University of Pennsylvania
Department of Electrical and System Engineering
Digital Signal Processing

ESE531, Spring 2017 Final Project: Audio Equalization Wednesday, Apr. 5

Due: Tuesday, April 25th, 11:59PM

Project Teams: You can work in groups of no more than 2 for this project. You may work alone. Each group must turn in one report with the clear contributions of each member clearly delineated. Everyone is responsible for understanding the design and report in its entirety, and the instructor reserves the right to interview the students to verify this. The teams must be reported to the instructor by April 11, 11:59pm via e-mail: taniak@seas.upenn.edu.

You are to complete all three parts of the handout: Part A, Part B, and Part C. Part D may be turned in for extra credit.

1. **Part A:** MULTI-CHANNEL FIR FILTER-BANK.
   You are to implement a perfect reconstruction filter-bank (PR-FB) using Matlab, and apply it to one-dimensional signals (audio). You can use Matlab routines such as firpr2chfb and mfilt.firdecim, etc., for this purpose.

   (a) Design a 4-channel octave band scheme, in which the full-band is decomposed into 2 low frequency bands (1/8 of full-band each), an intermediate 1/4 of full-band, and the upper 1/2 high-frequency band. Use reasonable parameter values (filter length and band edges for the filters). Filter length should not be unreasonably long.

   (b) To test the filter-bank, use a simple test signal such as a number of sinusoids at different frequencies, or any other test signal of your choice.

   (c) Now use the filter-bank on an audible waveform such as a segment of music or speech. (i) Use the filter-bank to implement a simple equalizer for the speech or music; by this we mean impart different gains on the 4 subband signals before reconstruction, to enhance certain frequency bands relative to others (e.g. bass boost, high-frequency boost, etc). (ii) Find out if introduction of small relative delays in the four subbands prior to reconstruction affects your perception of the audio (i.e. determine to what extent differential phase shifts or delays are important in perception of speech or music).

2. **Part B:** ADAPTIVE NOTCH FILTER.
   DO NOT use high level Matlab commands that may be available in the Signal Processing and other Matlab toolboxes for adaptive filtering in this part. It is easy and much more instructive to write your own Matlab code to implement these.

   A simple real IIR notch filter is a second order filter with two conjugate zeros on the unit circle and two conjugate poles inside the unit circle, with system function

$$H(z) = \frac{(1 - e^{j\omega_0} z^{-1})(1 - e^{-j\omega_0} z^{-1})}{(1 - re^{j\omega_0} z^{-1})(1 - re^{-j\omega_0} z^{-1})} = \frac{1 + az^{-1} + z^{-2}}{1 + raz^{-1} + r^2 z^{-2}} \text{ where } a = -2cos(\omega_0), \text{and } 0 \leq r < 1$$

The notch is at real frequency ω_0 and the closeness of the poles to the unit circle determines the notch sharpness. This filter can be used to reject a strong interfering sinusoid that is contaminating a desired signal. For unknown interfering frequency we want to build an adaptive notch filter, based on the principle that the filter output is minimized when the notch is at the correct location. Note that the adaptation is with respect to the parameter value a, with r generally set at a fixed value. Since this is not an FIR filter, we cannot directly apply the LMS algorithm derived for adaptive FIR filters. However it is possible to develop a simple algorithm for the adaptive notch filter.

We can decompose $H(z)$ as a cascade of the FIR part followed by the all-pole filter, and we can write the output sequence $y[n]$ when input sequence is $x[n]$ in terms of an intermediate sequence $e[n]$:

$$e[n] = x[n] + ax[n-1] - x[n-2]$$
$$y[n] = e[n] - ray[n-1] - r^2 y[n-2]$$

The gradient of $y[n]^2$ with respect to a is $2y[n]\frac{dy[n]}{da}$ where the derivative is not simple; thus, as an approximation we replace $\frac{dy[n]}{da}$ with $\frac{de[n]}{da} = x[n-1]$. To minimize $E(y[n]^2)$, we approximate its gradient as $2y[n]x[n-1]$. The adaptive notch filter update of the parameter a (for fixed choice of r) is therefore: $a[n+1] = a[n] - \mu y[n]x[n-1]$. Since $a = -2cos(\omega_0)$ we have $-2 \leq a < 2$. We should impose this constraint for the updates and reset $a[n] = 0$ if it is out of bounds.

(a) Create a simple simulation of an adaptive notch filter for a desired signal with unwanted additive sinusoidal interference. The desired signal may be some real sequence perhaps with a small amount of additive noise; you can use a single-frequency desired signal as one possibility, but experiment with other types of desired signal also. The interference will be a single strong frequency (low signal-to-interference power ratio). Start with $a = 0$ ($\omega_0 = \frac{\pi}{2}$, mid-point of frequency band). Consider different small values of μ , different fixed values of r (of the order of 0.85 to 0.98), different interfering frequencies and powers. Note that μ will have to be quite small. Give plots and results on frequency response, convergence, spectra, etc. to show how well your adaptive filter works.

