Impulse response measurements using MLS

by

Jens Hee
https://jenshee.dk

August 2003

Change log

20. May 2003

1. Document started.

11. July 2003

1. First version released.

30. July 2003

1. The section about primitive polynomials is replaced by a section about finite fields.
2. A program example is added.

1. August 2003

1. Extension of reordering example in section ”Deconvolution step by step”.
2. Bibliography added.

6. August 2003

1. Formula for deconvolution is corrected for DC offset.

7. March 2020

1. Meta data added.

Contents

1 Introduction **1**
1.1 What is MLS? 1
1.2 Measurement techniques 1

2 The Maximum Length Sequence **3**
2.1 Finite fields 3
2.2 MLS by recursion 4
2.3 Autocorrelation function and crest factor 4

3 Computing the impulse response **5**
3.1 Deconvolution 5
3.2 The M-sequence matrix 6
3.3 Decomposition of the M-sequence matrix 6
3.4 Hadamard matrix equivalence 7
3.5 Deconvolution step by step 7
3.6 Program example 9

Chapter 1

Introduction

1.1 What is MLS?

MLS (Maximum length sequence) for signal processing purposes is the name of a pseudo random signal consisting of the values 1 and -1. It is periodic with the period $P = 2^N - 1$ and has a flat frequency response (see Chapter 2).
MLS is also the name of a computational cost effective technique for measuring the impulse response of a linear system using the MLS signal as an input to the system (see Chapter 3).

1.2 Measurement techniques

The impulse response and the frequency response of a system are the Fourier transforms of each other. It is therefore only necessary to measure one of them, then the other one is known. In the following some of the more common techniques are briefly reviewed.

Impulse excitation

A strait forward method is to use an impulse as the excitation signal. The output from the system is then by definition the impulse response. The draw back of this method is that the impulse must be short enough to have a flat frequency response over the frequency range of interest and at the same time produce enough energy to give a reasonable signal to noise ratio without overloading the system. The signal to noise ratio can be improved by averaging, but the measurement time will increase correspondingly.

Linear sweep excitation

If a linear sweep is used for excitation, the frequency response of the system under consideration can be obtained by mixing the output of the system with the applied sweep signal followed by a low-pass filtering. In practice the method is more complicated than indicated, but the details are not given here. Since a sweep is used for excitation a high signal to noise ratio is obtained. Lowering the sweep rate can increase the S/N ratio. The method can also be used for measuring the distortion of non-linear systems.

Stepped sine excitation

Stepped sine analysis is carried out similar to the linear sweep method except that the excitation signal is stepped through a sequence of frequencies. Normally this method is used to obtain a logarithmic frequency axis and is not suitable for measuring the impulse response.

MLS excitation

The Impulse response of a system can be measured by applying an MLS signal and process the output by a modified Hadamard transform. The details are given below. The method has several advantages. The processing is simple (although theoretically complex), which can be important for small computer systems. The method gives a good S/N ratio due to the low crest factor of the MLS signal.

Two-channel FFT

The previous methods all use a particular excitation signal. It is also assumed that the measurement channel has a linear amplitude and phase characteristic. If both input and output of the system are measured and processed using FFT, the frequency response of the system can be obtained using almost any input signal and only amplitude and phase match between channels is required. The method is computational more complex than the MLS method, but gives a high degree of freedom for selecting the excitation signal.

Chapter 2

The Maximum Length Sequence

As mentioned above an MLS signal is a pseudo random periodic sequence. It can be generated by a one-bit linear feedback shift-register whose characteristic polynomial is primitive (see below), but for signal processing, the binary states of 0 and 1 are mapped into ± 1. The theory about primitive polynomials is part of the rather extensive theory about finite fields also called Gallois fields. The next section gives some facts from this theory.

