MSc- Adaptive Signal Processing Assignment

This assignment requires access to a UNIX workstation or PC running MATLAB together with hardcopy facility.

Results should be recorded in a laboratory notebook.

One of the assignments should be written up as a formal report.

Assignment 1

Title: **Adaptive Algorithms for Echo Cancellation**

Aim:

- To compare the performance of the Least Mean Square (LMS) and Recursive Least Squares (ALS) adaptation algorithms in the context of echo cancellation.

Background

Adaptive digital signal processing is the study of algorithms and techniques which have the capacity to vary in sympathy with changing statistical properties, characteristic of many real signals. Such techniques have been successfully applied in many application areas, for example, channel equalisation in communications beamforming for seismic prospecting, EGG monitoring in medicine, analysis of multiphase flow and the control of dynamic systems.

This assignment is based upon a recent application of adaptive digital signal processing to echo cancellation. In “hands-free” mobile telecommunications, the mobile unit which contains the loudspeaker and microphone is placed, for convenience, at some distance from the local speaker. Therefore, when the remote (far-end) speaker is talking, there is acoustic coupling between the loudspeaker and microphone of the mobile unit, which leads to the far-end speaker hearing a disturbing echo of his own voice. Elimination of this echo can be achieved with an adaptive filter, which models the time-varying acoustic coupling. The adaptive filter is applied in a system identification structure as shown in Figure 1. The input, $x(k)$, where k represents the sample index, corresponds to the loudspeaker signal, namely the far-end speech; the unknown system is the time-varying acoustic path between the loudspeaker and the microphone; the noise term, $n(k)$, represents that component of the microphone signal which has not been generated by the loudspeaker signal; and $d(k)$ is the microphone signal - the desired response for the adaptive filter. The adaptive filter attempts, in real-time, to track the time-variations of the acoustic path in order to generate, at its output, a signal, $y(k)$, which matches the component of the microphone signal due to the acoustic coupling between the loudspeaker and the microphone.

The adaptive filter is based upon a Finite Impulse Response (FIR) digital filter structure together with an adaptation algorithm to adjust the coefficients of the filter. The adaptation algorithm adjusts the coefficients of the FIR filter so as to minimise some function of the error, $e(k)$, between the desired response, $d(k)$, and the output of the adaptive filter, $y(k)$.

Project

Two families of adaptation algorithms are to be investigated, namely Least Mean Square (LMS) and Recursive Least Squares (RLS).

The basic LMS algorithm approximately minimises the mean square error $J = \mathrm{E}\{e^2(k)\}$ where the error is given by $d(k) - \underline{w}^T(k)\underline{x}(k)$ and $\underline{w}(k) = [w_1(k), w_2(k), ..., w_L(k)]^T$ is the coefficient vector of the adaptive filter of length L, and $\underline{x}(k) = [x(k), x(k-1), ..., x(k-L+1)]^T$ is the adaptive filter data vector. The coefficient vector update equation for the LMS algorithm is given by

$$\underline{w}(k+1) = \underline{w}(k) + 2\mu\underline{x}(k)e(k) \quad (1)$$

where the right hand term, $\underline{x}(k)e(k)$, is an instantaneous estimate of the negative gradient of the error performance surface J [Haykin, 1996], and m is the adaptation gain. The coefficient vector of the adaptive filter is generally initialised to zero.

Exercise 1.1 Write a MATLAB function to perform the LMS algorithm. The function inputs should be vectors for the desired response signal, $d(k)$, and the adaptive filter input, $x(k)$, whereas scalars for the adaptation gain, m, and the length of the adaptive filter L. The output should be the error signal, $e(k)$, together with the weight vector of the adaptive filter at each sample k. The call of the function from within MATLAB should have the form [e,$wmat$] = lms(x,d, m ,L) where x and d are $N \times 1$ vectors, m and L are scalars and e is a $N \times 1$ vector and $wmat$ is a $N \times L$ matrix.

The presence of feedback within the LMS algorithm, that is the output error is used to adjust the coefficients of the adaptive filter, may lead to instability. The adaptation gain, m, is therefore the key parameter which controls how the adaptive filter behaves and it should be chosen to lie within the range [Haykin, 1996]

$$0 < m < \frac{1}{LP_x} \quad (2)$$

where P_x is the power of the input signal to the adaptive filter.