(b) Consider one case where the single interfering sinusoid has a frequency that is changing slowly. For example, you might define the interference as a sinusoid with a slowly linearly increasing frequency or some other slowly changing profile. Examine the tracking ability of the adaptive notch filter and give your results and comments. (Note that the instantaneous frequency of a sinusoid $cos(2\pi \cdot \phi(t))$ is $\frac{d\phi}{dt}$)

(c) Can you extend your scheme of part 2(a) to a filter scheme creating two notches to reject two sinusoidal components? In particular, you might consider a cascade of two single-notch second-order notch filters, and adapt the a parameter of each (i.e. $a = a_1$ and $a = a_2$). Explain your approach and give your results for a test case of two interfering sinusoids on a desired signal.

(d) Now investigate use of your 4-channel octave-band filter-bank scheme already designed in **Part A** for adaptive notch filtering applied separately in each subband. Suppose you have interfering sinusoids contaminating a desired signal with the constraint that there can be at most one interfering sinusoid in each of the four octave bands. Show your results for this situation, and comment on the effectiveness of this approach.

3. **Part C:** ADAPTIVE EQUALIZATION *DO NOT use high level Matlab commands that may be available in the Signal Processing and other Matlab toolboxes for adaptive filtering in this part. It is easy and much more instructive to write your own Matlab code to implement these.*

Here you will use adaptive filtering to equalize or invert an unknown channel, with (1) the help of a training sequence and also (2) blindly, without training.

Training Mode Equalization:

Consider a source sending continuous-time pulses of amplitude A or $-A$ to represent bits 1 and 0. The sequence of such pulses passing through a channel can undergo distortion causing them to spread out and overlap with each other, creating what is called inter-symbol interference or ISI. In addition, any front-end filtering at the receiver may cause further pulse spreading.

The channel may be modeled in discrete-time as follows: an input sequence $s[n]$ (random sequence) of $\pm A$ amplitudes passes through a discrete-time LTI system (channel) with unit-sample response sequence $h[n]$ to produce the observed sequence of channel output samples $x[n]$ in the presence of noise. Thus we have $x[n] = h[n] * s[n] + w[n]$ where $w[n]$ is a sequence of zero-mean, independent, Gaussian variates representing additive noise. The channel may be assumed to be an FIR channel of order L (length $L + 1$), and its unit-sample response $h[n]$ is unknown. At the channel output we use an FIR adaptive equalizer filter of larger length $M + 1$ operating on $x[n]$ to try to invert the channel, to ideally get at its output a delayed version of the original input sequence $s[n]$, after the adaptive filter has converged. In order to implement the LMS training algorithm for the equalizer, we need a copy of the actual input sequence for use as the desired (delayed) output signal during an initial training phase. After the training phase, the equalizer should have converged to a good approximation of an inverse filter.

(a) Produce a random ± 1 amplitude training sequence of length 1000. Assume some channel unit-sample response $h[n]$; you may use for example something like $h = [0.3, 1, 0.7, 0.3, 0.2]$ for your initial trials, but you should also test your implementation for different unknown channels of such short lengths (≤ 7). At

the output of the channel (after convolution of input sequence with $h[n]$) add Gaussian noise to simulate a more realistic noisy output condition. You may use an SNR of around 20-35 dB. Now implement an adaptive filter operating on the noisy channel output $x[n]$, using a training sequence which is an appropriately delayed version of the input sequence. The delay will take care of any unknown channel delay (for example, for the channel given above the channel delay may be 2 units,) and equalizer filter delay. Examine the performance you get with different combinations of equalizer filter order M (choose this between 10 and 30), filter step-size μ , adaptive filter initialization, SNR, channel impulse response $h[n]$, etc.

Examine the *impulse* and *frequency* response and *pole-zero* plot of the channel, and the impulse and frequency responses of the equalizer upon convergence, and provide plots and results that show how well your adaptive equalizer works under different channel and noise conditions. Plot also the impulse response and frequency response of the equivalent LTI system between input and output, i.e. the result of the channel and equalizer in cascade, after the equalizer has reached reasonable convergence. Determine if output decisions about the transmitted bits are better after equalization, compared to the un-equalized case. (You can check the output of the system using the equalizer obtained after convergence, by sending several thousand further inputs into the channel.) Comment on your findings.

Blind Equalization:

It is not always possible to have an initial training phase, during which an equalizer at the receiver knows the input, to form an error to drive its LMS training algorithm. The transmitter may be continuously sending data through a channel, and it may be left to the receiver to figure out for itself how to equalize the channel, without the benefit of a known training input sequence.

In the context of the simple scenario of part 3(a), the situation now is that the receiver knows only that the transmitted pulse amplitudes are two-level ± 1 values (more generally $\pm A$ amplitudes). It has to use only this general knowledge about the nature of the input to learn how to equalize the channel. It has no knowledge of an explicit transmitted sequence that it can use for training, and we say it is operating in the blind mode.

A simple approach to blind equalization in this setting is based on the use of the constant- modulus property of the input; the modulus or absolute value of the input amplitude sequence is a constant (= 1 or more generally A). The constant-modulus (CM) blind equalization algorithm attempts to equalize the channel by iterative adjustments to the equalizer with the objective of minimizing a measure of deviation of the equalizer output modulus values from a constant modulus value.

