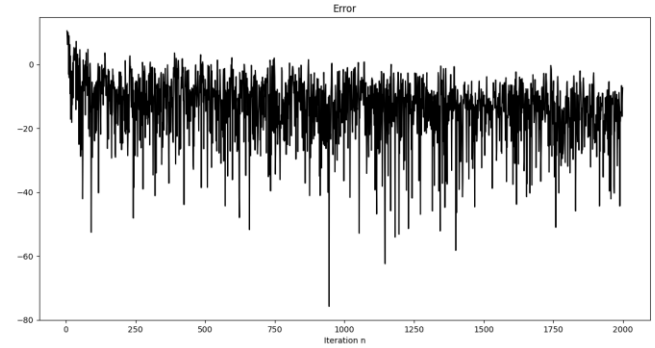

Fig. 30 Average error in db for 1000 iteration of each algorithm iterations

IX. Comparing LMS and NMLS

For the comparison of these two algorithms, first the initial learning rate should be fixed for the two algorithms so it will be fixed to 0.01, and no interfering noise to the signal will be assumed, so that no random occurring events to result in wrong assumptions.

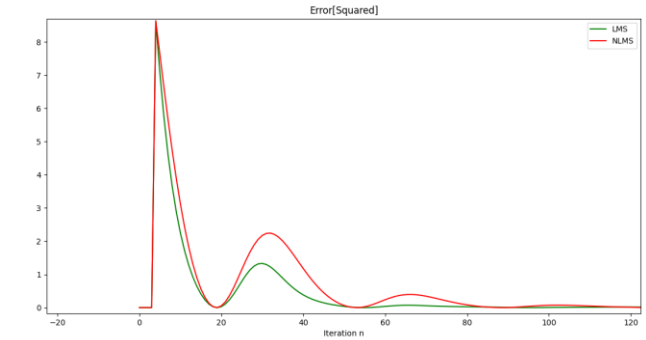

Fig. 31 NMLS vs LMS error squared comparison

The NLMS as in fig. 31 appears to be slow in convergence in comparison with LMS algorithm. In general, the NLMS algorithm tends to converge to a steady-state error slower than the LMS algorithm. This is because the normalization step in the NLMS algorithm reduces the magnitude of the weight update, which slows down the convergence process. On the other hand, the LMS algorithm updates the weights more aggressively, which can lead to instability, but also allows for faster convergence.

X. Development

Into developing a more robust adaptive filter, reducing the learning rate can make the algorithm stronger and more accurate, but there is more overhead occurring because of increasing the time complexity. Moreover, using other adaptive filter algorithms can sometimes be better suited for some

systems with specific events like specific type of noise can result in better accuracy like RLS, Kalman filter and APA.

XI. Conclusions

In practice, the choice between LMS and NLMS will depend on the specific requirements of the application. If stability is a concern, the NLMS algorithm may be the better choice, while if faster convergence is desired, the LMS algorithm may be more appropriate.

NLMS is more used in real-life application due to its robust and accuracy, but they both have their applications to use, and depends on the compromise between the accuracy and complexity.

References

[1] Widrow, Bernard. "Thinking about thinking: the discovery of the LMS algorithm." IEEE Signal Processing Magazine 22.1 (2005): 100-106.

[2] Srinivas, Kavalipati. (2011). ADAPTIVE ALGORITHMS FOR ACOUSTIC ECHO CANCELLATION IN SPEECH PROCESSING. arpapress. 7. 5.

[3] A. Pandey, L.D. Malviya, and V. Sharma, "Comparative Study of LMS and NLMS Algorithms in Adaptive Equalizer," International Journal of Engineering Research and Applications, vol. 2, no. 3, pp. 1584-1587, May-Jun 2012.

[4] Rahman, Md. Faisal & Huq, A. (2015). Comparison between LMS & NLMS Algorithms in Adaptive Noise Cancellation for Speech Enhancement (Dhaka University Journal of Applied Science and Engineering, ISSN: 2218-7413, Vol. 3(1), 107-111, 2015(January)). 107-111.

[5] Singh, Arpit & Pandya, Rahul & Sharma, Ashish & Singh, Tarush. (2021). Effect Of Learning Rates On Adaptive Filter Algorithms And Implementation Of Kernel Adaptive Filters.

