Ray Mack, W5IFS

17060 Conway Springs Ct, Austin, TX 78717; **w5ifs@arrl.net**

SDR Simplified

A Look at Noise Reduction and Adaptive Filters
(Any sufficiently advanced technology is indistinguishable from magic.—Arthur C. Clark*)*

Statistics and the Nature of Noise

I will give you a little exposure to statistics (in case you haven't used it before) because noise concepts heavily involve statistics. I recommend Chapter 2 of the book *The Scientist and Engineer's Guide to Digital Signal Processing* by Steven Smith for a good starting point or refresher on statistics as well as a good DSP book.[1]

In general, we mean Gaussian white noise when we talk about noise in radio systems. Figure 1 shows the Gaussian distribution and the spectrum (which has a value of 1 from $-\infty$ to ∞). The Gaussian distribution is the bell curve we probably all experienced when our grades were "curved" by our teachers. The sample size in that case was no more than probably 30 samples. In electronics, we have a continuous system with an infinite number of samples of voltage in the system we are measuring. The Gaussian distribution is a very close approximation to what we see in the real world for 99.9997 percent of noise voltages. The equation allows for an extremely low probability of a negative infinite and positive infinite voltage which, of course, will never happen in a real system (at least hopefully not in our lifetimes). Perhaps those occurred during the Big Bang. The 99.9997 percent probability corresponds to ±4.5 standard deviations and is the number used in the incorrectly named 6 Sigma manufacturing goal.

Another important characteristic of white noise is that it is uncorrelated. This is impor-

[1]Notes appear on page 46.

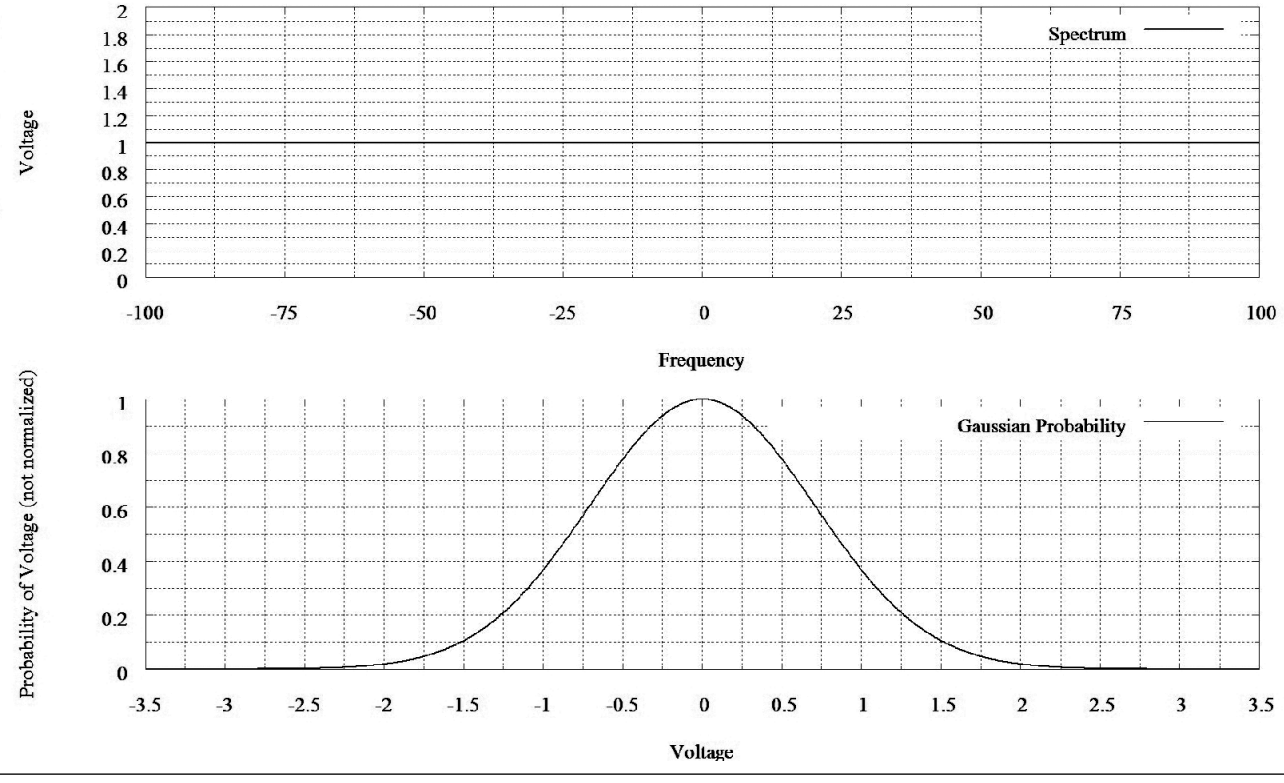

Figure 1 – (Bottom) A plot of the Gaussian probability distribution. (Top) The spectrum that corresponds to Gaussian white noise from -100MHz to 100 MHz.

tant if we implement an algorithm that acts to correlate an input signal. When I first started studying noise, I had trouble understanding the probability shape and the frequency spectrum. The Gaussian shape describes the ***probability*** function of any particular voltage at any particular time. However, noise is a totally random process, so it is reasonable for the voltage to change very quickly or very slowly as well as change from a large negative value to a large positive value. These describe the voltage as a ***time*** function. Remember that very quick changes in time translate into very broad frequency spectrum. The dirac delta function has an infinite frequency spectrum with a constant value. Random noise actually contains signals that come very close to random instances of very small dirac delta functions.

