The Cooper Union Department of Electrical Engineering
Prof. Fred L. Fontaine
ECE416 Adaptive Algorithms
Adaptive Equalization
March 15, 2023

Here you will conduct experiments on adaptive equalization using both standard RLS and inverse QRD-RLS. For simplicity, all data will be real.

Indexing can be tricky, so these notes will help you with that. Specifically, these notes align with the MATLAB convention that vectors are indexed starting at 1.

Let us take a simple signal and channel model. The signal $x_n = \pm 1$, iid equiprobably. Note the signal is 0-mean with power $\sigma_x^2 = 1$.

The channel model (input x, output y) is:

$$y(n) = \alpha x(n - N_0 + 1) + x(n - N_0) - \alpha x(N - N_0 - 1) + v(n)$$

where $v(n)$ is 0-mean white noise with variance σ_v^2, and α represents an interference term and N_0 is the nominal channel delay. We will assume in this formula that we use *pre-windowing,* i.e., $x(n) = 0$ for $n \leq 0$, and the output $y(n)$ is computed for $n = 1, 2, 3, \cdots$.

The impulse response of the channel can be written in MATLAB as:

$$h = [zeros(1, N0 - 1), alpha, 1, -alpha];$$

so we can write:

$$y = filter(h, 1, x) + v;$$

The adaptive equalizer will have $M_{\max}$ taps, i.e., the weight vector $w = [w_0, w_1, \cdots, w_{M_{\max}-1}]$, with center tap M_0. Note that M_0 is a parameter we preselect, and will not necessarily match N_0 (which the equalizer does not know). This suggests a good initial tap weight vector with 1 at the w_{M_0} position and 0 otherwise.

Let k be the iteration index for our adaptive algorithm, $1 \leq k \leq N_{\text{iter}}$. The input vectors $\mathbf{u}(k)$ are the column vectors of the $M_{\max} \times N_{\text{iter}}$ data matrix used by the adaptive algorithm:

$$A = \begin{bmatrix} y(M_{\max}) & y(M_{\max}+1) & \cdots & y(N_{\text{iter}} + M_{\max} - 1) \\ \vdots & \vdots & & \vdots \\ y(1) & y(2) & \cdots & y(N_{\text{iter}}) \end{bmatrix}$$

There are variations, e.g., using pre-windowing on y (taking $y(n) = 0$ for $n \leq 0$ and adding additional columns to the left). However, since we plan to run the algorithm for relatively few iterations, we prefer to only feed it data that conforms to the true underlying statistics (avoiding an initial period where some y are artificially set to 0). Using our standard notation for vectors containing shifted samples of a time series signal, the input at iteration k is:

$$\mathbf{u}(k) = \mathbf{y}_{M_{\max}}(M_{\max} + k - 1), \; 1 \leq k \leq N_{\text{iter}}$$

The equalizer is designed to map the received signal at time n, $y(n)$, to the delayed data signal $x(n - M_0)$. At iteration $k = 1$, for example, we line up $y(M_{\max})$ to $x(M_{\max} - M_0)$. Thus, for $1 \leq k \leq N_{\text{iter}}$:

$$d(k) = x(M_{\max} - M_0 + k - 1)$$

After running the algorithm for N_{iter} iterations, let $\mathbf{w}_{\text{f}}$ denote the final tap weight vector. The length of the training sequence (known x sequence) must be sufficient so all the desired signal values are known, i.e., $x(M_{\max} - M_0 + N_{\text{iter}} - 1)$ is that last point we need in the training sequence. The length of the training sequence is:

$$N_{\text{train}} = M_{\max} - M_0 + N_{\text{iter}} - 1$$

and the next point after is $x(N_{\text{train}} + 1)$, which we match up with the first column after A ends:

$$\mathbf{y}_{M_{\max}}(N_{\text{iter}} + M_{\max}) = \mathbf{y}_{M_{\max}}(N_{\text{train}} + 1 + M_0)$$

This is a time shift of M_0 between the most recent received sample $y(n)$ and the output of the equalizer $x(n - M_0)$ we would feed to the decision circuit for decoding. This is by design, as it allows the equalizer to correct for interference from terms on either side (before and after) the target symbol has been transmitted. The system introduces a latency of M_0 to achieve this lookahead.

After the training sequence, we recover the unknown data samples via:

$$x(n) \approx \mathbf{w}_{\text{f}}^H \mathbf{y}_{M_{\max}}(n + M_0) \text{ for } n = N_{\text{train}} + 1, N_{\text{train}} + 2, \cdots$$

Say we want to run this over K time steps. Let us define an estimated vector $x_{\text{est}}(n)$, $1 \leq n \leq K$, where:

$$x(n + N_{\text{train}}) \approx x_{\text{est}}(n) = \mathbf{w}_f^H \mathbf{y}_{M_{\max}}(n + N_{\text{train}} + M_0) \text{ for } n = 1, 2, \cdots, K$$

To summarize:

- Generate a total of $N_{train} + M_0 + K$ points x and v, and filter with h to obtain $N_{\text{train}} + M_0 + K$ points y.
- Run the algorithm for N_{iter} iterations, as described, and pull out the final tap weight vector $\mathbf{w}_{\text{f}}$.
- Compute K values of $x_{\text{est}}(n)$, for $1 \leq n \leq K$.