2.1 Finite fields

A finite field is a finite set where addition, subtraction, multiplication and division are defined. Finite fields exist only when the number of elements are p^N where p is a prime and N any positive integer. Only fields with $p = 2$ are of interest for MLS signals.
It can be shown that the elements in such a field can be represented by the 2^N polynomials of degree $\leq N - 1$ with coefficients 0 and 1 and the multiplication and division is defined modulo an irreducible polynomial of degree N. When adding and subtracting coefficients the operation is modulo-2.
For $N = 3$ an irreducible polynomial is $x^3 + x + 1$ because neither 0 nor 1 are roots. The elements in the field are:

0	000
1	001
x	010
x^2	100
$x+1$	011
x^2+x	110
x^2+x+1	111
x^2+1	101

It is seen that apart from the first element each element is equal to the previous one multiplied by x. When an irreducible polynomial has this property it is called a primitive polynomial and x is called a primitive element or generating element. For $N = 3$ both the irreducible polynomials are primitive but for $N = 4$ only two of the three irreducible polynomials are primitive.
It is not simple to find the primitive polynomials, but several textbooks give lists of primitive polynomials up to a very high degree. Some of them can also be recovered from the program example (see below).

2.2 MLS by recursion

If the primitive polynomial is given by:

$$p(x) = x^n + a_{n-1}x^{n-1} + ... + a_1 x + a_0$$

a maximum length sequence can be calculated from the recursion formula:

$$s_{k+n} = a_{n-1}s_{k+n-1} + ... + a_0 s_k$$

As an example the polynomial $x^3 + x + 1$ is primitive and the recursion $s_{k+3} = s_{k+1} + s_k$ can be used to generate the sequence:

$$1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, ...$$

Note that the sequence repeats itself after $2^3 - 1 = 7$ samples. If an irreducible but not primitive polynomial had been used the resulting repetition would have been shorter. It can be shown that the repetition can be no longer. That is why sequences generated by primitive polynomials are called maximum length sequences.
By replacing 0 by 1 and 1 by -1 one gets an MLS signal with period 7:

$$-1, -1, -1, 1, 1, -1, 1, -1, -1, -1, 1, ...$$

2.3 Autocorrelation function and crest factor

The autocorrelation function of the MLS signal is given by:

$$\rho(k) = \left\{ \begin{array}{cl} 1 & \text{for} \quad k = 0 \\ -\frac{1}{P} & \text{for} \quad 0 < k < P - 1 \end{array} \right.$$

It is seen that the spectrum of the sequence is flat except for the DC component. The crest factor $c_f = 1$ which is 3 dB lower than for a sine wave.

Chapter 3

Computing the impulse response

3.1 Deconvolution

If the impulse response of a system is h then the input output relation is given by the convolution:

$$y = h * x$$

If y is convolved with a sequence z:

$$y * z = h * x * z = h'$$

If:

1. $x * z$ is a sequence of delta functions with the periodicity of x
2. The length of h is shorter than the periodicity of x.

then h' is a periodic version of the impulse response h.
If $z_n = x_{-n}$ then $x*z$ is the autocorrelation function of the sequence x and it is seen from the above that when x is an MLS signal the first clause is fulfilled except for a DC offset. The periodicity of the MLS signal must be adjusted in order to fulfill the second clause.
For an MSL signal h' is given by:

$$h'(n) = \frac{1}{P+1}(\sum_{k=1}^{P} y(k)x(k-n) - \sum_{k=1}^{P} y(k))$$

where P is the periodicity of the sequence.
In matrix form the deconvolution term for a 7 sample sequence can be written:

$$\begin{pmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \\ h_5 \\ h_6 \\ h_7 \end{pmatrix} = \begin{pmatrix} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 \\ x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & x_1 \\ x_3 & x_4 & x_5 & x_6 & x_7 & x_1 & x_2 \\ x_4 & x_5 & x_6 & x_7 & x_1 & x_2 & x_3 \\ x_5 & x_6 & x_7 & x_1 & x_2 & x_3 & x_4 \\ x_6 & x_7 & x_1 & x_2 & x_3 & x_4 & x_5 \\ x_7 & x_1 & x_2 & x_3 & x_4 & x_5 & x_6 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{pmatrix}$$

