Wavelet Transform Method

-Theory and software coding-

Koichi KUSUNOKI
Visiting Assistant Project Scientist, University of California, San Diego
& Senior Researcher, Building Research Institute of Japan

July, 2005

CONTENTS

Preface

When I found a book of Wavelet transform in a bookstore in Tokyo, it looks great method. Then I bought two books there, and start to learn what the Wavelet transform is. I wanted to code a Wavelet transform software by myself, since I needed a software to eliminate erros in measured acceleration and calculate displacement from it with double integral.

Since then, I bought more than 10 books including two English books. But unfortunately, most of all are too mathematical. They would be good books for mathematicians or those who are famillier with high-level mathematics. I am, however, not one of them. I was not able to understand everything, then not able to code a software.

But one day, I met the book, "Wavelet Beginners' guide" (Dr. Susumu Kashiwabara, published by Tokyo Denki Univ., 2003. in Japanese). The book is wonderful; it shows us from mathematical background to coding technique.

This report is mostly based on the book. I would like to respect the book, therefore the same equation numbers and figures as shown in the book are used. The book is still a little above my head, then I tried to understand equations as much as possible. They are also shown in this report.

This report goes on the way to develop a software of Wavelet transform. I believe you can do it with only this report. However, if you are interested in other mathematical background, you can find a lot of good books and references in a bookstore.

In the last chapter, the source code of my software is shown. If you are interested, you can get it by sending me an e-mail.

Finally I would like to acknowledge Prof. Elgamal for giving me the chance to conduct a research with him here in University of California, San Diego.

Koichi KUSUNOKI kusunoki@kenken.go.jp
Visiting Assistant Project Scientist of UCSD
& Senior Researcher of Building Research Institute of Japan

1 Introduction

The wavelet transform can be shown in Equation (1.1).

$$\left(W_{\psi f}\right)_{(b,a)} = \int_{-\infty}^{\infty} \frac{1}{\sqrt{a}} \overline{\psi}_{\left(\frac{x-b}{a}\right)} f_{(x)} dx \tag{1.1}$$

The portion of the equation, $\int_{-\infty}^{\infty} \overline{\psi}_{\left(\frac{x-b}{a}\right)} f_{(x)} dx$, indicates how the signal, $f_{(x)}$, is similar to the function $\psi_{\left(\frac{x-b}{a}\right)}$ at around $x=b$. If similarity is high, the integral becomes large. The factor "a" represents scale, the width of time window. Larger "a" gives wider width of the window. Thus "a" is a factor for the frequency. The factor "b" represents shift. It shifts $\psi_{(x)}$ along the time axis. Thus "b" is a factor for the time.

The inverse wavelet transform can be conducted with Equation (1.2).

$$f_{(x)} = \frac{1}{C_{\psi}} \iint_{R^2} \left(W_{\psi f}\right)_{(b,a)} \frac{1}{\sqrt{a}} \psi_{\left(\frac{x-b}{a}\right)} \frac{da \cdot db}{a^2} \tag{1.2}$$

Then, the following admissible condition must hold:

$$C_{\psi} = \int_{-\infty}^{\infty} \frac{\left|\psi_{(\omega)}\right|^2}{|\omega|} d\omega < \infty \tag{1.3}$$

As the condition, Equation (1.4) can be used instead of Equation (1.3).

$$\int_{-\infty}^{\infty} \psi_{(x)} dx = 0 \tag{1.4}$$

The width of window for time and frequency domains for time-frequency analysis, Δt and Δf, must satisfy the uncertainty relation as shown below.

$$\Delta f \cdot \Delta t \geq \frac{1}{2}$$

Thus, the highest resolution is given when $\Delta f \cdot \Delta t$ is 1/2. Therefore, the wavelet transform gives the highest resolution when $\frac{1}{a} \cdot b$ is 1/2. Thereby, the following "a" and "b" gives highest resolution. The factor "j" is called rank of the wavelet transform. The number of data points for (j) rank is half of that for (j-1) rank.

$$\begin{cases} a = 2^{-j} \\ b = 2^{-j} \cdot k \end{cases}$$

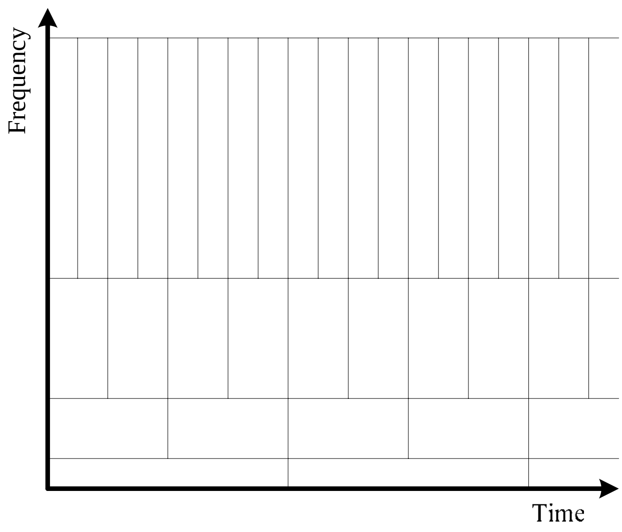


Figure 1.13 Divided time-frequency plane

If $d_k^{(j)}$ is defined as follows, the equation (1.7) and (1.8) are derived from equation (1.1) and (1.2).