The Moving Average Filter

Smith observed that the moving average filter is frequently the first choice by engineers looking to reduce noise because of its simplicity. The moving average filter is optimal when one wants to reduce noise in time encoded signals. It is also one of the worst filters for frequency encoded signals. The moving average filter excels at reducing noise in signals where we need to keep the sharpest step response. Two radio examples that can benefit from the moving average filter are receiver AGC and transmitter ALC. Receiver AGC is an excellent example where we are interested in detecting the edge of a step in signal level. Figure 2 shows an example of a 100 mV square wave such as an AGC input with 25 mV_{P-P} white noise (top plot) and the results of passing it through an eight element moving average filter (bottom plot) or four element filter (middle plot). The rise times of the edges are lengthened but the edges remain very sharp. It is interesting how much noise even a four element filter will remove. A moving average filter is actually a special case of FIR filter where each tap has the value 1/N with N being the number of taps.

The moving average filter is especially easy to implement in *C* code because all of the values have equal weight. That means that we do not care about the order we add each value. Listing one shows two different implementations in *C*. The first implements a circular buffer of arbitrary size that replaces the oldest sample with the newest sample and then computes the average. The second example illustrates improved efficiency of execution if the circular buffer size is a power of two.

The Matched Filter

The matched filter is another optimal time domain signal noise reduction technique. If you know the exact wave shape of the desired signal ahead of time, you can use the cross correlation function to determine when that signal occurs. A matched filter looks very similar to an FIR filter in implementation. An FIR filter works by performing a convolution of the input time samples with the impulse response of the desired frequency response. A cross correlation filter also passes the time samples through the filter, but the values in the filter are an exact replica of the desired signal. Figure 3 shows the damped sinusoid expected waveform that we used for an implantable defibrillator using magnetic pulses with pulse position modulation for encoding. The top graph shows the sample values used for the matched filter calculation. Figure 4 shows the expected signal with noise and the output of the correlation machine. The signal plus noise contains random noise (not really white noise) and two pulses. One pulse is a positive version of our signal that begins at sample 23 and the second is a negative version that begins at sample 57. The noise is roughly 25 mV_{P-P} and the signals are also 25 mV_{P-P}. You really have to use a lot

