The Cooper Union Department of Electrical Engineering
Prof. Fred L. Fontaine
ECE416 Adaptive Filters
Problem Set IV: RLS, QRD-RLS and Order Recursive Algorithms
July 12, 201c

1. **Orthogonality Principle:** Referencing the Kalman filter notation we used in the course:

$$\begin{aligned} x(n+1) &= Ax(n) + v(n) \\ y(n) &= Cx(n) + w(n) \end{aligned}$$

where v, w are 0-mean white, uncorrelated with each other and the initial state $x(0)$, with covariance matrices Q_v, Q_w, respectively. Although not indicated, in general A, C, Q_v, Q_w may depend on time. With $\mathcal{Y}_n = span\{y(k), 1 \leq k \leq n\}$, the notation $\hat{u}(m|n)$ means the projection of $u(m)$ onto $\mathcal{Y}_n$. The predicted state-error vector is:

$$\varepsilon(n, n-1) = x(n) - \hat{x}(n|n-1)$$

Prove that $\varepsilon(n, n-1) \perp v(n)$ and $\varepsilon(n, n-1) \perp w(n)$ using a BRIEF argument in each case. Your argument must be compact!

2. **RLS:** In the LMS problem set for this course, you implemented LMS for adaptive equalization and adaptive MVDR. Now repeat each using RLS. Use $\lambda = 0.95$ in each case; to select δ you may need some trial and error, but as an initial attempt try $\delta = 0.005$. Basically you need to run the adaptation long enough so the impact of δ is minimal. [This is similar to Haykin 10.10, 10.11 in 5th ed., 9.11,9.12 in 4th ed., except the adaptive equalization and MVDR problems you did for LMS were somewhat different that as they appear in the textbook]

3. **QRD-RLS:** First, write MATLAB code to implement the QRD-RLS and inverse QRD-RLS algorithms (as provided in appropriate tables in Haykin). In this core code, do not assume the data vector $\mathbf{u}(n)$ has any particular structural form (i.e., may not be a time series). Also write "wrapper code" that calls these inner, general algorithms, for the special case where these data vectors are obtained from a time-series, i.e., has the form $u_M(n) = [u(n), u(n-1), \cdots, u(n-M+1)]^T$; normally a prewindowing approach is used (i.e., the assumption is $u(i) = 0$ for $i \leq 0$), however your code should take an OPTIONAL input vector that prescribes initial conditions $u(0)$, $u(-1)$, $\cdots$ (you figure out how far back it needs to go). Also, one algorithm does not directly yield the tap-weight vector $\vec{w}$, the other doesn't directly yield the output error signal $e(n)$. Write code to compute each (don't run this at each time step; what I mean is if you stop the algorithm at some fixed time N, then write code so you can find $\hat{w}(N)$ at that time, or $e(N)$ at that time, respectively).

   Apply this to the same equalization problem you have worked on before. Recall that adaptation happens during a training sequence period, when the ideal transmitted ± 1 sequence is know. Therefore, we don't want to initialize the data matrix with zeros. Instead, consider an initial "prefix" in your training sequence: we transmit the first few symbols, and connect the first few received values, enough for us to fill the data matrix A, and what we call time "$n = 1$" is really the first time we have enough data to fill out the first column of A^H. With this in mind, run the adaptive algorithms.

Specifically, when you run both the QRD-RLS and inverse QRD-RLS, use the SAME set of (random) data, noise. Compute the final tap weight vectors $\hat{w}[N]$, and compare. Do they match exactly?

After the training sequence ends, adaptation ends, and the fixed tap weight vector $\vec{w}$ can be used. Take your $\vec{w}$ vector and test for BER (that means, generate a continued random sequence of $+/-1$ of some reasonable length, say $10,000$, and compute the fraction of errors with a simple decoder). Note that the initial conditions for your vector should not be all 0! The training sequence continues immediately with data used to test the BER on.

Remarks: Do not code up a systolic implementation. Instead use the MATLAB *qrd* function to compute the QR decomposition at each step, as necessary. Note that pre-array to post-array relationship in the algorithm doesn't exactly fit what *qrd* will do for you- there has to be some transposing done!

4. **QRD-RLS Beamformer:** Go back to the adaptive MVDR you had previously done with LMS and RLS. Now do it using both QRD-RLS and inverse QRD-RLS. As above, run each using IDENTICAL random data. Repeat 100 times, and graph the learning curve $J(n)$ (for the method that yields $e(n)$ directly), and the mean-square deviation $\mathcal{D}(n)$ (for the method that yields $\hat{w}(n)$ directly). Also, take ONE instantiation, and compare the FINAL beamformer vector $\vec{w}$ computed in each case, and see if they match.

5. **QRD-LSL:** Table 16.4 (in 5th ed) or 12.5 (in 4th ed) gives the QRD-LSL algorithm with angle-normalized error. Note that you don't need to call *qrd:* the tables give the specific computations you need to find the (2×2) rotation matrices. Be careful with the indexing: that is the most challenging part!

   Normally, it only computes the angle-normalized errors $\varepsilon_m(n)$; we can compute the posterior error output of the adaptive filter $e(n)$ by computing $\gamma_M^{1/2}(n) \cdot \varepsilon_M(n)$ (i.e., at the last stage only). If you look at the algorithm, however, there is a bit of a time offset. So be careful! In any case, I want you to write code as part of your QRD-LSL algorithm to output $e(n)$ (from the LAST stage only).

   Similarly, we could try to extract an overall equivalent tap-weight vector, but let's not do that!

   Take $u(n)$ to be a 3^{rd} order AR process with unit variance innovations, and innovations filter $H(z)$ with poles at $0.95\exp(j2\pi k/3)$ for $k = 0, 1, 2$. Take $w_0 = [1\ 1\ 1]^T$ and construct:

$$d(n) = w_0^H u_M(n) + v(n)$$

   where $v(n)$ is 0-mean unit variance white noise, and $M = 3$.

   (a) Using time averages, find the correlation matrix R for $u_M(n)$. Use this to select a reasonable μ for the LMS algorithm.

(b) Run the LMS algorithm to estimate w_0. Graph the learning curve.

(c) Run the QRD-LSL algorithm. Use $\lambda = 1$ and $\delta = 0.01$. Graph the learning curve. You may need to adjust δ to get reasonable results.

6. **Nonstationarity:** Go back to the above situation. Let N be the number of iterations you ran to get a reasonable convergence with QRD-LSL, based on your learning curve. Now try a modified experiment. Start running your algorithm as before, except after $N/2$ steps switch (instantly) to $w_0^{'} = [-1, -1, -1]^T$, after $N/2$ steps switch back, and repeat. That is, every $N/2$ steps it instantly switches between the two cases. Run the experiment and see what happens. Now do this again, switching every N steps instead. Finally, switch it every $2N$ steps.