Project: ECE 113 Digital Signal Processing
Fall 2021, Prof. G. Pottie

The project has the following components:

1) Manipulation of real impulse response to produce complex baseband representation of a communications channel
2) Creation of zero-forcing and MMSE equalizers when the channel response is known
3) Creation of adaptive MMSE equalizer to deal with both noise and narrow-band interference.

It will make use of the square root raised cosine filter developed as part of the homework. Much of the project will be gone over in detail in Lectures 15, 16, and 17.

Our objective is an equivalent complex baseband system corresponding to carrier frequency fc = 1600 Hz and a symbol rate = 2400 Hz. Recall that we have already created a square root raised cosine filter that applies to a sampling frequency of 9600 Hz that we will be able to use.

1. Complex baseband representation of channel c0.
The reason for a complex baseband representation is that simulations where the main impairments are noise and intersymbol intereference can all be done using vectors, instead of sampling rates sufficient to represent the carrier frequency. This produces a huge reduction in computations when carriers are above the audio band.

c0.txt consists of an integer (the number of samples) followed by 256 samples at 8229 Hz of a channel that was used for characterization of the performance of voice-band modems (e.g., as used even now in dial-up and fax machines). For MATLAB to read it and put the channel samples into a 256-element array M you can use the following commands:

```
M=readmatrix('c0.txt')';
M=M(1:end-2)';
```

As outlined in Lecture 16 you will then perform the following steps:

a. Use spline interpolation using the MATLAB function interp1 to get samples at a rate of 9600 Hz; zero stuff to get to 512 samples
b. Use fft to get the frequency domain representation; plot the absolute values as Figure 1
c. Zero the negative frequencies
d. Cyclically shift so that the carrier frequency of 1600 Hz now sits at sample=1; plot the absolute values as Figure 2
e. Compute a causal square root raised cosine filter corresponding to a cut-off frequency of 2400 Hz with 15% excess bandwidth that is sampled at 9600 Hz (512 samples), with truncation of the impulse response when coefficients are 30 dB down from the peak. Take the fft of a zero-stuffed response (for 512 samples).
f. Multiply in frequency domain. Plot results in Figure 3. Keep this array for further use.
g. Take the IFFT. Downsample by a factor of 4 to 2400 Hz. Plot the absolute values as Figure 4. Compute the sum of the squares of the absolute values (for use in noise normalization)=hpower.

2. Fixed Linear Equalizers

In an actual communication system, the equalizer would be preceded by a fixed SSRC filter to reject out of band noise and to result in no intersymbol interference when paired with the transmit SRRC filter, for a pure AWGN channel. Since we actually have an ISI channel and bandlimited noise, we'll neglect this step in the project. Compute two equalizers:

a. Zero forcing linear equalizer. Take the fft of the 128-point complex channel. Invert it in the frequency domain. Plot the results as Figure 5. Take the IFFT. Convolve with the channel and plot the impulse response as Figure 6. Find the maximum squared absolute value, and compute the signal to interference ratio as SIR=max/(sum of squared absolute values - max). Comment on the delay and degree of interference suppression.
b. Minimum mean squared error linear equalizer. We could in principle compute the Weiner filter via simulation. But much easier is to use the following equations (which take into account that array indexes start at 1 in MATLAB). If h=impulse response from 1 to N, and the noise variance is nvar=1/(sum of squares of channel*2000) so that the signal to noise ratio will be 30 dB assuming input signal power=1,

$$R_{ij} = \sum_{-\infty}^{\infty} h^*(k+1-i)h(k+1-j) + \{\text{nvar, if i=j; 0 otherwise}\}$$

You need to check that no indexes are less than 0 or more than 128 (the length of the filter). You can reduce computations by making the lowest value of k to be 1 and the highest value to be 256. Next create the array hd=zeros(128,1). Then compute hd(i)=h(d+1-i) for all non-negative indices.
The parameter d controls the "main tap" of the filter—at what delay we are making decisions. You will get quite different results as you vary it from 1 to 128. Pick a value that results in the maximum SIR.
The equalizer is then computed as c=R\conj(hd)
Convolve with the channel to compute the SIR. Plot the magnitudes for the fft of c with the best value of d as Figure 7. Compare the results to zero forcing in terms of SIR and delay. Note that the MMSE equalizer attempts to balance reduced noise enhancement against residual ISI, whereas the ZF equalizer may boost noise a large amount when the channel response is small.