$$d_k^{(j)} = \left(W_{\psi f}\right)_{\left(2^{-j}\cdot k,\, 2^{-j}\right)}$$

$$d_k^{(j)} = 2^j \int_{-\infty}^{\infty} \overline{\psi}_{\left(2^j x - k\right)} f_{(x)} dx \qquad (1.7)$$

$$f_{(x)} \approx \sum_j \sum_k d_k^{(j)} \psi_{\left(2^j x - k\right)} \qquad (1.8)$$

Here, $g_{j(x)}$ and $f_{j(x)}$ are defined as Equation (1.9) and (1.10).

$$g_{j(x)} \equiv \sum_k d_k^{(j)} \cdot \psi_{\left(2^j x - k\right)} \qquad (1.9)$$

$$f_{j(x)} = g_{j-1(x)} + g_{j-2(x)} + g_{j-3(x)} + \cdots \qquad (1.10)$$

With Equation (1.10), signal $f_{(x)}$ can be decomposed and reconstituted as Equation (1.11)

$$f_{(x)} = f_{0(x)} = g_{-1(x)} + g_{-2(x)} + g_{-3(x)} + \cdots \qquad (1.11)$$

The mother wavelet $\psi_{(x)}$ must be basis function so that the decomposition and reconstitution are unique. The mother wavelet calculated as Equation (1.13) with the scaling function $\phi_{(x)}$ is known as a basis function.

$$\psi_{(x)} = \sum_k q_k \cdot \phi_{(2x-k)} \qquad (1.13)$$

where q_k is a defined sequence. The scaling function $\phi_{(x)}$ must satisfy the two-scale relation as shown in Equation (1.12).

$$\phi_{(x)} = \sum_k p_k \cdot \phi_{(2x-k)}$$

where p_k is also a defined sequence. $f_{j(x)}$ can be calculated as Equation (1.15) with the scaling function.

$$f_{j(x)} = \sum_k c_k^{(j)} \cdot \phi_{\left(2^j x-k\right)} \tag{1.15}$$

From Equation (1.11), the decomposition and reconstitution are conducted as Equation (1.14).

$$f_{j(x)} = f_{j-1(x)} + g_{j-1(x)} \tag{1.14}$$

In calculation, Equation (1.16) is used for the decomposition instead of (1.14). Equation (1.14) is called decomposition algorism.

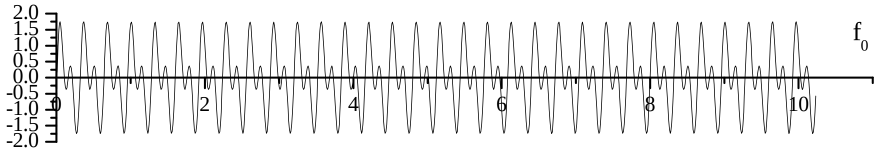
$$\begin{aligned} c_k^{(j-1)} &= \frac{1}{2}\sum_{\ell \in Z} g_{2k-\ell} \cdot c_\ell^{(j)} \\ d_k^{(j-1)} &= \frac{1}{2}\sum_{\ell \in Z} h_{2k-\ell} \cdot c_\ell^{(j)} \end{aligned} \tag{1.16}$$

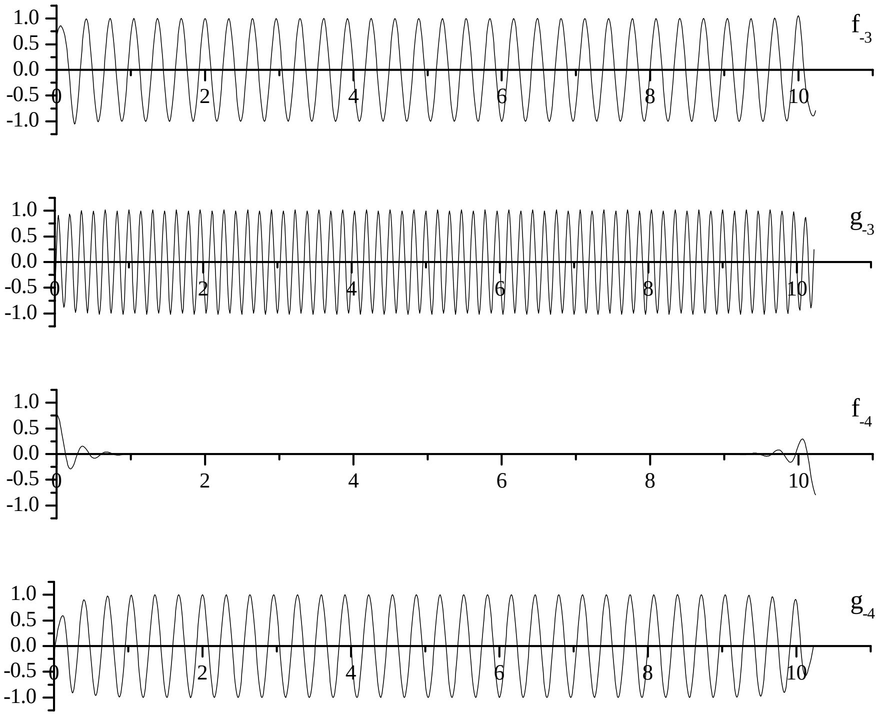
where g_k and h_k are defined sequences.

Note that sequences p_k, q_k, g_k, and h_k are uniquely defined according to the mother wavelet function, and independent upon the rank, j.