Exercise 1.2 Use the LMS routine from Exercise 1.1 in a system identification simulation. Represent the unknown system with a length 9 FIR filter with an impulse response sequence vector $\underline{w}$ of the form

$$\frac{1}{k}\exp(-(k-4)^2/4) \quad k=1,\ldots,9$$

and employ the *randn(200,* 1) function within MATLAB to generate $x(k)$ which will have unity power and zero mean. The desired response signal is generated with the *filter*(*w*,[1],*x)* function.

(i) Assume the additive noise, *n*(*k*), is zero and the length of the adaptive filter, *L*, equals that of the unknown system. Calculate the upper bound for the adaptation gain from equation (2), and multiply this by 1/3 [Bellanger, 1987]. Run the LMS algorithm with this setting for m and use *plot*() and *semilogy*() to observe the convergence of the squared error, $e^2(k)$, and the weight error vector norm with sample number - such plots give an indication of the learning rate of the adaptation algorithm. The normalised weight error vector norm is given by $\dfrac{\|\underline{w}(k)-\underline{w}\|^2}{\|\underline{w}\|^2}$,

where $\underline{w}(k)$ is the coefficient vector of the adaptive filter at sample number k and the norm $\|\underline{\bullet}\|^2$ represents the sum of squared values of the vector argument. One example of MATLAB code to achieve this is

```
wdn = wid*wid';                % Calculate denominator
temp = ones(200,1);
wi = temp*wid;
wnm = sum(((w-wi).^2)');
wnorm = wnm/wdn;
```

Plot the final weight vector for the adaptive filter, $\underline{w}$(*200*). Choose the length of the adaptive filter to be less than and greater than that of the unknown system. e.g. 5 and 11; recalculate the adaptation gain for each case, simulate and explain the results e.g. the final weight vectors and the effect upon the learning curves.

(ii) Repeat (i) 20 times for independent input sequences, i.e. call *randn*(200,1) and *filter*(*w*,[1],*x*) to generate new input and desired response vectors with the length of the adaptive filter equal to that of the unknown system. Plot the average squared error and weight error vector norms - this corresponds to an approximate ensemble average [Haykin, 1996]. Comment on the results.

(iii) Repeat (ii) for different settings of the adaptation gain m e.g. [0.5,0.01,0.001]. What are the effects of increasing and decreasing the adaptation gain? Comment on the convergence and misadjustment of the LMS algorithm [Haykin, 1996].

(iv) Add zero mean, independent noise to the desired response signal so that the signal-to-noise ratio of the desired response is 20dB, that is the ratio of the output power of the unknown system to the power of the independent noise is 100, and repeat (ii). Comment on the convergence and misadjustment performance of the LMS algorithm.

(v) In the echo cancellation application, the input to the adaptive filter, is speech. A speech file called s1.mat is available which can be loaded with the *load* command within MATLAB. Use s1 as the input $\underline{x}(k)$. Comment upon the choice of adaptation gain for the LMS algorithm when the input to the adaptive filter is a real speech signal. A modification to the LMS algorithm is the Normalised LMS algorithm with an update equation of the form

$$\underline{w}(k+1) = \underline{w}(k) + \frac{2\mu}{1+\|\underline{x}(k)\|^2}\underline{x}(k)e(k) \quad (3)$$

Why is this algorithm more suitable for use with real signals? Write a MATLAB function to simulate this algorithm. Apply this algorithm to the system identification simulation with s1 as input. Comment upon its performance. Compare the complexities of the LMS and NLMS algorithms for real-time implementation. What is the implication of the use of a fixed point precision digital signal processor chip upon the operation of a LMS adaptive filter [Haykin, 1996]?

In contrast, the RLS algorithm is based on the exact minimisation of the sum of weighted errors squared given by

$$J(k) = \sum_{l=1}^{k} l^{k-l}(d(l) - \underline{w}^T(k)\underline{x}(l))^2 \quad (4)$$

where $l \in (0,1]$ is the forgetting factor which describes the memory of the algorithm. Equation (4) is based on an underlying growing exponential window of the input data. The length of the window is given, to a first approximation, by $1/(1-l)$ [Haykin, 1996]. A unity value for the forgetting factor corresponds to infinite memory and is only suitable when statistical stationarity can be assumed. The update equation for the RLS algorithm is given by

$$\underline{w}(k+1) = \underline{w}(k) + R^{-1}(k)\underline{x}(k)e(k) \quad (5)$$

where $R^{-1}(k)$ is the inverse of the data input matrix given by $\sum_{l=1}^{k} l^{k-l}\underline{x}(l)\underline{x}^T(l)$.

