Hideo Okawara's
Mixed Signal Lecture Series

DSP-Based Testing – Fundamentals 48
Lagrange Interpolation

ADVANTEST Corporation

February 2013

Preface to the Series

ADC and DAC are the most typical mixed signal devices. In mixed signal testing, analog stimulus signal is generated by an arbitrary waveform generator (AWG) which employs a D/A converter inside, and analog signal is measured by a digitizer or a sampler which employs an A/D converter inside. The stimulus signal is created with mathematical method, and the measured signal is processed with mathematical method, extracting various parameters. It is based on digital signal processing (DSP) so that our test methodologies are often called DSP-based testing.

Test/application engineers in the mixed signal field should have thorough knowledge about DSP-based testing. FFT (Fast Fourier Transform) is the most powerful tool here. This corner will deliver a series of fundamental knowledge of DSP-based testing, especially FFT and its related topics. It will help test/application engineers comprehend what the DSP-based testing is and assorted techniques.

Editor's Note

For other articles in this series, please visit the Advantest web site at www.verigy.com/go/gosemi.

Preface

When applying S-parameter de-embedding to a measured waveform in order to remove the distortion caused by the loss of signal path, you may often need to interpolate S-parameters at required frequencies. The S-parameters on hand are measured at different frequency points from the frequencies you need. Usually in that case you may apply the linear interpolation to calculate the S-parameters with referring to the nearest two points right before and after the target test frequency. It works fine when the available data is dense, but the linear approximation may often cause big error when the original data points are distributed on an extreme curve such as S-parameters. The Lagrange interpolation is very useful in that situation. This paper focuses on this specific interpolation method. You can find further discussions on the web so that you should learn its theoretical background by yourself if you are interested in. The point of this article is how to apply it appropriately in our test.

Interpolation of Curve

Equation (1) describes a linear line shown in Figure 1. The factors *a* and *b are unknown*.

$$y = a \cdot x + b \qquad (1)$$

If you know the coordinates of two different points A(x_1,y_1) and B(x_2,y_2) on the line, the unknown *a* and *b* can be resolved. Then Equation (1) can solve the coordinates of any location P(x,y) on the line.

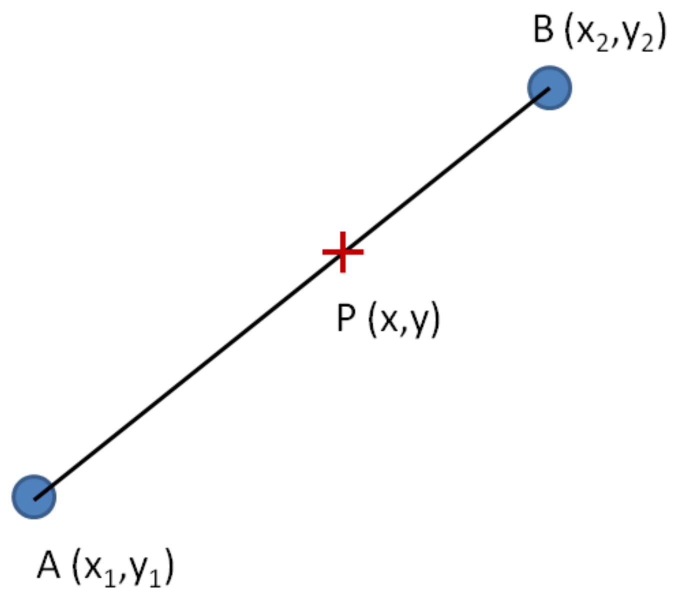

Figure 1: Linear Approximation

The Lagrange polynomial for the line in Figure 1 is defined as Equation (2).

$$y = \frac{x - x_2}{x_1 - x_2} y_1 + \frac{x - x_1}{x_2 - x_1} y_2 \qquad (2)$$

If x=x_1, then Equation (2) becomes y_1. If x=x_2, (2) becomes y_2. So Equation (2) definitely describes the line in Figure 1. Equation (2) can be modified as follows;

$$y=\frac{x-x_2}{x_1-x_2}y_1+\frac{x-x_1}{x_2-x_1}y_2=\frac{y_1-y_2}{x_1-x_2}x+\frac{x_1y_2-x_2y_1}{x_1-x_2} \tag{3}$$

Consequently Equation (3) is identical to Equation (1) when a=(y_1-y_2)/(x_1-x_2) and b=(x_1y_2-x_2y_1)/(x_1-x_2). So Equation (2) indicates the coordinates of any location P(x,y) on the line. If you have the coordinate x of the point P, Equation (2) resolves the coordinate y of P.