If the sequences of p_k, q_k, g_k, h_k, and $c_k^{(0)}$ are defined for a mother wavelet and input signal $f_{0(x)}$, the signal can be decomposed to $g_{j(x)}$ and $f_{N(x)}$. Some decomposed components $g_{i(x)}$ can be neglected for reconstitution by simply neglecting the i-th component $g_{i(x)}$ in Equation (1.11) or putting zero to d_k^i.

As an example, the wavelet transform results of the combination of two sine wave (F=6.25 and 3.125Hz) are shown below. The Nyquist frequencies for -3 and -4 rank are 6.25Hz and 3.125 Hz. It can be seen that two sine waves are successfully decomposed to these rank (g_{-3} and g_{-4}).





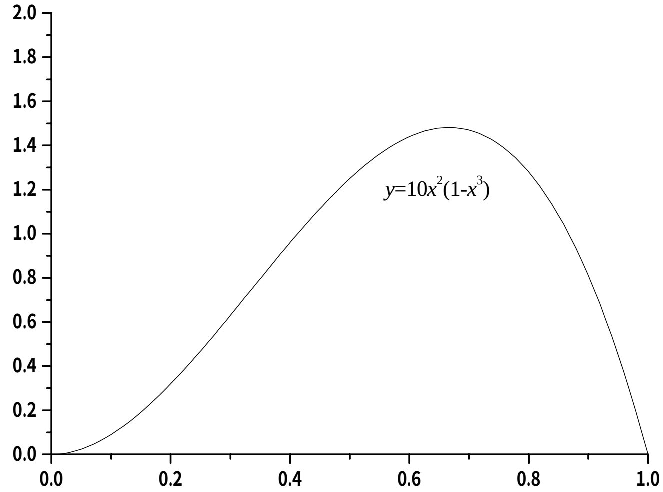
f_0 , f_{-3} , g_{-3} , f_{-4} , and g_{-4} for $y = \sin\left(\frac{2\pi}{32} \cdot \frac{t}{0.01}\right) + \sin\left(\frac{2\pi}{16} \cdot \frac{t}{0.01}\right)$

Since the transform is conducted in the temporal domain, the Wavelet transform method is useful for non-linear response. It can be observed along the time axis that the predominant component may change due to non-linearity, unlike Fourier transform.

2 Haar wavelet

2.1 Decomposition and reconstitution algorisms

The function of $y = 10x^3(1-x)$ (shown below) in the range of $I = [0 \quad 1)$ was studied.



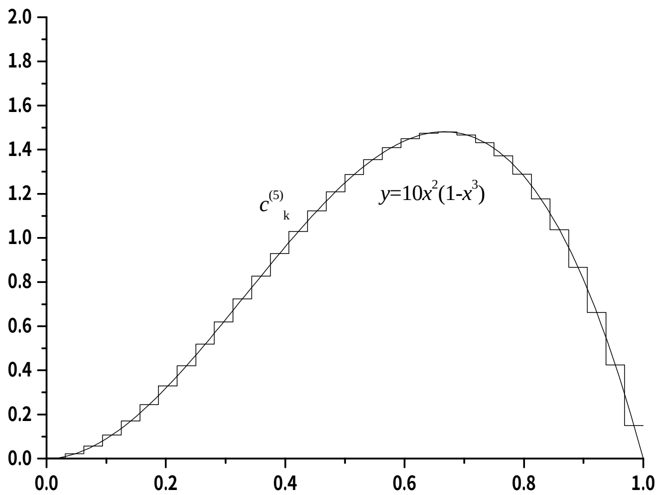
The range of I is divided into 2^5 segments when j is 5. The range for the k-th segment is shown in the following fashion.

$$I_k^{(5)} = \left[\frac{k}{2^5} \quad \frac{k+1}{2^5}\right) \qquad k = 0,1,\cdots,2^5-1$$

The average value for the k-th segment can be adopted for the approximate value for the segment $c_k^{(5)}$.

$$c_k^{(5)} \cdot \frac{1}{2^5} = \int_{I_k^{(5)}} f_{(x)} dx$$

$$\Leftrightarrow c_k^{(5)} = 2^5 \int_{\frac{k}{2^5}}^{\frac{k+1}{2^5}} f_{(x)} dx \tag{2.3}$$



$c_k^{(4)}$ and $c_k^{(3)}$ are calculated as Equation (2.4) and (2.5) respectively.

$$c_k^{(4)} = 2^4 \int_{\frac{k}{2^4}}^{\frac{k+1}{2^4}} f_{(x)} dx \tag{2.4}$$

$$c_k^{(3)} = 2^3 \int_{\frac{k}{2^3}}^{\frac{k+1}{2^3}} f_{(x)} dx \tag{2.5}$$

The following scaling function $\phi_{H(x)}$ is applied for the Haar wavelet.

$$\phi_{H(x)} = \begin{cases} 1 & 0 \le x < 1 \\ 0 & x < 0, 1 \le x \end{cases}$$

For each segment, $\phi_{H\left(2^j x-k\right)}$ is calculated as follows.

$$\phi_{H\left(2^j x-k\right)} = \begin{cases} 1 & \dfrac{k}{2^j} \le x < \dfrac{k+1}{2^j} \\ 0 & other\ x \end{cases}$$

Therefore, $\phi_{H\left(2^j x-k\right)}$ times $c_k^{(j)}$ becomes the approximate value for the $I_k^{(j)}$ segment of the j-th rank. Thus, following equations are derived.

$$f_{5(x)} = \sum_{k=0}^{2^5-1} c_k^{(5)} \cdot \phi_{H\left(2^5 x-k\right)} \tag{2.6}$$

$$f_{4(x)} = \sum_{k=0}^{2^4-1} c_k^{(4)} \cdot \phi_{H\left(2^4 x-k\right)} \tag{2.7}$$