Notice the commonality between the adaptation algorithms, eqns. (1), (3) and (5), they are all based on the same form of update equation. The distinguishing feature is the form and complexity of the adaptation gain, for LMS it is a scalar constant, for NLMS it is time varying scalar, whereas for RLS it is a time varying matrix.

Matrix inversion is an $O(L^3)$ operation, therefore it would not be feasible for a real-time algorithm to calculate a new value for $R^{-1}(k)$ at each sample. However, this is not necessary. Some complexity reduction results from writing $R(k) = \; l\,R(k-1) + \underline{x}(k)\underline{x}^T(k)$ which shows that the change in the data input matrix from one iteration to the next is a simple vector outer product of the most recent data vector, that is, a rank one matrix. The matrix inversion lemma [Haykin, 1996] can then be applied to obtain a simple update for $R^{-1}(k)$ which does not require a matrix inversion, just division by a scalar, i.e.

$$R^{-1}(k) = \frac{1}{l}\left[R^{-1}(k-1) - \frac{R^{-1}(k-1)\underline{x}(k)\underline{x}^T(k)R^{-1}(k-1)}{l + \underline{x}^T(k)R^{-1}(k-1)\underline{x}(k)}\right] \quad (6)$$

For convenience, a new quantity is defined, $R^{-1}(k)\underline{x}(k)$, which is sometimes called the Kalman gain vector, notice the RLS algorithm and Kalman filtering are very closely linked. The complete RLS algorithm is given by the next three equations

$$\underline{g}(k) = \frac{R^{-1}(k-1)\underline{x}(k)}{l + \underline{x}^T(k)R^{-1}(k-1)\underline{x}(k)} \quad (7)$$

$$R^{-1}(k) = \frac{1}{l}\left[R^{-1}(k-1) - \underline{g}(k)\underline{x}^T(k)R^{-1}(k-1)\right] \quad (8)$$

$$\underline{w}(k+1) = \underline{w}(k) + \underline{g}(k)e(k) \quad (9)$$

To initialise the RLS algorithm the most simple procedure is to use a soft constraint and set $R^{-1}(0)=kI$, that is, a scaled version of the identity matrix [Haykin, 1996]. When the forgetting factor is set to be less than unity the effect of this soft constraint will quickly disappear. The coefficient vector is, as for the LMS algorithm, initialised as the zero vector.

Exercise 1.3. The core of an RLS algorithm has been written in MATLAB (Given in Appendix 1). Use this code to write a routine which can be called from within MATLAB with the following command line [*e*,*wmat*] = rls(*x*,*d*,*X*,*L*).

(i) Compare the MATLAB code with equations (7), (8) and (9). Explain how the MATLAB code implements the algorithm and how the algorithm is initialised. What is the complexity, in terms of number of additions and multiplications, of the RLS routines?

(ii) Repeat Exercises 1.2 (i), (ii) and (iv) for the RLS algorithm with the forgetting factor set at unity.

(iii) Repeat Exercises 1.2 (i), (ii) and (iv) for the RLS algorithm with the forgetting factor set at 0.99, 0.95 and 0.8.

(iv) **Advanced challenge** - How do the LMS, NLMS and RLS algorithms perform if you vary, with sample number, the coefficients of the unknown system? One example would be to fix the unknown system vector for 200 samples and then to change the coefficients at sample number 201 and keep these fixed for the next 200 samples (Read the help information for the command *filter).*

References

Haykin, S., "Adaptive Filter Theory", Third Edition, Prentice-Hall, 1996, *Much improved on first and second editions, good sections on numerical issues in adaptive filtering and emerging adaptive techniques, particularly blind signal processing.*

Widrow, B., and S.D. Steams, "Adaptive Signal Processing", Prentice Hall, 1985, *was in the vanguard of adaptive filtering, introductory but strong on applications.*