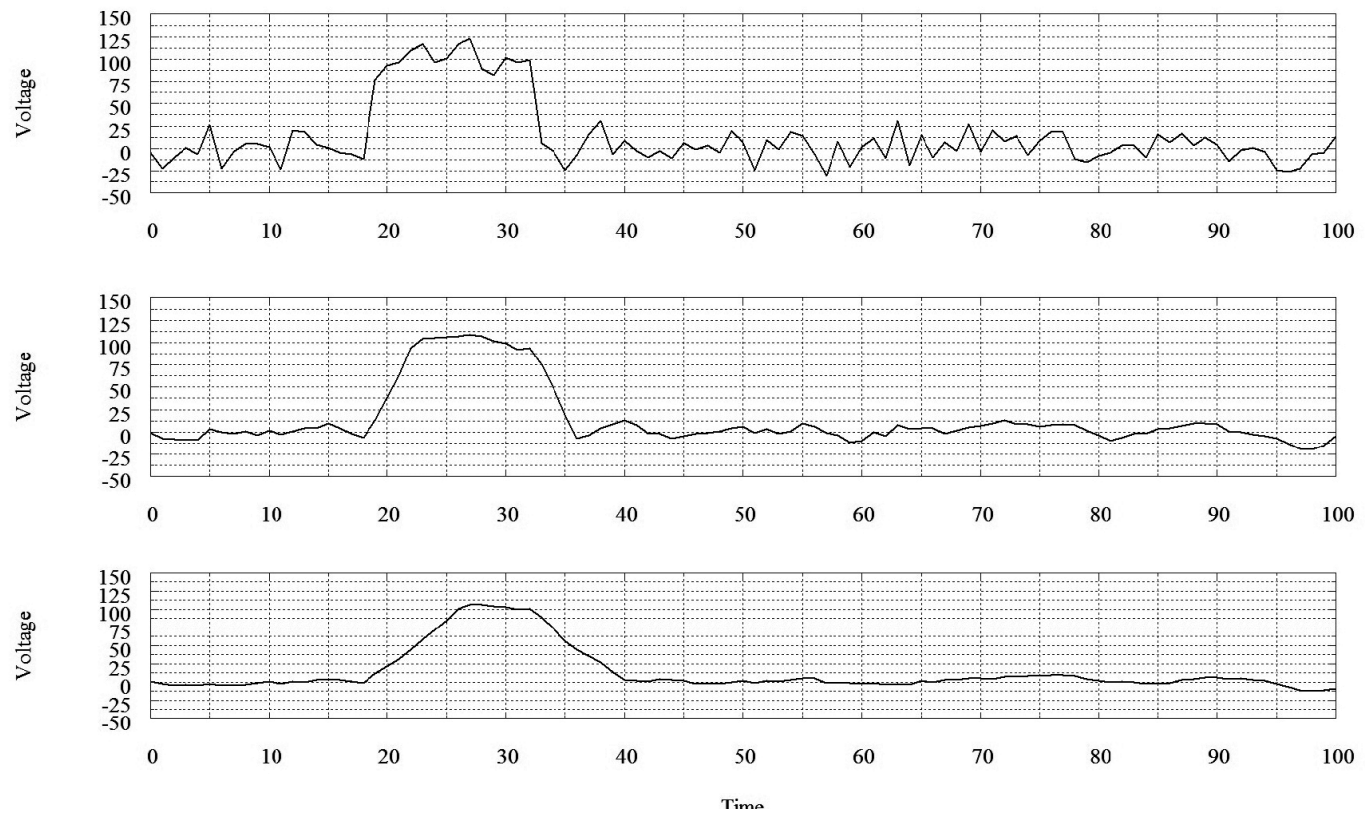

Figure 2 – (Top) A plot of an example AGC square pulse contained in random noise. (Bottom) The output of an 8 tap moving average filter. (Middle) The output from a four tap moving average filter.

Listing 1

Two Moving Average Filters

```
#define NUMBER_OF_SAMPLES  11@@// an arbitrary number that fits
                                // the number of samples we want
                                // to average
@
int@sample_store[NUMBER_OF_SAMPLES][ @
int next_index;

int arbitrary_moving_average(int new value)
{
unsigned int i;
int          accumulator;

    // store the new sample in the ring buffer
    sample_store[next_index] = new_value;
    // set up the pointer to the next entry in the ring buffer
    next_index++;
    if (next_index ==  NUMBER_OF_SAMPLES)
    {
      next_index = 0;
    }
    // we do not care about the sample order
    // because of the commutative property of addition
    accumulator = 0;
    for (i=0; i < NUMBER_OF_SAMPLES; i++)
    {
      accumulator += sample_store[i];
    }
    // the function result is the new moving average value
    return (accumulator / NUMBER_OF_SAMPLES);
}

#define ARRAY_SIZE   8       // This value must be a power of two
                              // for the logic to work
#define MASK         0x7     // the bit pattern to mask for the array
                              // size
#define BITS_IN_SIZE 3       // the number of bits that correspond to
                              // the exponent of the array size
                              // 8 == 2**3
int binary_moving_average(int new value)
{
unsigned int i;
int          accumulator;

    // store the new sample in the ring buffer
    sample_store[next_index] = new_value;
    // set up the pointer to the next entry in the ring buffer
    // we save several instructions because the bit masking
    // replaces a compare and several load instructions
    next_index++;
    next_index &= MASK;
    // we do not care about the sample order
    // because of the commutative property of addition
    accumulator = 0;
    for (i=0; i < ARRAY_SIZE; i++)
    {
      accumulator += sample_store[i];
    }
    // the function result is the new moving average value
    // a shift operation takes an order of magnitude fewer
    // CPU cycles compared to an arbitrary divide
    return (accumulator >> BITS_IN_SIZE);
}
```