3. Adaptive equalizer.
Use the LMS algorithm to train an MMSE equalizer with the same delay d as was optimized in part 2. Begin by creating the filter w=zeros(128,1), and then setting w(d)=1. Set the delta parameter del=2/(128*hpower*1000) so that the final result has little excess noise, where hpower=sum of squares of impulse response. Use the same noise variance as above. Initialize the training data array data=zeros(128,1), and the channel output array u=zeros(128,1). Initialize the squared error=0. Then over a large number of iterations (e.g., a million), do the following

a. Generate complex training data and shift into array data, one symbol each iteration, where the symbols consist of (+/- sqrt(2)+/-j(sqrt(2)), i.e., are complex numbers whose power=1. You can make use of rand, round, and some logic to do this.
b. Compute the channel output as the sum of h(i)*data(i); then add a complex zero mean Gaussian using normrnd(0,sqrt(nvar))
c. Shift the output into u, and compute the equalizer output as the sum of w(i)*u(i).
d. Compute the error E as data(delay)-equalizer output. If we're in the last 5% of iterations, compute squared error=squared error+abs(E)^2
e. Update the equalizer using w=w+E*del*conj(u)

When the iterations are complete, the SINR is given by .05*number of iterations/squared error total. Compare this result to the calculation in 2b. Are we in the same ballpark? Plot the magnitudes of the fft of w against the magnitudes of the fft of c as Figure 8.

Note that to debug the LMS algorithm, it is useful to begin with a toy short channel (e.g., two coefficients, first real, then complex) with short equalizers. The same is true for the equations in 3b. In fact, you can break this down further by checking first that the data generated really is of the form you expect with a very small number of iterations (data alone, noise alone, channel outputs as expected etc.).

Now add an interferer at 1200 Hz with 10x the power of the data sequence. You have previously designed a notch filter with 3 dB points at 1100 Hz and 1300 Hz. We could put this filter in the system before the equalizer. Why would this then present a problem if we are attempting to use a ZF equalizer? (Hint: consider how much suppression it achieves at the notch frequency, and what the ZF equalizer would attempt to do to the notch.) As it turns out, the LMS algorithm can handle this case quite well. But there are a few details to include it in our simulation.

a. In the complex baseband representation will represent the sinusoid is a complex exponential; 1200 Hz gets shifted to -400 Hz. Since the system is sampled at 2400 Hz, this means 6 samples per period. For this to have 10x the power of the data sequence, this implies an amplitude of sqrt(10), multiplied by an exponential with argument jj*pi*n/3, where n is the sample number and jj is sqrt(-1).
b. Use the same for loop as for noise only, adding the sampled sinusoid at step b.
c. Compare the SINR in this case to that achieved with noise alone. Compute the fft of w and compare the magnitudes to the noise-only case as Figure 9. How much suppression is happening in the neighborhood of the interference? Comment on the merits of this approach compared to using a cascade of a notch filter and an equalizer.

It turns out that while the LMS algorithm was originally devised to deal with white Gaussian noise in linear systems, it works quite well with many other impairments, including even mild non-linearities. It is also the basis of back-propagation, used to train neural networks (a highly non-linear situation), although in such cases convergence to a good value is not guaranteed.

Report Format

The report is basically a set of commentaries on the requested figures and computations, where you are comparing the different ways of dealing with channel impairments. You may note alternatives to the methods used (e.g., how the book suggests doing deconvolution vs. the approach to ZF equalization recommended here), and why one might prefer some methods over others in various practical situations. Include your MATLAB code as an appendix.

It is recommended that you do this project with a partner where each first attempts each piece of code and then discusses; this will assist in debugging. Equal credit will go to both partners.