Nathan Monson

ECE 529

**A Desktop Implementation of LMS and RLS Adaptive Filter Algorithms**

Digital filtering is the process of performing computational operations on discretized, quantized data for the purpose of extracting, enhancing or shaping signal characteristics. Advances in process technology and the optimization of DSP chips have led to the advent of cheap and fast digital filtering of large streams of data. The availability of cheap DSP has revolutionized wireless communications, image processing, sound processing, and array processing, among other applications. Adaptive filters are becoming more and more common as the computational power required to implement them becomes more available. Rather than a filter having fixed coefficients, adaptive filters dynamically optimize coefficients based on state parameters. Common applications of adaptive filters are in system and channel identification and echo and noise cancellation.

In the text below we will demonstrate the implementation of a pair of adaptive filter algorithms. This includes the least-mean-square adaptive filter algorithm and the

**Adaptive Filter Theory**

An adaptive filter can be broken down into two major components as shown in the following figure (from Wikipedia [1]):

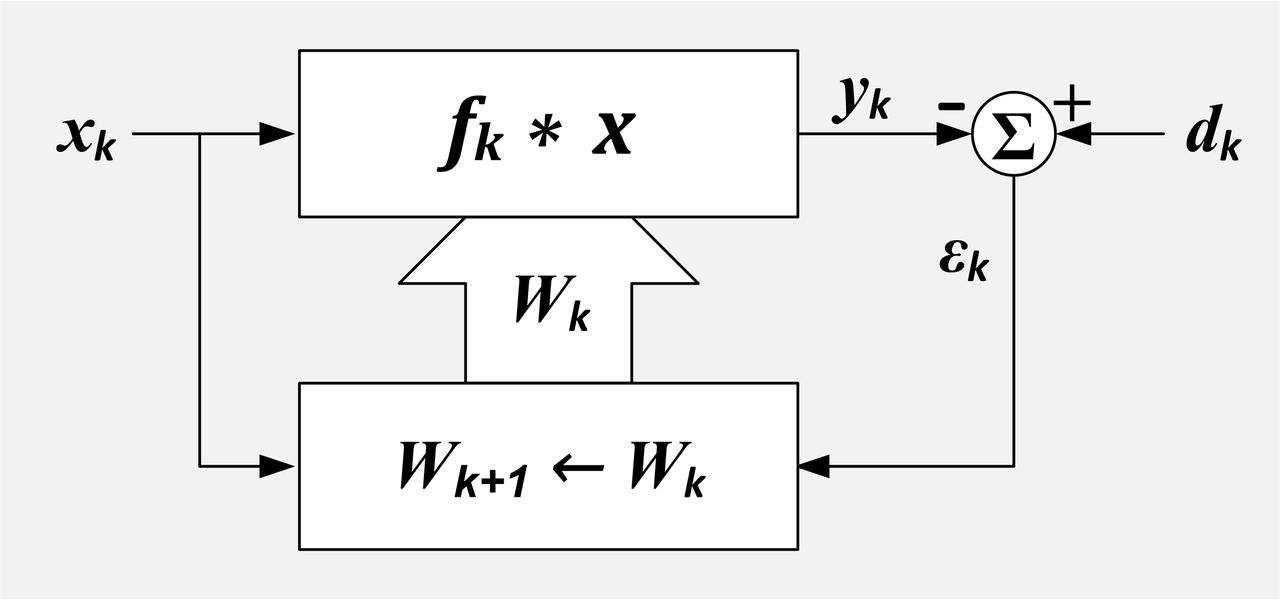


Figure 1. Block diagram for one possible realization of an adaptive filter.

The components are the filtering operation (indicated by the top box) and the optimization algorithm (indicated by the bottom box). As is common to all sorts of filters, not just adaptive filters, the filtering operation can be represented in the time domain as a convolution operation between the filter impulse response and the input delay line from the sampling operation. Adaption comes into play when we include the other component which is the computational block that will update our filter weight coefficients.