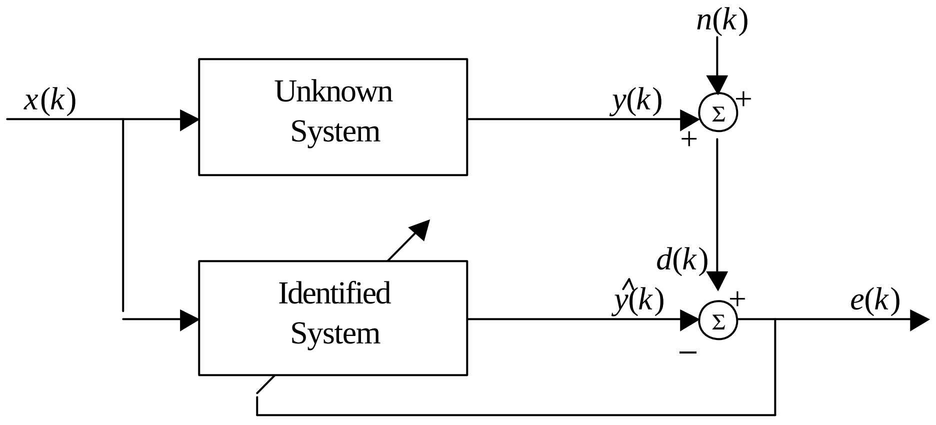
Bellanger, M., "Adaptive Digital Filters and Signal Analysis", Marcel Dekker, 1987. *A first-class book, he has really worked with adaptive filters.*

Appendix 1 Core of RLS Code

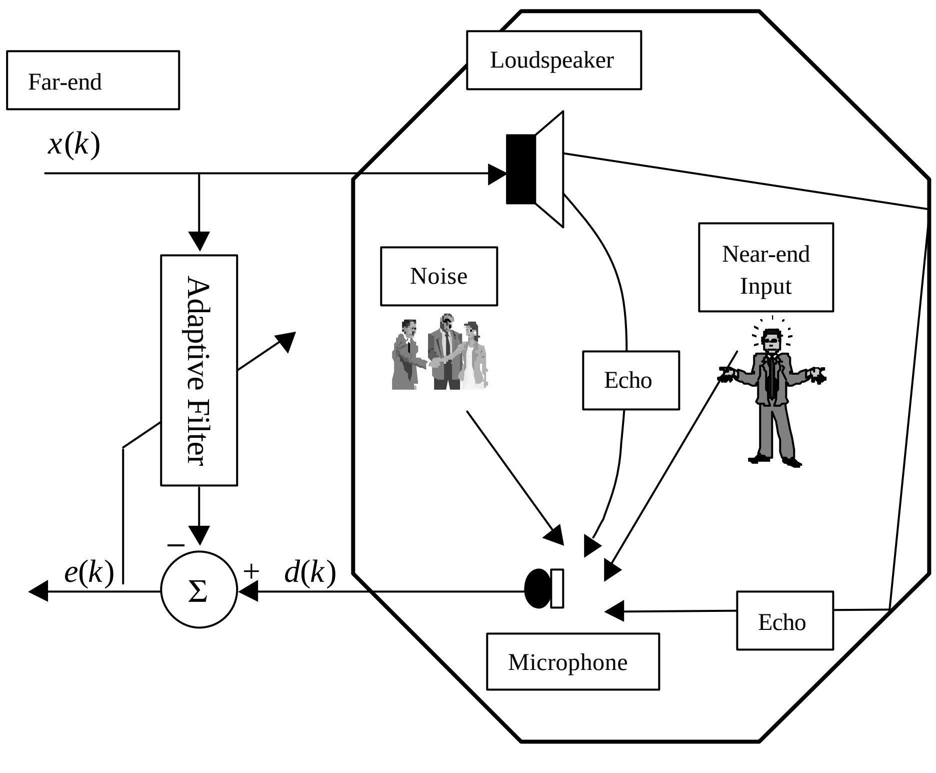
```
w = zeros(L,1);
x = zeros(L,1),
z= zeros(L,1);
rinv = eye(L,L);

for count = 1:npoints
        x(1) = xin(count);
        y=x w,
        e(count) = d(count)-y
        z = rinv*x;
        q = x'*z;
        v = 1/(lambda+q);
        update = e(count)*v;
        w = w+update*z;
        rinv = (rinv – z*z'*v )/lambda;
        x = vrotate(x,1);
end
```

Jonathon Chambers, July 1994, November 1999, Ver. 2.



(a)



(b)

Figure 1. (a) System Identification, (b) Echo Cancellation Application