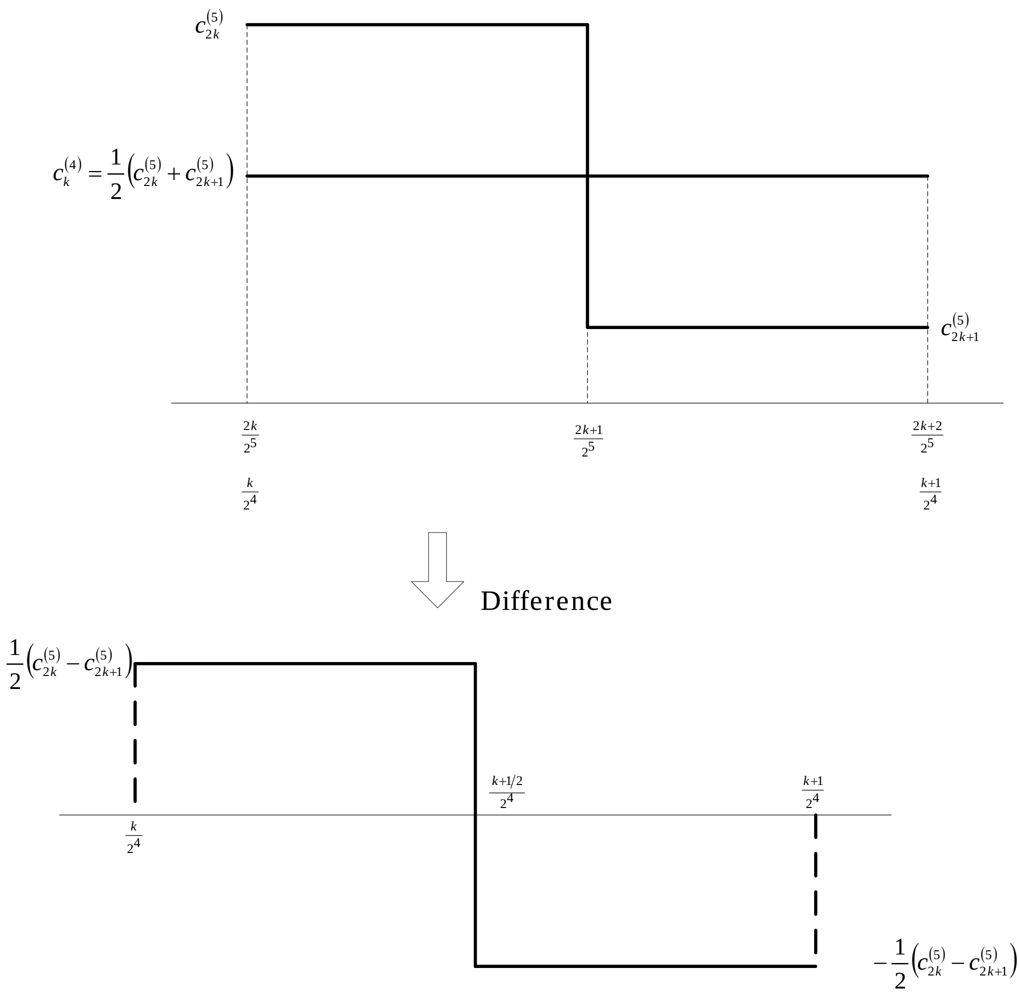
$$f_{3(x)} = \sum_{k=0}^{2^3-1} c_k^{(3)} \cdot \phi_{H\left(2^3 x-k\right)}$$

The decomposition algorism is derived as Equation (2.8) from Equation (2.4) and (2.3).

$$c_k^{(4)} = 2^4 \int_{\frac{k}{2^4}}^{\frac{k+1}{2^4}} f_{(x)} dx = 2^4 \int_{\frac{2k}{2^5}}^{\frac{2k+1}{2^5}} f_{(x)} dx + 2^4 \int_{\frac{2k+1}{2^5}}^{\frac{2k+2}{2^5}} f_{(x)} dx = \frac{1}{2}\left(c_{2k}^{(5)} + c_{2k+1}^{(5)}\right) \tag{2.8}$$

The difference between $f_{5(x)}$ and $f_{4(x)}$, $g_{4(x)}$ can be calculated as below.

$$g_{4(x)} = f_{5(x)} - f_{4(x)}$$



The following function, called mother wavelet $\psi_{H(x)}$ is applied for the Haar wavelet.

$$\psi_{H(x)} = \begin{cases} 1 & 0 \le x < 1/2 \\ -1 & 1/2 \le x < 1 \\ 0 & Other \end{cases} \tag{2.9}$$

With $\psi_{H(x)}$, $g_{4(x)}$ can be calculated as Equation (2.10)

$$g_{4(x)} = \sum_k d_k^{(4)} \psi_{\left(2^4 x-k\right)}$$
$$d_k^{(4)} = \frac{1}{2}\left(c_{2k}^{(5)} - c_{2k+1}^{(5)}\right) \qquad (2.10)$$

Therefore, $f_{j(x)}$ and $g_{j(x)}$ are calculated as Equation (2.11) and (2.14) respectively.

$$f_{j(x)} = \sum_k c_k^{(j)} \cdot \phi_{H\left(2^j x-k\right)} \qquad (2.11)$$

$$g_{j(x)} = \sum_k d_k^{(j)} \cdot \psi_{H\left(2^j x-k\right)} \qquad (2.14)$$

The decomposition and reconstitution algorisms with Equation (2.13) are calculated as Equation (2.12) and (2.15) respectively.