3.2 The M-sequence matrix

As mentioned above the MLS signal only consists of $+1$ and -1 and the matrix can be written:

$$\begin{pmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \\ h_5 \\ h_6 \\ h_7 \end{pmatrix} = \begin{pmatrix} -1 & -1 & -1 & 1 & 1 & -1 & 1 \\ -1 & -1 & 1 & 1 & -1 & 1 & -1 \\ -1 & 1 & 1 & -1 & 1 & -1 & -1 \\ 1 & 1 & -1 & 1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & -1 & 1 \\ -1 & 1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & -1 & 1 & 1 & -1 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{pmatrix}$$

It is clear that no multiplications are required and the number of additions is $P(P-1)$. The number of additions can be reduced to approximately $P\log_2(P)$ by reordering the matrix and using the fast Hadamard transform as shown in the next section.

3.3 Decomposition of the M-sequence matrix

Returning to the binary formulation in Chapter 2 it is easy to see that the matrix M is symmetric and thus not only the rows but also the columns satisfy the recursion formula. Therefore every row (column) of M can be expressed as a linear modulo-2 combinations of the first N rows (columns) and we can write:

$$M = LS = L^{'}S^{'}$$

where L is a matrix of order $P \times N$, S is the $N \times P$ matrix formed by the first N rows of M and the primes denote matrix transposition. M can then be written:

$$M = LS = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix}$$

Since all rows of M are distinct all rows of L are distinct. Moreover since S consists of the first N rows of M the first N rows of L form an identity matrix.
If the the first N columns of S are considered one can write:

$$L\sigma = S'$$

and since every nonzero binary N-vector appears in both L and S' we get:

$$L = S'\sigma^{-1}$$

Consequently L consists of columns in M which forms an identity matrix by the first N rows.

3.4 Hadamard matrix equivalence

It can be shown that the Hadamard matrix can be written in a similar way to the M matrix:

$$H = BB' = \begin{pmatrix} 0&0&0&0&0&0&0&0 \\ 0&1&0&1&0&1&0&1 \\ 0&0&1&1&0&0&1&1 \\ 0&1&1&0&0&1&1&0 \\ 0&0&0&0&1&1&1&1 \\ 0&1&0&1&1&0&1&0 \\ 0&0&1&1&1&1&0&0 \\ 0&1&1&0&1&0&0&1 \end{pmatrix} = \begin{pmatrix} 0&0&0 \\ 0&0&1 \\ 0&1&0 \\ 0&1&1 \\ 1&0&0 \\ 1&0&1 \\ 1&1&0 \\ 1&1&1 \end{pmatrix} \begin{pmatrix} 0&0&0&0&1&1&1&1 \\ 0&0&1&1&0&0&1&1 \\ 0&1&0&1&0&1&0&1 \end{pmatrix}$$

Note that H has $P+1$ rows and $P+1$ columns. If a zero row and a zero column are bordered to the left and top of M, M can be transformed into H by reordering the columns and rows according to L and S. The fast Hadamard transform can then be used to compute the matrix product involving M and the number of additions can be reduced to $P\log_2(P)$.

3.5 Deconvolution step by step

Although theoretically complex the steps necessary to perform the deconvolution are fairly simple and are as follows:

1. Generate the MLS signal.
2. Reorder the output sequence from the system under consideration according to the columns of S and add a zero element in front. If the system under consideration is DC coupled $-\sum y_n$ should be used as the first element.
3. Apply the fast Hadamard transform to the reordered output sequence.
4. Omit the first element of the output from the Hadamard transform, reorder according to the rows of L and divide by $P+1$.