Figure 2 shows a second order curve. Points A(x_1,y_1), B(x_2,y_2) and C(x_3,y_3) exactly sit on the curve. The Lagrange polynomial for the curve is defined as follows;

$$y=\frac{x-x_2}{x_1-x_2}\cdot\frac{x-x_3}{x_1-x_3}y_1+\frac{x-x_1}{x_2-x_1}\cdot\frac{x-x_3}{x_2-x_3}y_2+\frac{x-x_1}{x_3-x_1}\cdot\frac{x-x_2}{x_3-x_2}y_3 \tag{4}$$

If x=x_1, then Equation (4) becomes y_1. If x=x_2, (4) becomes y_2. If x=x_3, (4) becomes y_3. So Equation (4) definitely describes the curve in Figure 2. If you know the coordinates of three different points A(x_1,y_1), B(x_2,y_2) and C(x_3,y_3), the coordinates of any location P(x,y) on the curve is held by Equation (4). If you know the coordinate x of the point P, Equation (4) resolves the coordinate y of P.

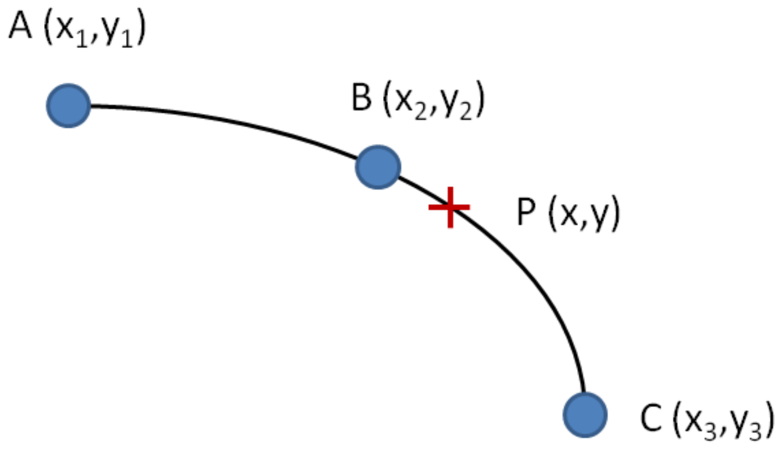

Figure 2: **3 Points Curve Fitting**

The same discussion goes for the third order curve in Figure 3. The Lagrange polynomial for the curve is defined as follows;

$$\begin{aligned}y=&\frac{x-x_2}{x_1-x_2}\cdot\frac{x-x_3}{x_1-x_3}\cdot\frac{x-x_4}{x_1-x_4}y_1+\frac{x-x_1}{x_2-x_1}\cdot\frac{x-x_3}{x_2-x_3}\cdot\frac{x-x_4}{x_2-x_4}y_2\\&+\frac{x-x_1}{x_3-x_1}\cdot\frac{x-x_2}{x_3-x_2}\cdot\frac{x-x_4}{x_3-x_4}y_3+\frac{x-x_1}{x_4-x_1}\cdot\frac{x-x_2}{x_4-x_2}\cdot\frac{x-x_3}{x_4-x_3}y_4\end{aligned} \tag{5}$$

If x=x_1, then Equation (5) becomes y_1. If x=x_2, (5) becomes y_2. If x=x_3, (5) becomes y_3. If x=x_4, (5) becomes y_4. So Equation (5) definitely describes the curve in Figure 3. If you know the coordinates of four different points A(x_1,y_1), B(x_2,y_2) C(x_3,y_3) and D(x_4,y_4), the coordinates of any location P(x,y) on the curve meet Equation (5). If you have the coordinate x of the point P, Equation

(5) resolves the coordinate *y* of P.