$$f_{j(x)} = f_{j-1(x)} + g_{j-1(x)} \qquad (2.13)$$

$$\begin{cases} c_k^{(j-1)} = \frac{1}{2}\left(c_{2k}^{(j)} + c_{2k+1}^{(j)}\right) \\ d_k^{(j-1)} = \frac{1}{2}\left(c_{2k}^{(j)} - c_{2k+1}^{(j)}\right) \end{cases} \qquad (2.12)$$

$$\begin{cases} c_{2k}^{(j)} = c_k^{(j-1)} + d_k^{(j-1)} \\ c_{2k+1}^{(j)} = c_k^{(j-1)} - d_k^{(j-1)} \end{cases} \quad \text{(from Equation (2.12))} \qquad (2.15)$$

2.2 Two-scale relation for the Haar wavelet

The following equation is derived from Equation (2.11).

$$\begin{aligned} f_{j(x)} &= \sum_k c_k^{(j)} \cdot \phi_{H\left(2^j x-k\right)} \\ &= \sum_k c_{2k}^{(j)} \cdot \phi_{H\left(2^j x-2k\right)} + \sum_k c_{2k+1}^{(j)} \cdot \phi_{H\left(2^j x-2k-1\right)} && (\textit{odd and even parts}) \\ &= \sum_k \left(c_k^{(j-1)} + d_k^{(j-1)}\right) \cdot \phi_{H\left(2^j x-2k\right)} + \sum_k \left(c_k^{(j-1)} - d_k^{(j-1)}\right) \cdot \phi_{H\left(2^j x-2k-1\right)} && (\textit{from Equation}\,(2.15)) \\ &= \sum_k c_k^{(j-1)} \left(\phi_{H\left(2^j x-2k\right)} + \phi_{H\left(2^j x-2k-1\right)}\right) \cdot + \sum_k d_k^{(j-1)} \left(\phi_{H\left(2^j x-2k\right)} - \phi_{H\left(2^j x-2k-1\right)}\right) \end{aligned}$$

On the other hand, the following equation is derived from Equation (2.13).

$$\begin{aligned} f_{j(x)} &= f_{j-1(x)} + g_{j-1(x)} \\ &= \sum_k c_k^{(j-1)} \cdot \phi_{H\left(2^{j-1} x-k\right)} + \sum_k d_k^{(j-1)} \cdot \psi_{H\left(2^{j-1} x-k\right)} \end{aligned}$$

Then the following relations can be derived from these two equations.

$$\phi_{H\left(2^{j-1} x-k\right)} = \phi_{H\left(2^j x-2k\right)} + \phi_{H\left(2^j x-2k-1\right)}$$
$$\psi_{H\left(2^{j-1} x-k\right)} = \phi_{H\left(2^j x-2k\right)} - \phi_{H\left(2^j x-2k-1\right)}$$

Finally, the equation (2.16) and (2.17) can be derived by replacing $2^{j-1}x - k$ by x. These equations (relations) are called two-scale relation.

$$\phi_{H(x)} = \phi_{H(2x)} + \phi_{H(2x-1)} \tag{2.16}$$

$$\psi_{H(x)} = \phi_{H(2x)} - \phi_{H(2x-1)} \tag{2.17}$$

As mentioned earlier, the general two-scale relation is described as follows.

$$\phi_{(x)} = \sum_k p_k \cdot \phi_{(2x-k)} \tag{2.19}$$

$$\psi_{(x)} = \sum_k q_k \cdot \phi_{(2x-k)} \tag{1.13}$$

The sequences of p_k and q_k are as follows for the Haar wavelet.

$$p_0 = 1, p_1 = 1,\ p_k = 0,\ k \neq 0,1$$

$$q_0 = 1, q_1 = -1,\ q_k = 0,\ k \neq 0,1$$

3 Two-scale relation

3.1 Support

The two scale relation can be described as Equation (3.1).

$$\phi_{(x)} = \sum_k p_k \cdot \phi_{(2x-k)} \tag{3.1}$$

$\operatorname{supp} f = [a \quad b]$ means that $f_{(x)}$ has non-zero value in the range of $x = [a \quad b]$ and it is called support of function *f*.

Here, if $\operatorname{supp}\phi = [a \quad b]$ and $p_k \neq 0 \quad k = 0,1,\cdots,L$,

Leftmost element $p_0 \cdot \phi_{(2x)}$ in order not to be zero $x \in \left[\frac{a}{2} \quad \frac{b}{2}\right]$

Rightmost element $p_L \cdot \phi_{(2x-L)}$ in order not to be zero $x \in \left[\frac{a+L}{2} \quad \frac{b+L}{2}\right]$

Therefore, $\operatorname{supp}\phi = \left[\frac{a}{2} \quad \frac{b+L}{2}\right]$.

Since $\operatorname{supp}\phi$ is assumed to be $[a \quad b]$, $a = 0, b = L$.

Thus, $\operatorname{supp}\phi = [0 \quad L]$.

As for $\psi_{(x)} = \sum_k q_k \cdot \phi_{(2x-k)}$;

Here, if $q_k \neq 0 \quad k = M,1,\cdots,N$,

Leftmost element $q_M \cdot \phi_{(2x-M)}$ in order not to be zero $x \in \left[\frac{M}{2} \quad \frac{M+L}{2}\right]$

Rightmost element $q_N \cdot \phi_{(2x-N)}$ in order not to be zero $x \in \left[\frac{N}{2} \quad \frac{N+L}{2}\right]$

Thus, $\operatorname{supp}\psi = \left[\frac{M}{2} \quad \frac{N+L}{2}\right]$.