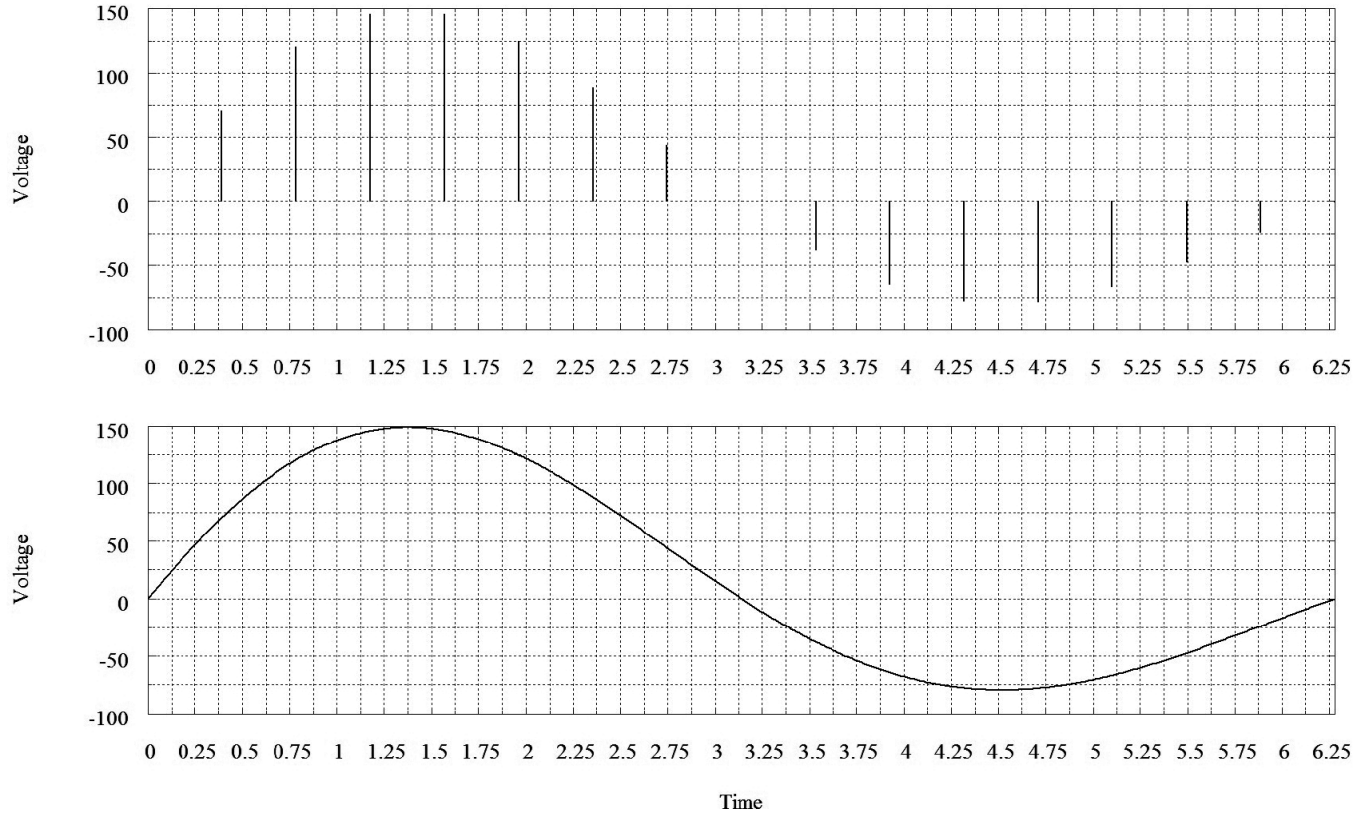

Figure 3 – (Bottom) The waveform used for our matched filter example. (Top) The sampled version of the waveform used in the matched filter.