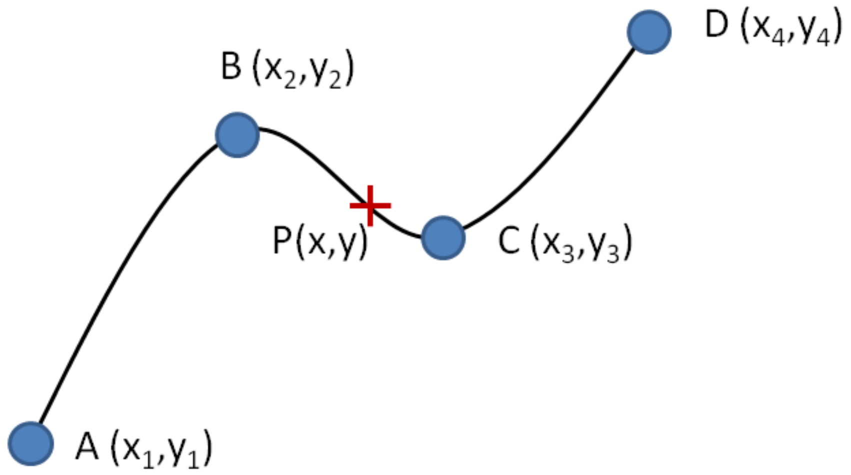

Figure 3: 4 Points Curve Fitting

This discussion goes on for the larger order curves and polynomials. But the second order or third order curve with three or four known coordinates of points is good enough for our interpolation. As you already notice, you need one more point of data than the order of the polynomial curve. When you have two known points of data, you can draw a linear line between them. When you have three known points of data, you can draw a second order polynomial among them. When you have four known points of data, you can draw a third order polynomial among them.

When the distance between adjacent two points is so far away that the linear interpolation is not accurate to approximate a halfway point, the Lagrange interpolation on adjacent three or four points is practically useful. The key of this interpolation is the estimated curve definitely goes through the given points. This is not the case of regression. This is the important point.

Examples of Lagrange Interpolation

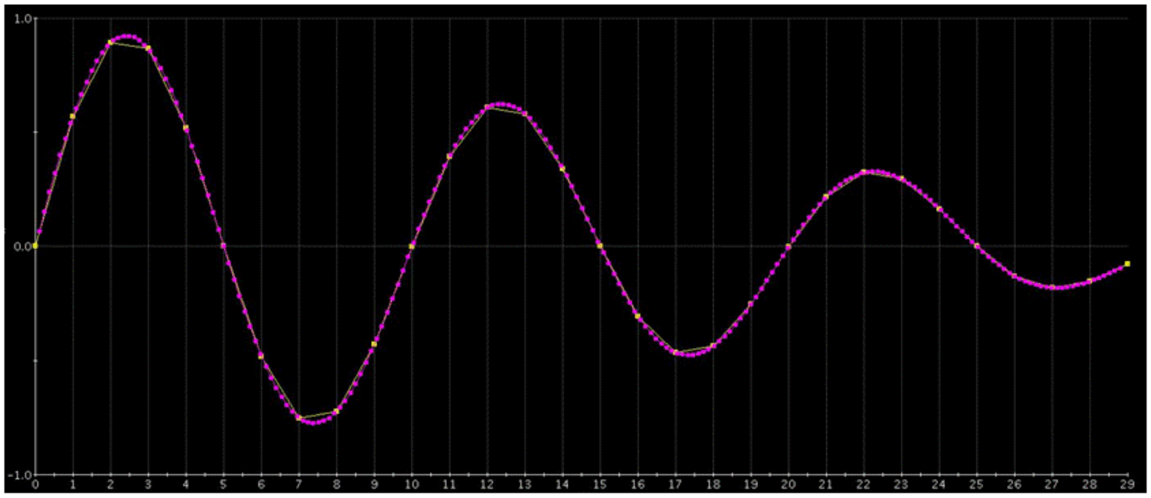

Figure 4: Sinusoid Attenuating Amplitude

Let’s look at Figure 4. There are 30 yellow points which is 3-cycle sinusoidal waveform whose