3.2 Multi-scale relation

From Equation (3.1), followings can be derived.

$$
\begin{aligned}
\phi_{(x)} &= \sum_{\ell} p_{\ell} \cdot \phi_{(2x-\ell)} \\
&= \sum_{\ell} p_{\ell} \cdot \sum_{k} p_{k} \cdot \phi_{(2(2x-\ell)-k)} \\
&= \sum_{\ell} \sum_{k} p_{\ell} \cdot p_{k} \cdot \phi_{\left(2^2 x - 2\ell - k\right)} \\
&= \sum_{\ell} \sum_{k} p_{\ell} \cdot p_{k-2\ell} \cdot \phi_{\left(2^2 x - k\right)} \qquad (2\ell + k \to k) \\
&= \sum_{k} \sum_{\ell} p_{\ell} \cdot p_{k-2\ell} \cdot \phi_{\left(2^2 x - k\right)}
\end{aligned}
$$

By repeating this procedure, Equation (3.9) is obtained with Equation (3.8).

$$
p_k^{(j)} = \sum_{\ell} p_{k-2\ell} \cdot p_{\ell}^{(j-1)}, \qquad p_k^{(1)} = p_k \tag{3.8}
$$

$$
\phi_{(x)} = \sum_{k} p_k^{(j)} \cdot \phi_{\left(2^j x - k\right)} \tag{3.9}
$$

Thus, internally dividing points between integer points can be calculated with Equation (3.11)

$$
\phi_{\left(\frac{n}{2^j}\right)} = \sum_{k} p_k^{(j)} \cdot \phi_{(n-k)} \tag{3.11}
$$

Internally dividing points of $f_{j(x)}$ can be calculated with the same procedure as follows.

$$
\begin{aligned}
f_{j(x)} &= \sum_{\ell} c_{\ell}^{(j)} \cdot \phi_{\left(2^j x - \ell\right)} \\
&= \sum_{\ell} c_{\ell}^{(j)} \cdot \sum_{k} p_k \cdot \phi_{\left(2\left(2^j x - \ell\right) - k\right)} \\
&= \sum_{\ell} c_{\ell}^{(j)} \cdot \sum_{k} p_k \cdot \phi_{\left(2^{j+1} x - 2\ell - k\right)} \\
&= \sum_{\ell} c_{\ell}^{(j)} \cdot \sum_{k} p_{k-2\ell} \cdot \phi_{\left(2^{j+1} x - k\right)} \qquad (2\ell + k \to k) \\
&= \sum_{k} \sum_{\ell} c_{\ell}^{(j)} \cdot p_{k-2\ell} \cdot \phi_{\left(2^{j+1} x - k\right)}
\end{aligned}
$$

With $c_k^{(j+1)}$ defined as Equation (3.16), $f_{j(x)}$ can be calculated as Equation (3.17)

$$
c_k^{(j+1)} = \sum_{\ell} c_{\ell}^{(j)} \cdot p_{k-2\ell} \tag{3.16}
$$

$$
\begin{aligned}
f_{j(x)} &= \sum_{k} c_k^{(j+1)} \cdot \phi_{\left(2^{j+1} x - k\right)} \\
&= \sum_{k} c_k^{(j+\ell)} \cdot \phi_{\left(2^{j+\ell} x - k\right)}
\end{aligned} \tag{3.17}
$$

Thus, internally dividing points of $f_{j(x)}$ can be calculated as follows.

$$
f_{j\left(\frac{n}{2^{j+\ell}}\right)} = \sum_{k} c_k^{(j+\ell)} \cdot \phi_{(n-k)}
$$

The sequence $c_k^{(j+\ell)}$ is calculated recursively as Equation (3.18).

$$c_k^{(j+i)} = \sum_{\ell} c_\ell^{(j+i-1)} \cdot p_{k-2\ell} \tag{3.16}$$

Internally dividing points of $g_{j(x)}$ can be calculated with the same procedure as follows.

$$\begin{aligned} g_{j(x)} &= \sum_{\ell} d_\ell^{(j)} \cdot \psi_{\left(2^j x-\ell\right)} \\ &= \sum_{\ell} d_\ell^{(j)} \cdot \sum_k q_k \cdot \phi_{\left(2\left(2^j x-\ell\right)-k\right)} && (\,\mathit{from\ Equation}\,(1.13)) \\ &= \sum_{\ell} d_\ell^{(j)} \cdot \sum_k q_k \cdot \phi_{\left(2^{j+1} x-2\ell-k\right)} \\ &= \sum_{\ell} d_\ell^{(j)} \cdot \sum_k q_{k-2\ell} \cdot \phi_{\left(2^{j+1} x-k\right)} && (2\ell+k \to k) \\ &= \sum_k \sum_{\ell} d_\ell^{(j)} \cdot q_{k-2\ell} \cdot \phi_{\left(2^{j+1} x-k\right)} \end{aligned}$$

With $d_k^{(j+i)}$ defined as Equation (3.19), $g_{j(x)}$ can be calculated as Equation (3.20)

$$d_k^{(j+i)} = \sum_{\ell} d_\ell^{(j+i-1)} \cdot q_{k-2\ell} \tag{3.19}$$

$$g_{j(x)} = \sum_k d_k^{(j+\ell)} \cdot \phi_{\left(2^{j+\ell} x-k\right)} \tag{3.20}$$