Referring to the LMS adaptive algorithm description, we now have input sequence $x[n]$ to an equalizer that produces output $y[n]$. The error function for the CM blind equalizer is defined as $e_n^2 = \left(|y[n]|^2 - 1\right)^2 = \left(|\boldsymbol{g}_n^T \boldsymbol{x}_n|^2 - 1\right)^2$ where $\boldsymbol{g}_n$ is the equalizer coefficient vector at time n. Differentiating this with respect to $\boldsymbol{g}_n$ we easily find

that the gradient $\frac{de_n^2}{d\boldsymbol{g}_n}$ is proportional to $(|y[n]|^2 - 1)y[n]\boldsymbol{x}_n$. Thus the corresponding stochastic gradient algorithm becomes $\boldsymbol{g}_{n+1} = \boldsymbol{g}_n - \mu(|y[n]|^2 - 1)y[n]\boldsymbol{x}_n$.

(b) Implement this blind adaptive equalizer. Does the blind equalizer converge to a reasonably equalized condition? You will have to try different settings for μ and equalizer order M. (You will need possibly many tens of thousands of iterations, and will need to experiment with rather small value of μ , perhaps of the order of 10^{-4} depending on the specifics of your other parameters.) Use short channel impulse response lengths (around 5) and dont use equalizer lengths that are too long (around 20 maximum). Try different SNRs, but expect poor results if the SNR is not high.

Provide plots and results, and give explanations/comments as in the case of part 2(a) above. Also provide a one-dimensional scatter plot of the output amplitudes after equalizer convergence, to get a visual sense of how well the equalizer is able to bring the output amplitudes close to the desired two amplitudes.

4. **Part D:** FILTER DESIGN USING LMS ALGORITHM
DO NOT use high level Matlab commands that may be available in the Signal Processing and other Matlab toolboxes for adaptive filtering in this part. It is easy and much more instructive to write your own Matlab code to implement these.

Suppose a desired frequency response $H_d(e^{j\omega})$ is specified (with even magnitude and odd phase function). You want to design a real, causal, FIR impulse response $\{h[0], h[1], ..., h[M]\}$ for an M-th order FIR filter which gives a good approximation to this $H_d(e^{j\omega})$. The desired frequency response may not be of a standard type such as a low-pass or band-pass specification. For this we may form a long, real, test input signal sequence $\{x_{in}[n], n = 0, 1, ..., K\}$ consisting of a large number L of individual frequencies spanning the range (0, 0.5) . The corresponding ideal desired output $\{x_d[n], n = 0, 1, ..., K\}$ is obtained using the magnitude scaling and phase shift specified by $H_d(e^{j\omega})$ for each such input frequency. The impulse response coefficients of an FIR filter of order M may then be adaptively learned using the LMS algorithm to get approximately this desired output sequence $x_d[n]$ when driven by the test input sequence $x_{in}[n]$.

(a) Let the desired magnitude response be:

$$|H_d(e^{j2\pi f})| = \begin{cases} 2, & \text{for } 0 \le |f| \le 0.1 \\ 1, & \text{for } 0.1 \le |f| \le 0.3 \\ 3, & \text{for } 0.3 \le |f| \le 0.5 \end{cases}$$

and the FIR filter of order M is to have linear phase.

Form reasonable test input and corresponding desired output sequences, containing an appropriate number L of individual frequencies spread over the (0,0.5) interval. Use the LMS adaptive FIR filter scheme to find a good design for a causal FIR filter of order M that will produce a good approximation of the desired output. Note that you will have to experiment with different combinations of

step-size (this has to be generally small), filter length M, number of test frequencies L, and length of the test sequences N (this will generally be large). Give plots of your designed filter(s) characteristics (impulse response, frequency response). Discuss briefly/comment on any specific aspects of your method or results that you want to highlight.

(b) Now suppose the desired response is

$$|H_d(e^{j2\pi f})| = \begin{cases} j2\pi f, & \text{for } 0 \leq |f| \leq 0.3 \\ 0, & \text{for } 0.3 \leq |f| \leq 0.5 \end{cases}$$

any additional linear phase term is allowed. The real FIR filter of order M approximating this is to have generalized linear phase. Repeat your procedure of part 4(a) for this case. After you have obtained your FIR filter, test it on some interesting inputs.

Project Submission: Your report submission for this project will consist of two parts.

- Project Report:
  You should submit by the due date a single filed report (preferably pdf) explaining what you did and the results you obtained, including figures, test cases, interpretations and comments, as well as responses to any specific questions asked above. Please explain briefly your Matlab code; include a copy of all your Matlab code in your report in an Appendix. The report must be uploaded to Canvas by midnight on the due date.

- MATLAB Code and Other Soft Files: Also submit (upload) by the due date through the Assignments Area on the ESE 531 Canvas Site all supporting material (all your Matlab code files, test input/output files, and any other results files). Ideally all placed in a compressed .zip file. Please make sure you follow the proper procedure for submitting files through Canvas.