amplitude is attenuating. Each of the points is connected with a yellow linear line. The yellow lines show the linear approximation. The red dots are calculated by using the Lagrange interpolation with the third order polynomials based on Equation (5). There are 206 points approximated. Each of them is calculated with referring adjacent four yellow points, which are the two points right before the taget location and the two points right after the target location in most cases. Even if the target is located near the terminal points, the equation can work appropriately. As a result, the estimated red dots make a smooth amplitude-attenuating sinusoid.
List 1 is the example program code that is used for drawing Figure 4.   "dXarray[]" and "dYarray[]" are the (X, Y) coordinates of known points which is colored yellow in the figure. The pass parameter "dPx" is the X value of a target unknown point. The subroutine returns an estimated Y value.

```
double  dLagrangeInterpolation(
    double      dPx,
    ARRAY_D  &  dXarray,
    ARRAY_D  &  dYarray
)
{
    int     k,N;
    double  dPy;

    N=dXarray.size();

    k=0;
    while (dPx>dXarray[k]) k++;
    if (k<2)     k=2;
    if (k>(N-2)) k=N-2;

    ARRAY_D  dX4pts(dXarray,4,k-2);
    ARRAY_D  dY4pts(dYarray,4,k-2);

    dPy=dLagrange4(dPx,dX4pts,dY4pts);

    return(dPy);
}

```

```
double dLagrange4(
    double    dX,
    ARRAY_D & dXA,
    ARRAY_D & dYA
)
{
    double dY;
    double dU,dV;

    dU=(dX-dXA[1])*(dX-dXA[2])*(dX-dXA[3]);
    dV=(dXA[0]-dXA[1])*(dXA[0]-dXA[2])*(dXA[0]-dXA[3]);
    dY=dU/dV*dYA[0];

    dU=(dX-dXA[0])*(dX-dXA[2])*(dX-dXA[3]);
    dV=(dXA[1]-dXA[0])*(dXA[1]-dXA[2])*(dXA[1]-dXA[3]);
    dY=dY+dU/dV*dYA[1];

    dU=(dX-dXA[0])*(dX-dXA[1])*(dX-dXA[3]);
    dV=(dXA[2]-dXA[0])*(dXA[2]-dXA[1])*(dXA[2]-dXA[3]);
    dY=dY+dU/dV*dYA[2];

    dU=(dX-dXA[0])*(dX-dXA[1])*(dX-dXA[2]);
    dV=(dXA[3]-dXA[0])*(dXA[3]-dXA[1])*(dXA[3]-dXA[2]);
    dY=dY+dU/dV*dYA[3];

    return(dY);
}

```

List 1: Example Code of Lagrange Interpolation Calculation

Let’s look at Figure 5. There are 30 yellow points which is 3-cycle spiral waveform whose amplitude is attenuating. This kind of curve is often observed in the Smith chart display in RF measurement. Each point would be a (x, y) vector location with monotonically ascending or descending a frequency. Each of the points is connected with a yellow linear line. The yellow lines show the linear approximation. You can read it at a glance that the linear approximation is not accurate in this case.

The red dots are calculated by using the Lagrange interpolation with the third order polynomials. There are 206 points approximated. Each of them is calculated with referring adjacent four yellow points, which are the two points right before the taget location and the two points right after the target location in most cases. Even if the target is located near the terminal points, the equation can work appropriately. As a result, the estimated red dots make a smooth spiral curve.

List 2 is the example program code that is used for drawing Figure 5. The“dFREQ[]” and “CMPLX[]” are the monotonically increasing frequency data and (X, Y) coordinates of known points which is colored yellow in the figure. These array data would correspond to, for example, the S-parameter data of a certain network. The pass parameter “dFrequency” is the frequency of a target unknown point. The subroutine returns a pair of estimated vector location value. The procedure is basically the same as the one in List 1. The difference is that a real number is returned in List 1 and a complex number is returned in List 2.