Appendix

Code1

```
import numpy as np
import matplotlib.pylab as plt


def lms(x, dn, mu, M, N):
    N = N
    w = np.zeros(M)
    w1 = np.zeros((N, M))
    y = np.zeros(N)
    e = np.zeros(N)
    for n in range(M, N):
        x1 = np.flip(x[n - M + 1 : n + 1])
        # y[n] = np.dot(w, x1)
        y[n] = sum(w[i] * x1[i] for i in range(M))
        e[n] = dn[n] - y[n]
        # w = w + 2 * mu * e[n] * x1
        for i in range(M):
            w[i] = w[i] + 2 * mu * e[n] * x1[i]
        w1[n] = w

    J = e**2
    return w, y, e, J, w1

def nlms(x, dn, mu, M, N):
    N = N
    w = np.zeros(M)
    w1 = np.zeros((N, M))
    y = np.zeros(N)
    e = np.zeros(N)
    for n in range(M, N):
        x1 = np.flip(x[n - M + 1 : n + 1])
        y[n] = sum(w[i] * x1[i] for i in range(M))
        e[n] = dn[n] - y[n]
        x1_norm = np.linalg.norm(x1)
        if x1_norm != 0:
            w = w + (2 * mu * e[n] / x1_norm) * x1
        w1[n] = w
    J = e**2
    return w, y, e, J, w1

def awgn(x, snr, signalpower):
    # Calculate the noise power
    noise_power = signalpower / snr

    # Generate Gaussian noise
```

```
    noise = np.random.normal(scale=np.sqrt(noise_power), size=x.shape)

    # Add the noise to the signal
    return x + noise

N = 2000
n = np.arange(N)
x_n = np.cos(0.03 * np.pi * n)
# Add white Gaussian noise to the signal with a SNR of 20 dB
snr = 40
signalpower = np.mean(np.power(x_n, 2))
# x_n = awgn(x_n, snr, signalpower)
# Plot Noise
# plt.plot(n, x_n)
# plt.show()
# plt.close()
# exit()

d_n = x_n - (2 * np.roll(x_n, 1)) + (4 * np.roll(x_n, 2))
print(d_n[:10])
# LMS
# Initialize
M = 4  # Filter length
mu = 0.01  # learning rate
# Single iteration
w, y, e, J, w1 = nlms(x_n, d_n, mu, M, N)

# 1000 Trails
# all = []
# y_list = []
# for i in range(1000):
#     w, y, e, J, w1 = lms(x_n, d_n, mu, M)
#     J_dB = 10 * np.log10(J)
#     all.append(J_dB)
#     y_list.append(y)
# average = np.mean(all, axis=0)
# y_list = np.mean(y_list, axis=0)
# # Plot Average error
# plt.title("Error")
# plt.xlabel("Iteration n")
# plt.plot(average, "r")
# plt.show()
# plt.close()
# # Plot the adaption graph
# plt.title("Adaptation")
# plt.xlabel("Iteration n")
```

```
# # plt.plot(d_n, "r", label="d: Target Signal")
# plt.plot(y_list, "g", label="y: Adaptive Filter")
# plt.show()
# plt.close()
# exit()

for i in range(len(w1[-1])):
    print(f"W{i}: {w1[-1][i]}")

colors = ["m", "r", "b", "c", "y", "g", "k"]

# Plot the adaption graph
plt.title("Adaptation")
plt.xlabel("Iteration n")
plt.plot(d_n, "r", label="d: Target Signal")
plt.plot(y, "g", label="y: Adaptive Filter")
plt.legend()
plt.show()
plt.close()

# Plot Error Graph
plt.subplot(3, 1, 1)
plt.title("Error")
plt.xlabel("Iteration n")
plt.plot(e, "m")



# Plot Error Squared Graph
plt.subplot(3, 1, 2)
plt.title("Error[Squared]")
plt.xlabel("Iteration n")
plt.plot(J, "k")



# Calculate J in dB scale
J_dB = 10 * np.log10(J)

# Plot J_dB Graph
plt.subplot(3, 1, 3)
plt.title("Error[Squared] in dB")
plt.xlabel("Iteration n")
plt.plot(J_dB, "k")
plt.tight_layout()
plt.show()
plt.close()

# Plot filter Ciefficients
```

```
plt.title("Filter Coefficients")
plt.xlabel("Iteration n")
plt.ylabel("Value")
for i in range(M):
    plt.plot([wi[i] for wi in w1], colors[i], label=f"w{i}")
plt.legend()
plt.show()

plt.close()

# Define the coefficients of the system
b_d = [1, -2, 4]
b_y = w1[-1]
# Define the frequency range
w = np.linspace(-np.pi, np.pi, 2000)

# Calculate the frequency response
H_y = np.zeros(w.shape, dtype=complex)
H_d = np.zeros(w.shape, dtype=complex)
for i in range(len(b_d)):
    H_d += b_d[i] * np.exp(-1j * i * w)
for i in range(len(b_y)):
    H_y += b_y[i] * np.exp(-1j * i * w)
w = w / (np.pi)



plt.figure()
plt.subplot(2, 1, 1)
plt.plot(w, np.abs(H_d), label="d")
plt.plot(w, np.abs(H_y), label="y")
plt.xlabel("Frequency (rad/sample)")
plt.ylabel("Amplitude")
plt.title("Amplitude Response")
plt.legend()
plt.grid()

plt.subplot(2, 1, 2)
plt.plot(w, np.angle(H_d), label="d")
plt.plot(w, np.angle(H_y), label="y")
plt.xlabel("Frequency (rad/sample)")
plt.ylabel("Phase (rad)")
plt.title("Phase Response")
plt.legend()
plt.grid()
plt.tight_layout()
plt.show()
plt.close()
```

```
exit()
```

Code2

```
    import numpy as np
    import matplotlib.pyplot as plt

    N = 2000
    n = np.arange(N)
    x = np.cos(0.03 * np.pi * n)

    D = np.fft.fft(x)
    f = np.fft.fftfreq(N, 1 / N)

    plt.subplot(2, 1, 1)
    plt.xlabel("Sample n")
    plt.ylabel("Magnitude")
    plt.title("x[n] = Cos(0.03πn)")
    plt.plot(x, "g")

    plt.subplot(2, 1, 2)
    plt.plot(f, np.abs(D), "g")
    plt.xlabel("Frequency (Hz)")
    plt.ylabel("Magnitude Spectrum")
    plt.title("Magnitude Spectrum")

    plt.tight_layout()
plt.show()
```