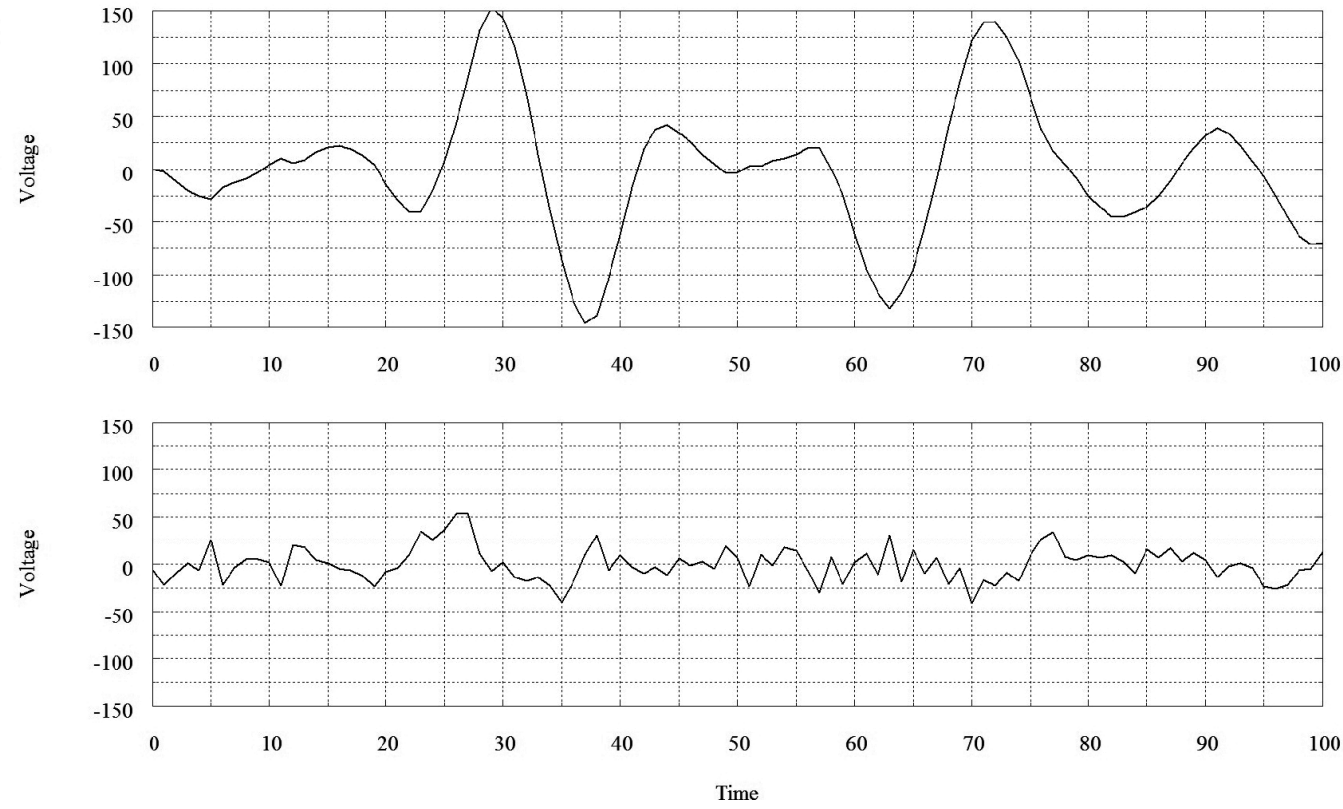

Figure 4 – (Bottom) Two matched filter pulses buried in noise. The first pulse is a positive version and the second pulse is an inverted version. (Top) The output of the matched filter showing two real pulses and a third pulse that is actually noise.

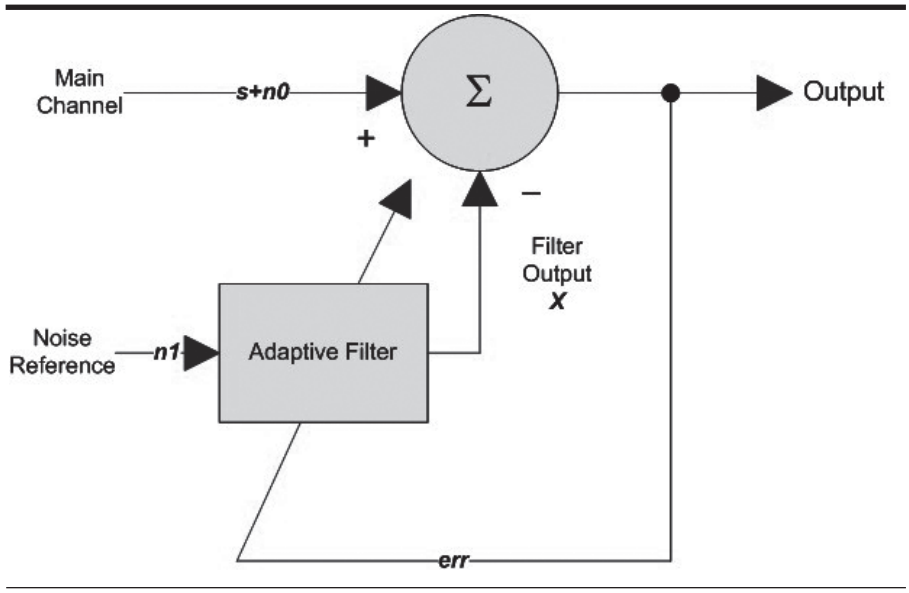

Figure 5 – A block diagram of an adaptive noise cancelling system..

Listing 2

An LMS Example

```
#define NUMBER_OF_TAPS      64
int   LMS_weights[NUMBER_OF_TAPS];
int   sample_data[NUMBER_OF_TAPS];

int   newest_sample;

void LMS_weight_adjuster(int new_sample, int error, int gain)
{
int   i, j, k;

   sample_data[newest_sample] = new_sample;
   // our pointer to input data in its circular buffer
   i = newest_sample;
   // our pointer into the weight array
   j = 0;
   // going backwards through an array is
   // an excellent example of where a do- while
   // construct works best
   do
   {
      LMS_weights[j] = LMS_weights[j] + (2 * error * gain * sample_data[i]);
      j++;
      i--;
      if (i < 0)
      {
         i = NUMBER_OF_TAPS - 1;
      }
   } while (i != newest_sample);
   // point to the next position in the data circular buffer
   newest_sample++;
   if (newest_sample == NUMBER_OF_TAPS)
   {
      newest_sample = 0;
   }
}
```

of imagination to see the two pulses among the noise. We get two output pulses that look like a single cycle of a sine wave. This is a consequence of our signal being a damped sine. Other signal shapes would have other shapes. We identify our two pulses because there is a positive peak followed by a roughly equal negative peak or a negative peak followed by a positive peak. The peaks of the output will always be 8 samples apart for this particular waveform. There is noise in the output since even noise will have some resemblance to the desired signal. In this particular sequence, the noise very much looks like our signal beginning around sample 83, but the signal level is not enough to declare a pulse if we require the output to be above 50. The beauty is that the correlation greatly enhances the detection of the signal compared to the noise. The code for this issue contains an *Excel* spreadsheet where I have implemented the moving average filter and the matched filter.[2]

The Adaptive Transversal Filter

An FIR filter is a transversal filter because the signal traverses the filter from input to output in a serial fashion. An FIR filter is a fixed function, however. There are many algorithms that can be used to modify the filter coefficients in order to accomplish varying filter performance as the signal environment varies; it adapts to the changing environment. We normally encounter two such scenarios in amateur radio. The first is adaptive notching of heterodynes or CW signals in the pass band of an SSB signal. The other is reduction of white noise on an SSB or CW signal. In the second case, the desired signal actually occupies small numbers of bins in the received spectrum and the signals are highly correlated where the noise is totally uncorrelated to the desired signal. My first DSP system to implement these functions