In a non-adaptive filter, we determine our desired filter characteristics and design a filter that has a particular transfer function. From the poles and zeroes of this transfer function we are able to determine the various input and feedback terms (for a causal filter) and their associated weights. For applications requiring an adaptive filter, the desired filter characteristics are unknown or are changing and so we calculate the weights dynamically as inputs come in. The specifics of how this dynamic weighting is done will be demonstrated in the computational steps of the adaptive filter algorithms highlighted below.

The distinguishing characteristic of the different types of adaptive filters, as alluded to earlier, is the method by which we calculate filter term weights dynamically. Developing these optimization operations are part of a broad field of research into mathematical optimization. Cost functions or .

* Idea of an optimization function
* Stationary signals converge to Weiner filter
* Differences between two of the main choices for optimization algorithms (LMS/statistical and RLS/deterministic)
  + The optimization function for the LMS is a derives from the negative of the gradient from the error.

In this paper, I will briefly introduce the adaptive filtering theory I will attempt to implement an adaptive filter using the least-mean-square-method (LMS), one of the more common methods to achieve adaptive filtering.

At its core, the LMS filter works by iteratively examining the gradient of the system error. System error in this case being the difference between the actual output signal of a system and the expected output of a system. If the gradient is positive, the error is increasing and so the coefficient weights must be adjusted in the opposite direction of the gradient slope. This is where the name comes from as the optimal solution is one where the LMS of the error is the smallest. The mathematical underpinnings of this algorithm are a probabilistic cost optimization function, the mean-square error cost function. The least-mean-square cost function can be thought of as an instantaneous estimate of the mean-square error cost function (MSE).

**LMS and Weiner**

Gradient of steepest descent

Converging on Weiner solution

**RLS Theory**

S

**Translating Theory into Practice**

As we saw above, computationally intensive operations such as differentiation were reduced to multiplication and accumulation operations in filter realization. This translation means that we can quickly process large datasets with high-order filters.

**Implementation Details**

These adaptive filtering algorithms were implemented in software, in C, on a desktop microprocessor. As the project was coded using Visual Studio 2015, code was compiled using the Microsoft C/C++ compiler. A wrapper main function was used to generate the signals, output files, and call the filtering algorithms. Real-time implementation of the LMS and RLS algorithms may not be possible since I am not using an FPGA.

**LMS Algorithm and Filter Results**

/\* LMS adaptive filter

\*

\* Author: Nathan Monson

\*

\* Parameters:

\* x - most recent input sample

\* w - vector of filter term weights

\* d - value of the desired output at this index

\* mu - the step size (also referred to as the adaptation constant

\* return\_array - results of computatation passed in this array - the filter output value in the first

\* index and the error value in the second index

\*

\* This function implements the three main computational steps of an LMS filter:

\* (1) Update the input\_buffer with the most recent sample and calculate filter output (y) by mult-accumulate

\* the input\_buffer by the weights of the filter terms (w)

\* (2) Calculate the error term by subtracting the desired output value (d) from the calculated output value

\* (3) Update the weights vector (w) by adding the update term at each index

\*/

void lms\_filter(double x, double w[], double d, double mu, double return\_array[])

{

double error = 0.0;

double y = 0.0;

//stores the input(x) values that have previously been fed into the filter, most recent at index 0

static double input\_buffer[FILTER\_ORDER];

for (int i = 0; i < FILTER\_ORDER; i++)

{

input\_buffer[i + 1] = input\_buffer[i]; //shift previous input values

}

input\_buffer[0] = x; //push most recent input value into input\_buffer

for (int j = 0; j < FILTER\_ORDER; j++)

{

y += input\_buffer[j] \* w[j]; // calculate filter output

}

return\_array[0] = y;

error = d - y; // calculate error term

return\_array[1] = error;

for (int k = 0; k < FILTER\_ORDER; k++)

{

w[k] = w[k] + (mu \* error \* input\_buffer[k]); //update filter weight vector

}

}

Below are a collection of plots indicating how the algorithm performs with varied parameters, input signals and desired waveforms. Pay particular attention to the stability of the systems and how long it takes the signal to converge on the desired waveform.