Assessing Results:
We have $x(n)$ and $y(n)$ indexed for $1 \leq n \leq N_{\text{train}} + M_0 + K$, and $x_{\text{est}}(n)$ indexed for $1 \leq n \leq K$. The first N_{train} points are the training sequence, and used to run the adaptive algorithm. To compare the raw output of the channel y to x, and the output of the equalizer x_{est} to x, for $1 \leq n \leq K$:

$$\begin{aligned} x(n + N_{\text{train}}) &\approx y(n + N_{\text{train}} + N_0) \\ x(n + N_{\text{train}}) &\approx x_{\text{est}}(n) \end{aligned}$$

In each case, we want to calculate the mean-square error between the vectors.

A key parameter in describing the kind of distortion we want to eliminate is *signal-to-noise-plus-interference ratio, SNIR.*

In the above, both y and x_{est} will contain components due to the additive white noise, plus interference terms among adjacent x samples. Since the signal power is 1, the SNIR are as follows:

$$\begin{aligned} SNIR_{\text{raw}} &= -10\log_{10} E\left(|y-x|^2\right) \\ SNIR_{\text{equalized}} &= -10\log_{10} E\left(|x_{\text{est}}-x|^2\right) \end{aligned}$$

where here for simplicity we have omitted the proper time alignments.

To compute the theoretical SNIR, what is often used is the *worst case* interference. Since $|x(n)| = 1$ at each time, applying the filter h yields a target sample $x(n-N_0)$ with interference from adjacent samples with amplitude $\pm\alpha$. Depending on the signs of adjacent x values, the worst case interference would be $(|\alpha| + |-\alpha|)\cdot 1 = 2|\alpha|$, which corresponds to power $4|\alpha|^2$. Thus:

$$SNIR_{\text{theory,raw}} = -10\log_{10}\left(4|\alpha|^2 + \sigma_v^2\right)$$

Finally, even if the interference is perfectly cancelled out, we would expect the best that could happen is a residual noise power due to the white noise term. Thus:

$$SNIR_{\text{optimal}} = -10\log_{10}\sigma_v^2$$

In other words, we could not expect out equalizer to provide a better SNIR.

Experiment:
Work with the following parameters:
$N_0 = 3$, $M_0 = 5$, $M_{\max} = 11$ (it is common to make the "center tap" M_0 near the midpoint of the span of the equalizer).
$N_{\text{iter}} = 20$, $K = 10^4$.
$\sigma_v^2 = 10^{P_{\text{dB}}/10}$ where P_{dB} is noise power in decibels.

Try these cases: $\alpha = 0.1, 0.2, 0.3$ with powers $P_{\text{dB}} = -30$ and -10. You will use two algorithms: RLS and inverse QRD-RLS, $\lambda = 0.9$, $\delta = 0.01$. More details about the algorithms will be described below.

Before computing SNIR, check the first few entries of the x, y and x_{est} vectors and verify that the values seem to align in the way described above. Then compute the SNIRs and comment.

I'm not asking you to run the algorithm many times and average results. However, you should (manually) repeat each permutation of each experiment a few times, and see if results are somewhat consistent. Report just typical values of the SNIR.

Also, check the $\mathbf{w}_f$ and see if there is indeed one term that dominates the others, and check its position in the vector.

Now try one more variation: set $N_{\text{iter}} = 50$ and see if you are getting better results.

About the algorithms

For RLS, write a script as follows:
[w,xi,k,Pfinal]= dorls(d,A,lambda,delta,winit,Niter);
It should return an $M \times Niter$ matrix w, each column are the w tap-weight vectors for $1 \leq n \leq N_{iter}$ (the first column should not be $winit$, the initial tap weight vector; it is the first updated w vector).
Also, xi is the a-priori estimation error, k is a matrix whose columns are the computed Kalman gain vectors, and P_{final} is the final computed $P(k)$ matrix (the inverse covariance matrix).

For inverse QRD-RLS, we want a similar script:
[w,xi,k,gamma,Pch]= doinvqrdrls(d,A,lambda,delta,winit,Niter);
Here, *gamma* is the sequence of conversion factors, and *Pch* is the final value of the Cholesky factor of the P matrix.

When you run your algorithms, the w vectors they generate should be, in theory, identical. Do one test where you check that, and also compare the final *Pch* to the final P (i.e., that *P=Pch*Pch'*; specifically, find the spectral norm of the matrix $P - Pch * Pch'$).

Within the QRD-RLS, you need to map the prearray to the postarray through a unitary transformation, that nominally is a QR decomposition. In MATLAB, if you call:

$$R = qr(A)$$

it will give you the upper triangular R, with $A = QR$, or equivalently, $R = UA$ for some unitary U. However, if you look at the inverse QRD-RLS algorithm, the postarray should be lower triangular, and we should multiply the prearray on the *right.* This suggests:

postarray=(qr(prearray'))';

The problem is the postarray may have negative entries on the diagonal. Since here the data is all real, the fix is with $\pm$ signs (more generally, you need $e^{j\theta}$ factors to cancel). Specifically, you want:

postarray=postarray*D

where D is a diagonal matrix with ± 1 on the diagonal, to make the postarray have positive entries on the diagonal. (Note D is unitary, so we still get the result that **prearray x unitary = postarray**). **Hint:** In MATLAB, for a matrix A, $diag(A)$ is a vector of the diagonal entries, and for a vector v, $diag(v)$ is a matrix with the elements of v on the diagonal. This should allow you to make the sign corrections on the postarray in one line.

As a remark, we are using the inverse QRD-RLS here since it leads to an easier computation of the **w** vector, which is what we want here.