The reordering of y in the example above gives the following:

$$\begin{array}{c} \begin{pmatrix} 1&1&1&0&0&1&0 \\ 1&1&0&0&1&0&1 \\ 1&0&0&1&0&1&1 \end{pmatrix} \\ \begin{matrix} 7&6&4&1&2&5&3 \end{matrix} \end{array}$$

$$\begin{pmatrix} y_1 & y_2 & y_3 & y_4 & y_5 & y_6 & y_7 \end{pmatrix} \rightarrow \begin{pmatrix} y_4 & y_5 & y_7 & y_3 & y_6 & y_2 & y_1 \end{pmatrix}$$

The reordering of the output of the Hadamard transform gives the impulse response h:

$$\begin{array}{c}\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix}\end{array}\begin{array}{c} 4 \\ 2 \\ 1 \\ 6 \\ 3 \\ 7 \\ 5 \end{array}$$

$$\begin{pmatrix} h_4 & h_2 & h_1 & h_6 & h_3 & h_7 & h_5 \end{pmatrix} \rightarrow \begin{pmatrix} h_1 & h_2 & h_3 & h_4 & h_5 & h_6 & h_7 \end{pmatrix}$$

3.6 Program example

```
#include "stdio.h"

void GenerateSignal(bool *mls, double *signal, long P)
{
    long i;
    double *input = new double[P];

    for (i = 0; i < P; i++)             // Change 0 to 1 and 1 to -1
    {
        input[i] = -2 * mls[i] + 1;
    }

    for (i = 0; i < P; i++)             // Simulate a system with h = {2, 0.4, 0.2, -0.1, -0.8}, just an example
    {
        signal[i] =
          2.0 * input[(P + i - 0) % P]
        + 0.4 * input[(P + i - 1) % P]
        + 0.2 * input[(P + i - 2) % P]
        - 0.1 * input[(P + i - 3) % P]
        - 0.8 * input[(P + i - 4) % P];
  }

    delete []input;
}

void GenerateMls(bool *mls, long P, long N)
{
    const long maxNoTaps = 18;

    const bool tapsTab[16][18] =   {
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0,
        0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0,
        0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};


    bool taps[maxNoTaps];
    long i, j;
    bool *delayLine = new bool[maxNoTaps];
    long sum;

    for (i = 0; i < N; i++)         // copy the N'th taps table
    {
        taps[i] = tapsTab[maxNoTaps - N][i];
        delayLine[i] = 1;
    }

    for (i = 0; i < P; i++)         // Generate an MLS by summing the taps mod 2
    {
        sum = 0;
        for (j = 0; j < N; j++)
        {
            sum += taps[j] * delayLine[j];
        }
        sum &= 1;               // mod 2


        mls[i] = delayLine[N - 1];

        for (j = N - 2; j >= 0; j--)
        {
            delayLine[j + 1] = delayLine[j];
        }
        delayLine[0] = *(bool*)&sum;
    }

    delete []delayLine;
}

void FastHadamard(double *x, long P1, long N)
{
    long i, i1, j, k, k1, k2;
    double temp;

    k1 = P1;

    for (k = 0; k < N; k++)
    {
        k2 = k1 >> 1;
        for (j = 0; j < k2; j++)
        {
            for (i = j; i < P1; i = i + k1)
            {
                i1 = i + k2;
                temp = x[i] + x[i1];
                x[i1] = x[i] - x[i1];
                x[i] = temp;
            }
        }
        k1 = k1 >> 1;
    }
}

void PermuteSignal(double *sig, double *perm, long *tagS, long P)
{
    long i;
    double dc = 0;

    for (i = 0; i < P; i++)
        dc += sig[i];

    perm[0] = -dc;
    for (i = 0; i < P; i++)         // Just a permutation of the measured signal
        perm[tagS[i]] = sig[i];
}

void PermuteResponse(double *perm, double *resp, long *tagL, long P)
{
    long i;
    const double fact = 1 / double(P + 1);


    for (i = 0; i < P; i++)         // Just a permutation of the impulse response
    {