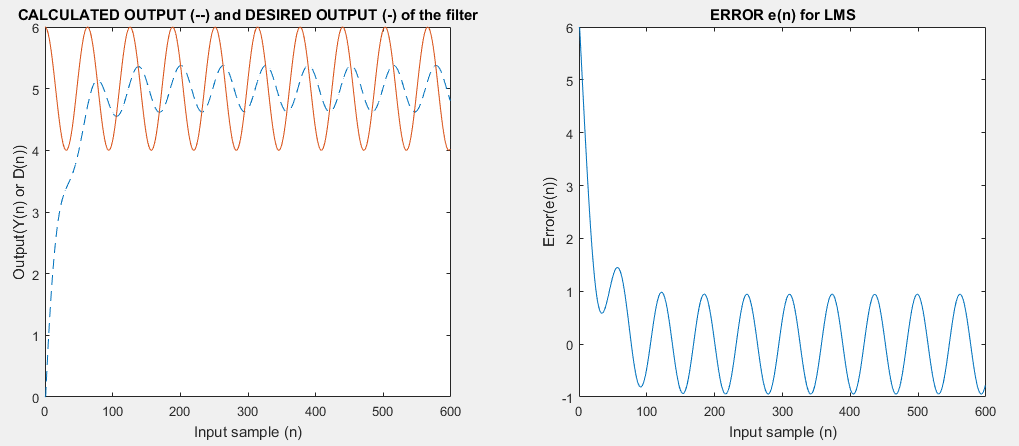


Figure 2. Input waveform (x) = step function with magnitude 1; Desired waveform (d) = cos(0.1n) + 5; Step size (mu) = 0.002; Filter order = 20

This filter is stable takes approximately 100 samples to achieve convergence.

*Reducing the filter order*

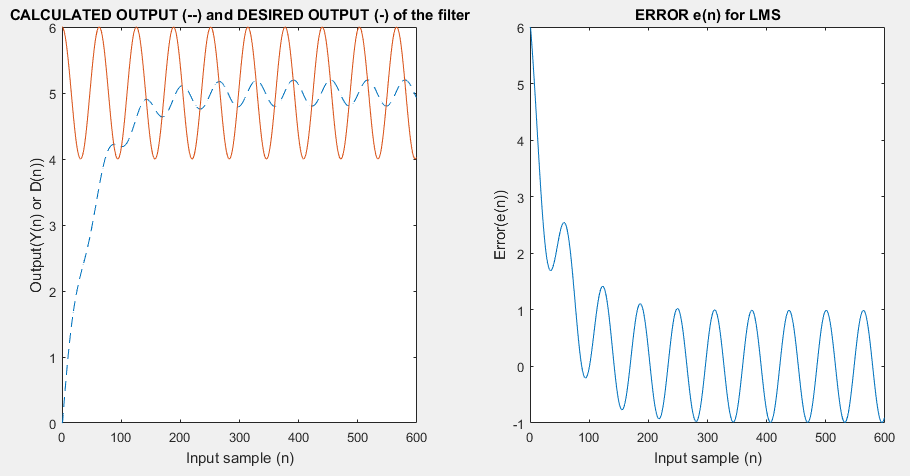


Figure 2. Input waveform (x) = step function with magnitude 1; Desired waveform (d) = cos(0.1n) + 5; Step size (mu) = 0.002; Filter order = 10

This filter of smaller order takes longer to achieve convergence.

**RLS Algorithm and Filter**

**Notes on Performance**

* Algorithms implement sample filtering, whereas a closer to real-time implementation would implement block filtering

**Resources**

Diniz, P.S. Adaptive Filtering: Algorithms and Practical Application. Springer. 2013.

Douglas, S.C. Introduction to Adaptive Filters. Digital Signal Processing Handbook. CRC Press. 1999.

Farhang-Boroujeny, B. Adaptive Filters: Theory and Applications. John Wiley and Sons. 2013.

Haykin, S. and Widrow, B. Least-mean-square Adaptive Filters. Wiley-Interscience. 2003.

National Instruments. Least Mean Square Adaptive Filter. <http://www.ni.com/example/31220/en/>. 8/23/2013.

Wikipedia – “Adaptive Filter”. “https://en.wikipedia.org/wiki/Adaptive\_filter”

Widrow, B. et al. The complex LMS algorithm. Proceedings of the IEEE. 63. 1975. Pp. 719-720.