Thus, internally dividing points of $g_{j(x)}$ can be calculated as follows.

$$g_{j\left(\frac{n}{2^{j+\ell}}\right)} = \sum_k d_k^{(j+\ell)} \cdot \phi_{(n-k)}$$

By replacing $d_k^{(j)}$ by q_k, $g_{j(x)}$ becomes $\psi_{(x)}$. Therefore internally dividing points of $\psi_{(x)}$ can be calculated as follows.

$$\psi_{\left(\frac{n}{2^j}\right)} = \sum_k q_k^{(j)} \cdot \phi_{(n-k)}$$

$$q_k^{(j)} = \sum_{\ell} q_\ell^{(j-1)} \cdot q_{k-2\ell}\ , \quad q_k^{(1)} = q_k$$

As mentioned in chapter 1, (j+1)-th rank has half as much data points as (j)-th rank has. Therefore, the sampling rate for (j+1)-th rank becomes twice as long as (j)-th rank. In order to make the sampling rate the same as the sampling rate of the original signal, internally dividing points need to be calculated with the equations derived here.

3.3 Decomposition algorism

$\phi_{(x)}$ has the relation as follows from a mathematical background.

$$\phi_{(2x-\ell)} = \frac{1}{2}\sum_k \left(g_{2k-\ell} \cdot \phi_{(x-k)} + h_{2k-\ell} \cdot \psi_{(x-k)} \right) \tag{3.22}$$

$f_{j(x)}$ can be derived as follows from Equation (1.15) with Equation (3.22).

$$\begin{aligned}
f_{j(x)} &= \sum_\ell c_\ell^{(j)} \cdot \phi_{\left(2^j x-\ell\right)} \\
&= \sum_\ell c_\ell^{(j)} \cdot \frac{1}{2} \sum_k \left(g_{2k-\ell} \cdot \phi_{\left(2^{j-1} x-k\right)} + h_{2k-\ell} \cdot \psi_{\left(2^{j-1} x-k\right)} \right) \\
&= \sum_k \sum_\ell \left(\frac{1}{2} c_\ell^{(j)} \cdot g_{2k-\ell} \cdot \phi_{\left(2^{j-1} x-k\right)} + \frac{1}{2} c_\ell^{(j)} \cdot h_{2k-\ell} \cdot \psi_{\left(2^{j-1} x-k\right)} \right) \\
&= \sum_k \left\{ \frac{1}{2} \sum_\ell \left(c_\ell^{(j)} \cdot g_{2k-\ell} \right) \right\} \cdot \phi_{\left(2^{j-1} x-k\right)} + \sum_k \left\{ \frac{1}{2} \sum_\ell \left(c_\ell^{(j)} \cdot h_{2k-\ell} \right) \right\} \cdot \psi_{\left(2^{j-1} x-k\right)}
\end{aligned}$$

On the other hand, from Equation (1.14),

$$\begin{aligned}
f_{j(x)} &= f_{j-1(x)} + g_{j-1(x)} \\
&= \sum_k c_k^{(j-1)} \cdot \phi_{\left(2^{j-1} x-k\right)} + \sum_k d_k^{(j-1)} \cdot \psi_{\left(2^j x-k\right)}
\end{aligned}$$

By comparing these two equations, the following decomposition algorism is obtained.

$$\begin{cases} c_k^{(j-1)} = \dfrac{1}{2} \sum_\ell c_\ell^{(j)} \cdot g_{2k-\ell} \\ d_k^{(j-1)} = \dfrac{1}{2} \sum_\ell c_\ell^{(j)} \cdot h_{2k-\ell} \end{cases} \tag{3.23}$$

3.4 Reconstitution algorism

$f_{j(x)}$ can be derived as follows from Equation (1.15).

$$\begin{aligned}
f_{j(x)} &= \sum_k c_k^{(j)} \cdot \phi_{\left(2^j x-k\right)} \\
&= \sum_\ell c_\ell^{(j-1)} \cdot \phi_{\left(2^{j-1} x-\ell\right)} + \sum_\ell d_\ell^{(j-1)} \cdot \psi_{\left(2^j x-\ell\right)} && (\textit{from Equation}\,(1.14)) \\
&= \sum_\ell c_\ell^{(j-1)} \cdot \sum_k p_k \cdot \phi_{\left(2\left(2^{j-1} x-\ell\right)-k\right)} + \sum_\ell d_\ell^{(j-1)} \cdot \sum_k q_k \cdot \phi_{\left(2\left(2^{j-1} x-\ell\right)-k\right)} && (\textit{from Equation}\,(3.1)\,\textit{and}\,(1.13)) \\
&= \sum_\ell c_\ell^{(j-1)} \cdot \sum_k p_k \cdot \phi_{\left(2^j x-2\ell-k\right)} + \sum_\ell d_\ell^{(j-1)} \cdot \sum_k q_k \cdot \phi_{\left(2^j x-2\ell-k\right)} \\
&= \sum_\ell c_\ell^{(j-1)} \cdot \sum_k p_{k-2\ell} \cdot \phi_{\left(2^j x-k\right)} + \sum_\ell d_\ell^{(j-1)} \cdot \sum_k q_{k-2\ell} \cdot \phi_{\left(2^j x-k\right)} && (2\ell + k \to k) \\
&= \sum_k \sum_\ell c_\ell^{(j-1)} \cdot p_{k-2\ell} \cdot \phi_{\left(2^j x-k\right)} + \sum_k \sum_\ell d_\ell^{(j-1)} \cdot q_{k-2\ell} \cdot \phi_{\left(2^j x-k\right)} \\
&= \sum_k \left(\sum_\ell c_\ell^{(j-1)} \cdot p_{k-2\ell} + \sum_\ell d_\ell^{(j-1)} \cdot q_{k-2\ell} \right) \cdot \phi_{\left(2^j x-k\right)}
\end{aligned}$$