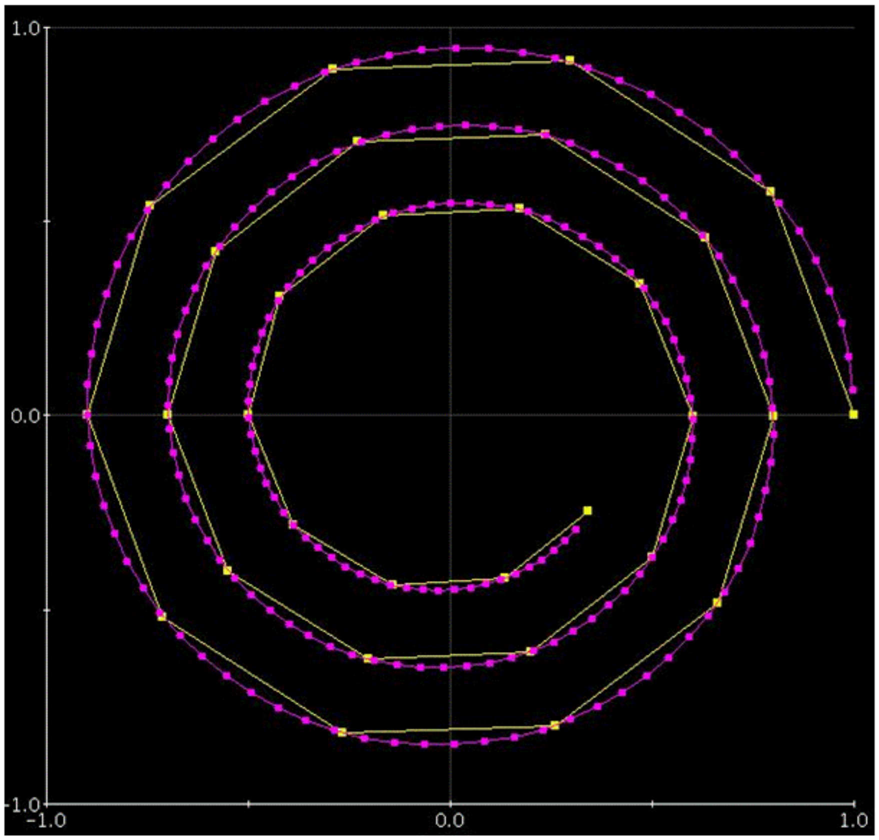

Figure 5: Sinusoid Attenuating Amplitude

```
COMPLEX CLagrangeInterpolation(
   double           dFrequency,
   ARRAY_D       &  dFREQ,       // Frequency Array   (f)
   ARRAY_COMPLEX &  CMPLX        // S-parameter Array (x,y)
)
{
   int      k,N;
   COMPLEX  Cresult;

   N=dFREQ.size();

   k=0;
   while (dFrequency>dFREQ[k]) k++;
   if (k<2)     k=2;
   if (k>(N-2)) k=N-2;

   ARRAY_D        dFREQ4(dFREQ,4,k-2);
   ARRAY_COMPLEX  CMPLX4(CMPLX,4,k-2);

   Cresult=CLagrange4(dFrequency,dFREQ4,CMPLX4);

   Return(Cresult);
}

COMPLEX   CLagrange4(
   double          dX,
   ARRAY_D       & dXA,
   ARRAY_COMPLEX & CYA
)
{
   COMPLEX        CX,CY;
   COMPLEX        CU,CV;
   ARRAY_COMPLEX  CXA;


   CX.real()=dX;  CX.imag()=0.0;

   CXA.resize(dXA.size());
   DSP_CONV_D_C(dXA,CXA,1.0,0.0);

   CU=(CX-CXA[1])*(CX-CXA[2])*(CX-CXA[3]);
   CV=(CXA[0]-CXZ[1])*(CXA[0]-CXA[2])*(CXA[0]-CXA[3]);
   CY=CU/CV*CYA[0];

   CU=(CX-CXA[0])*(CX-CXA[2])*(CX-CXA[3]);
   CV=(CXA[1]-CXA[0])*(CXA[1]-CXA[2])*(CXA[1]-CXA[3]);
   CY=CY+CU/CV*CYA[1];

   CU=(CX-CXA[0])*(CX-CXA[1])*(CX-CXA[3]);
   CV=(CXA[2]-CXA[0])*(CXA[2]-CXA[1])*(CXA[2]-CXA[3]);
   CY=CY+CU/CV*CYA[2];

   CU=(CX-CXA[0])*(CX-CXA[1])*(CX-CXA[2]);
   CV=(CXA[3]-CXA[0])*(CXA[3]-CXA[1])*(CXA[3]-CXA[2]);
   CY=CY+CU/CV*CYA[3];


   return(CY);
}
```
List 2: Example Code of 2-dimentional Lagrange Interpolation Calculation