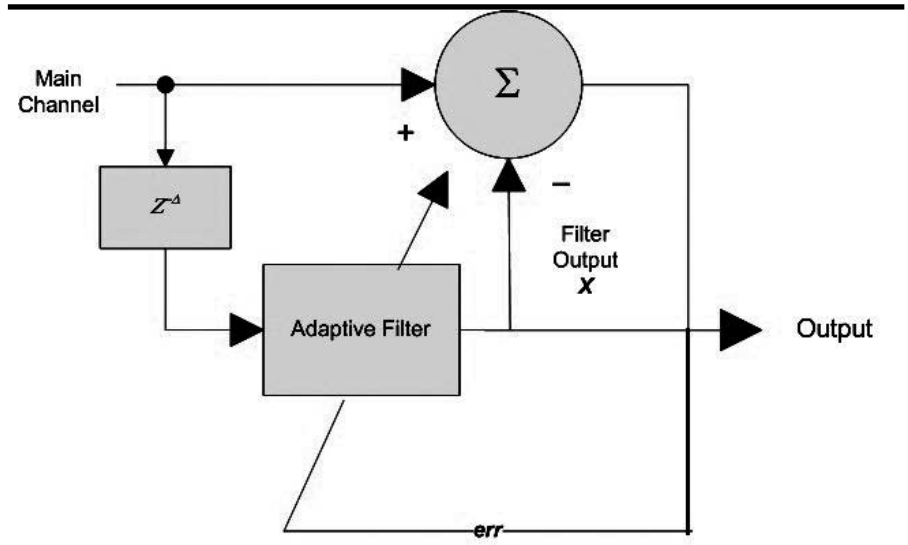

Figure 6 – A modified adaptive system using the input signal for both the main channel and the noise reference channel.

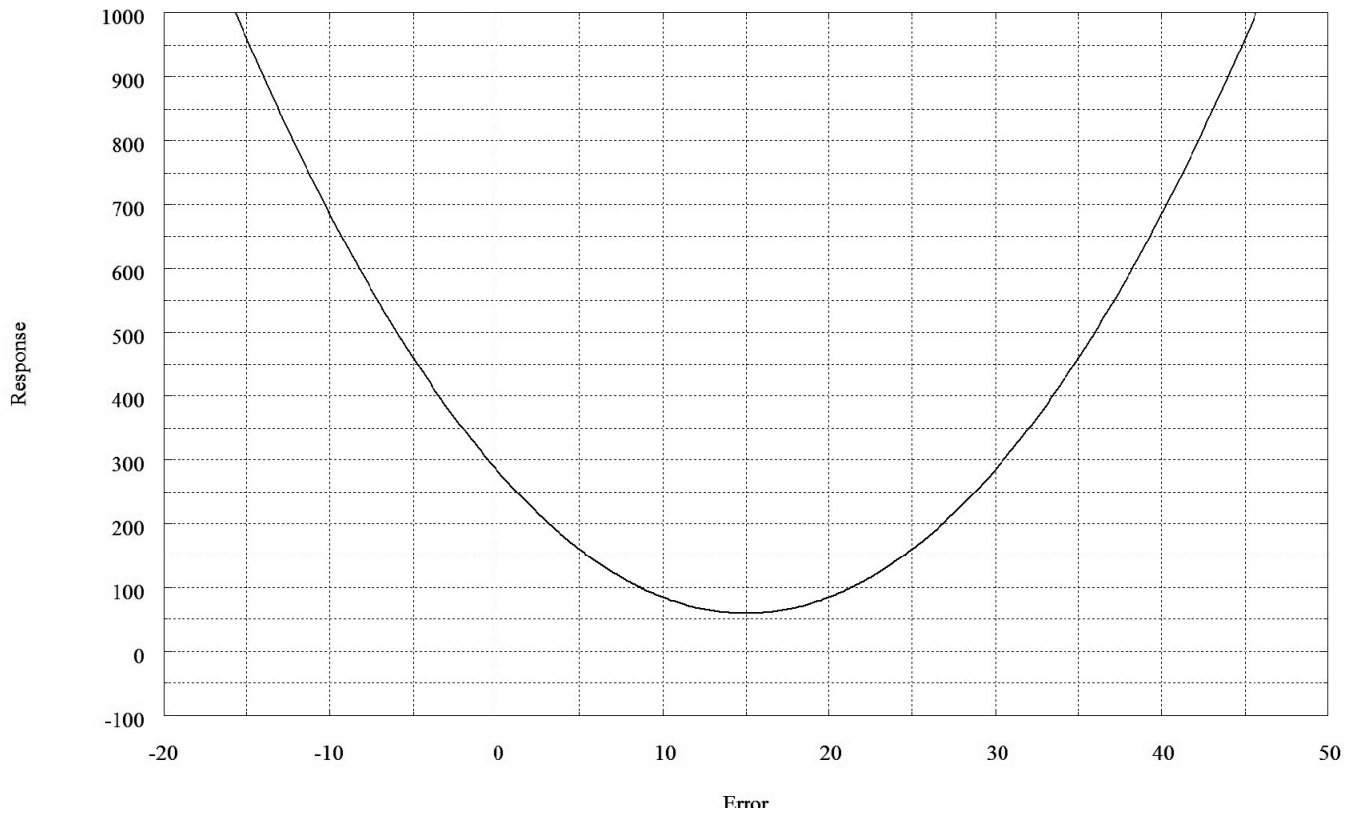

Figure 7 – A plot of a Quadratic response function. The LMS function has a response that corresponds to this type of response.

