Adaptive Filters

(LMS)

Tapped delay line

\[ y(n) = \sum_{k=-M}^{M} c_k x(n-k) \]

We would like to choose \( \{c_k\} \) so that

\[ \left[ z(n) - y(n) \right]^2 \]

is minimized, where \( z(n) \) is the ideal signal.

Initially, let us assume that \( x(n) \) and \( y(n) \) are given (known).
Commonly used algorithms are based on the gradient technique which is derived as follows:

\[ e^2(n) = \left[ x(n) - y(n) \right]^2 \]

\[ = \left[ x(n) - \sum_{k=-M}^{k} c_k x(n-k) \right]^2 \]

Now let’s compute

\[ \frac{de^2(n)}{dc_m} = -2 \left[ x(n) - \sum_{k=-M}^{k} c_k x(n-k) \right] x(n-m) \]

\[ = -2e(n) x(n-m) \]

The gradient Eq. for obtaining the optimum weights (cm’s)
requires that their values be adjusted according to

\[ \frac{d C_m(n)}{d n} = -\mu \frac{d^2 e^2(n)}{d C_m(n)} \]

\[ \frac{d C_m(n)}{d n} = 2 \mu e(n) x(n-m) \]

\[ C_m(n+1) - C_m(n) = 2 \mu e(n) x(n-m) \]

\[ = C_m(n) + 2 \mu e(n) x(n-m) \]

\[ e(n) = \bar{z}(n) - y(n) \]
This is the eye diagram of the waveform when no channel distortion is present:
This the eye diagram of the raw “distorted” waveform at the input of the equalizer:
This is the eye diagram of the equalized waveform at the output of the equalizer.
void LMSFilter(float *x, int *xx, int index)
{
    
    // LMS starts here
    k = 0;
    n = j;
    while (haveMoreSamples)
    {
        y[k] = (float) 0;
        for (i=0; i<filterLength; i++)
        {
            y[k] += c[i] * u[(k+n+i)%24]; // non-causal filter
        }

        if (k%2)
        {
            #define BLIND
            #ifdef BLIND
                if (u[(n+k+filterMidPoint)%24] > (float) 0.5)
                    d = (float) 1;
                else if (u[(n+k+filterMidPoint)%24] <= (float) 0.5)
                    d = (float) 0;
            #else
                if (uu[(n+k+filterMidPoint)%24] > (float) 0.5)
                    d = (float) 1;
                else if (uu[(n+k+filterMidPoint)%24] <= (float) 0.5)
                    d = (float) 0;
            #endif // BLIND

            // error value
            e = d - y[k];
        }

        if (k%2) !(k < 0x1)
        {
        }
}
s = (float) 0;
for (i=0; i<filterLength; i++)
    s += u[(k+n+i)%24] * u[(k+n+i)%24];

mu = (float) 1 / ((float) 100 + s);

// tap update
tap_error = (float) 0;
for (i=0; i<filterLength; i++)
{
    prev_c[i] = c[i];
    c[i] += mu * e * u[(k+n+i)%24];
    tap_error += (float) pow (c[i] - prev_c[i],
                              (float) 2);
}

// Print a subset of filter coefficients
fCoeffFile << (float) tap_error << "\t";
for (i=0; i<filterLength; i++)
    fCoeffFile << (float) c[i] << "\t";
fCoeffFile << endl;

k++;
haveMoreSamples--;
number++;