        resp[i] = perm[tagL[i]] * fact;
    }
    resp[P] = 0;
}

void GeneratetagL(bool *mls, long *tagL, long P, long N)
{
    long i, j;
    long *colSum = new long[P];
    long *index = new long[N];

    for (i = 0; i < P; i++)         // Run through all the columns in the autocorr matrix
    {
        colSum[i] = 0;
        for (j = 0; j < N; j++)     // Find colSum as the value of the first N elements regarded as a binary number
        {
            colSum[i] += mls[(P + i - j) % P] << (N - 1 - j);
        }

        for ( j = 0; j < N; j++)    // Figure out if colSum is a 2^j number and store the column as the j'th index
        {
            if (colSum[i] == (1 << j))
                index[j] = i;
        }
    }

    for (i = 0; i < P; i++)         // For each row in the L matrix
    {
        tagL[i] = 0;
        for ( j = 0; j < N; j++)    // Find the tagL as the value of the rows in the L matrix regarded as a binary number
        {
            tagL[i] += mls[(P + index[j] - i) % P] * (1 << j);
        }
    }

    delete []colSum;
    delete []index;
}

void GeneratetagS(bool *mls, long *tagS, long P, long N)
{
    long i, j;

    for (i = 0; i < P; i++)         // For each column in the S matrix
    {
        tagS[i] = 0;
        for (j = 0; j < N; j++)     // Find the tagS as the value of the columns in the S matrix regarded as a binary number
        {
            tagS[i] += mls[(P + i - j) % P] * (1 << (N - 1 - j));
        }
    }
}

void main()
{
    const long N = 18;
    const long P = (1 << N) - 1;

    long i;

    bool *mls = new bool[P];
    long *tagL = new long[P];
    long *tagS = new long[P];

    double *signal = new double[P];

    double *perm = new double[P + 1];

    double *resp = new double[P + 1];

    GenerateMls(mls, P, N);                     // Generate the Maximum length sequence
    GeneratetagL(mls, tagL, P, N);              // Generate tagL for the L matrix
    GeneratetagS(mls, tagS, P, N);              // Generate tagS for the S matrix

    GenerateSignal(mls, signal, P);             // Do a simulated measurement and get the signal
    PermuteSignal(signal, perm, tagS, P);        // Permute the signal according to tagS
    FastHadamard(perm, P + 1, N);                // Do a Hadamard transform in place
    PermuteResponse(perm, resp, tagL, P);        // Permute the impulseresponse according to tagL

    printf("Impulse response:\n");
    for (i = 0; i < 10; i++)
        printf("%10.5f\n", resp[i]);

    delete []mls;
    delete []tagL;
    delete []tagS;
    delete []signal;
    delete []perm;
    delete []resp;
}
```

Bibliography

[1] W. T. Chu, "Impulse Response and Reverberation-Decay Measurements Made by Using a Periodic Pseudorandom Sequence," *Applied Acoustics*, vol. 29, pp. 193-205, 1990.

[2] M.Cohn and A. Lempel, "On Fast M-Sequence Transform," *IEEE Trans. Inf.Theory*, IT-23, pp 135-137 1977.

[3] F. J. MacWilliams and N. J. A. Sloane, "Pseudo-random Sequences and Arrays," *Proc. Inst Elec. Eng.*, vol. 64, no 12, pp. 1715-1729, 1976.

[4] M. R. Schroeder, *Number Theory in Science and Communication*, Third Edition 1997, Springer.

[5] R. J. McEliece, *Finite Fields for Computer Scientists and Engineers*, 1987, Kluwer Academic Publishers.

[6] R. Lidl & H. Niederreiter, *Finite Fields*, Enclyclopedia of Mathematics and its Applications 20, 1997, Cambridge University Press.