was the RadioShack DSP-40. Timewave and others make newer systems that are even better at adaptive notching and white noise reduction. Both functions are very easy to implement at baseband because the noise is confined to our audio bandwidth and we can sample fast enough to have narrow frequency bins.

Figure 5 shows one method for adaptive noise reduction. We have a signal plus noise input from the main antenna plus an auxiliary noise antenna feeding the system. The noise at the main and auxiliary channels are related, but may vary over time so that a fixed filter implementation will not be optimum. The noise channel is applied to an adaptive filter that creates a second signal that is subtracted from the main channel. If the adaptive filter is perfect, the control signal will exactly match the noise contained in the main channel and the output will be the desired signal with no noise. This is rarely (if ever) possible, but our goal is to reduce as much as possible the noise in the output signal.

This is where the math gets messy again. We start by assuming that s, n_0, n_1, and x are statistically stationary, meaning the mean and standard deviation do not change with time, and that they have a zero mean value. We also assume that s is uncorrelated with n_0 and n_1, but n_0 and n_1 are correlated. These are pretty safe assumptions in the real world. It is way beyond our scope to describe why, but the expected value of the product of two signals that are uncorrelated with zero mean is equal to zero. Likewise, the expected value of the square of any signal is the square of its value (auto-correlation). The output of our system is:

$$err = s + n_0 - x$$

Squaring (which gives us power):

$$err^2 = s^2 + (n_0 - x)^2 + 2s(n_0 - x)$$

Now we take the expected value of all of the elements:

$$E[err^2] = E[s^2] + E[(n_0 - x)^2] + 2E[s(n_0 - x)]$$
$$E[err^2] = E[s^2] + E[(n_0 - x)^2]$$

The last term falls out because the two signals are not correlated. (Yes, it seems like magic to me too.) If we adjust our filter to minimize the power in the error signal (which is also our output signal) we will maximize the signal power in relation to the noise power. We do not accidentally reduce the signal, s, instead of the noise (which would also reduce the total power) because the signal, x, is derived only from the noise that is present in both channels.

A voice signal is actually composed of a fundamental frequency plus harmonics with some small sidebands around each of the fundamental plus harmonics. If we could create a filter that was a series of band pass filters that passed only the fundamental and harmonics while attenuating all of the other frequencies in the input spectrum, we could get rid of all of the noise power at those other frequencies. We would still have the noise power that falls in the same bins as our voice signal, but the potential improvement in signal to noise is huge. This is exactly what an adaptive noise canceller does. It constantly tracks the changing characteristics of the voice signal (which, for the purposes of tracking, change rather slowly) and implements just the number of narrow band pass filters to pass the voice and eliminate the noise.

What do we do if we do not have a viable reference noise channel? Figure 6 shows a modification of the system of Figure 5. Our input consists of a correlated signal and an uncorrelated broadband component (white noise in our receiver systems). We place a delay between the input channel and the reference channel sufficient to make the signal in the input uncorrelated from the reference. The information in the reference channel remains correlated with the main channel. The result is that we attempt to minimize the correlated signal power in the error signal. The output is taken from the output of the adaptive filter rather than from the adder.

The separation of the correlated signal such as a constant sine wave from an uncorrelated signal can also be used as an adaptive notch filter. Again, by placing a significant delay between the reference and the input channels, we can take a signal such as voice that becomes less correlated as the time difference increases and create a noise signal which is basically uncorrelated. The sine wave interference has almost perfect correlation even after significant delay so the adaptive noise canceller will create an adaptive notch at the frequency.

The LMS (Least Mean Squared) Algorithm

Figure 7 shows a plot of a quadratic equation (one having squared terms). We created a function with the same general shape when we squared the error signal in Figure 5. The square of the error signal will always fall on the line and our goal is to adjust the filter values so that we find the bottom of the curve. The equation for a real system is actually a three dimensional surface, but we will simplify it to be a simple curve in one plane. All adaptive algorithms look at where we were with the last error estimate and where we are with this estimate. The goal is to always have the difference move in a negative direction on the curve. The Least Mean Squared (LMS) algorithm is constrained to work with a transversal filter, so it is not a general algorithm for all systems. The LMS algorithm is probably the most used adaptive technique for electronic DSP implementations because DSP processors are designed to implement transversal filters. Figure 8 shows an implementation of an adaptive linear combiner. It is just a normal FIR

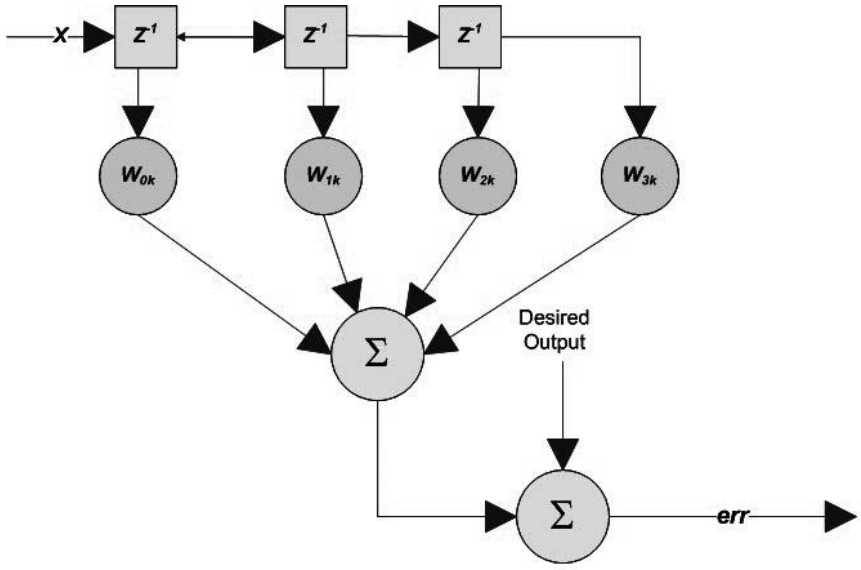

Figure 8 – A block diagram of the LMS algorithm implementation.

filter but instead of each tap value being a filter coefficient, it is an adaptive weight value. The math guys call this a weight vector. The math gets really messy with vectors and matrices and such, but as with a lot of DSP, we can ignore the math and go right to the implementation. We need just a little math to describe the LMS equation, though. We call the set of weights a vector where $\boldsymbol{W}_k$ is just $[W_{0k},W_{1k},W_{2k},W_{3k},\ldots]$ from Figure 8. Likewise, the input signal vector X_k is just $[X_{0k},X_{1k},X_{2k},X_{3k},\ldots]$. In an FIR filter, the coefficients are static so $\boldsymbol{W}_{k+1}$ (the values at z^{-1}) would be exactly identical to $\boldsymbol{W}_k$. That is not the case in an adaptive filter. The adaptive process first calculates the output of the filter and then replaces all of the weight values every time we add a new data sample to the filter. In essence we have two filter processes going on in parallel. Fortunately, the math is actually quite simple:

$$W_{k+1} = W_k + 2\mu\varepsilon_k X_k$$

where μ is just a constant gain value and ε_k the single data value that is the output of the error calculation. The software simply walks down the present set of weight values and data values and does two multiply operations and one addition per element to create the new set of weights. Listing 2 shows how simple the process is in *C*. The listing is a representative expression of the algorithm. An actual implementation is included in the source code for this issue, but can also be downloaded as part of version 1.8.0 of *PowerSDR*.[3] The file *lmadf.c* contains all of the code used in *PowerSDR*.

Reader Feedback

Peter Traneus Anderson wrote the first article about digital down conversion for *QEX* in 1996.[4] He reminded me that one must be careful when looking at the commercial digital down converter chips that are designed for broadband digital applications. Digital may work just fine at 100 dB spurious free dynamic range (SFDR), but narrow band applications frequently need 120 dB to 160 dB SFDR. Commercial applications have improved in the past 17 years, though, and some of the commercial chips should do quite well in narrow band applications.

Gary Heckman, KC7FHP, has been following our work for some time and did some work on his own to understand the Hilbert Transform. He implemented the algorithm in *MS-BASIC/QBASIC* and had good results. I have included his e-mail to me (which includes the *BASIC* source code) in the ZIP file for this issue.

More Reading

I recommend two books that I used as reference for this installment. I have already mentioned the book by Smith.[1] It is very well written and uses a conversational style. He also tries to keep the ugly math to a minimum. I also used *Adaptive Signal Processing* by Bernard Widrow and Samuel Stearns.[5] This book is full of really nasty math, so it will not help you much unless you can suffer through sophomore level engineering math. It is interesting that Widrow has a couple of equations named after him and that a lot of the earliest work in adaptive DSP signal processing only goes back to the 1960s. This is a very new area of study and products like the DSP-40 were available very shortly after the concepts were developed in academia and Bell Labs.

Notes

[1]Steven Smith, *The Scientist and Engineer's Guide to Digital Signal Processing*, Prentice-Hall, 1997.

[2]**www.arrl.org/qexfiles**. Look for *SDR Simplified 3-2013.zip*.

[3]**support.flexradio.com/Downloads.aspx?fr=1**

[4]"A Better and Simpler A/D for the DDC-Based Receiver", by Peter Traneus Anderson, KC1HR, *QEX*, August 1996, pages 21 to 24.

[5]Bernard Widrow and Samuel Stearns, *Adaptive Signal Processing*, Prentice-Hall,