Therefore, the following reconstitution algorism can be obtained.

$$c_k^{(j)} = \sum_\ell c_\ell^{(j-1)} \cdot p_{k-2\ell} + \sum_\ell d_\ell^{(j-1)} \cdot q_{k-2\ell} \tag{3.24}$$

3.5 Decomposition of periodic signal

The original signal shown as follows is considered.

$$f_{(i)} = \sin\left(2\pi\frac{\Delta t}{T}\cdot i\right) = \sin\left(2\pi\frac{\Delta t}{n\cdot\Delta t}\cdot i\right) = \sin\left(2\pi\frac{i}{n}\right)$$
$$= \sin(2\pi\cdot\Delta t\cdot F\cdot i)$$

where, Δt is time interval (sec), T is period (sec) ($= n\cdot\Delta t$), and F is resonant frequency.

Since $g_{j(x)} = \sum_{\ell} d_{\ell}^{(j)}\cdot\psi_{(2^j x-\ell)}$, time interval for $g_{j(x)}$ is $\Delta t_j = \Delta t\cdot 2^{-j}$.

The Nyquist frequency for $g_{j(x)}$ is $1/(2\cdot\Delta t\cdot 2^{-j})$.

The rank when the frequency of the signal coincides with the Nyquist frequency is;

$$\frac{1}{2^{-j+1}\cdot\Delta t} = \frac{1}{n\cdot\Delta t}$$
$$\Leftrightarrow n = 2^{-j+1}$$
$$\Leftrightarrow j = -\log_2\frac{n}{2}$$

Therefore, the j-th rank has the sine wave of which frequency is $1/(2^{-j+1}\cdot\Delta t)$.

The sine wave of whish frequency is inbetween $1/(2^{-j+1}\cdot\Delta t)$ and $1/(2^{-j}\cdot\Delta t)$ is decomposed into j-th and (j+1)-th rank. In other words, the energy of the signal is dispersed into two ranks.

For example, Figure 1 shows the result of $\sin(2\pi\cdot 3.125\cdot t)$, of which resonant frequency is 3.125Hz. Since $\Delta t = 0.01$ sec, the Nyquist frequency of -4 rank is $1/(2^{4+1}\cdot 0.01) = 3.125$. Therefore, the sine wave was decomposed as g_{-4} as shown in the figure. The remaining f_{-4} is very small, but relatively large components are observed at both ends. They are called “edge effect”, which is because the signal suddenly starts and ends at both ends. If the signal starts and ends smoothly at both ends, the effect becomes very small.

Figure 2shows the result of $\sin(2\pi\cdot 2.34375\cdot t)$, of which resonant frequency is 2.34375Hz. The Nyquist frequency of -4 rank is $1/(2^{4+1}\cdot 0.01) = 1.5625$. Therefore, the sine wave was just the median of Nyquist frequencies for -4 and -5 ranks. The sine wave was decomposed as g_{-4} and g_{-5} as shown in the figure. g_{-5} is relatively larger than g_{-4}, but it depends on the mother wavelet. The cardinal B-spline with the order of 4 was used for this case. It can be said that sine wave inbetween two ranks tends to be decomposed as lower rank component with the cardinal B-spline. The remaining component f_{-5} is very small except the edge effect.

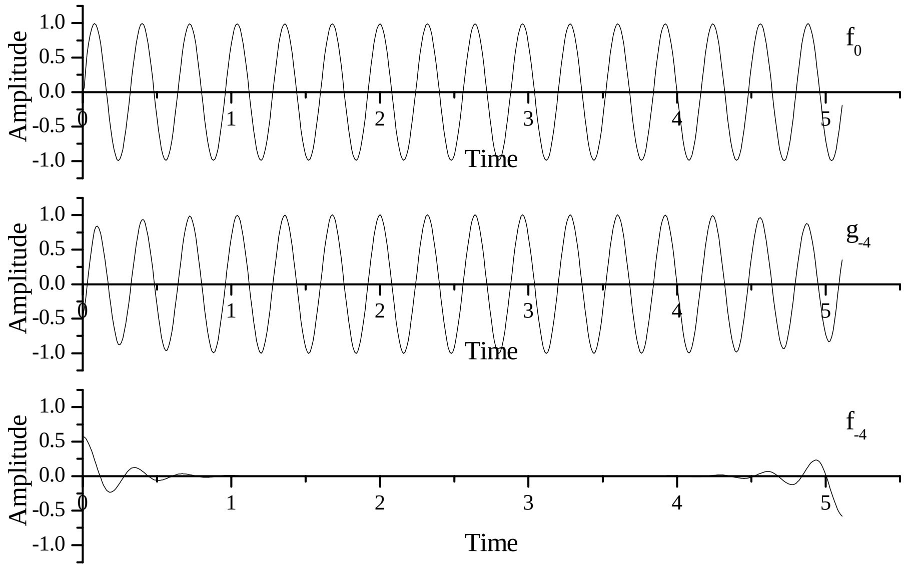


Figure 1 $\sin(2\pi \cdot 3.125 \cdot t)$

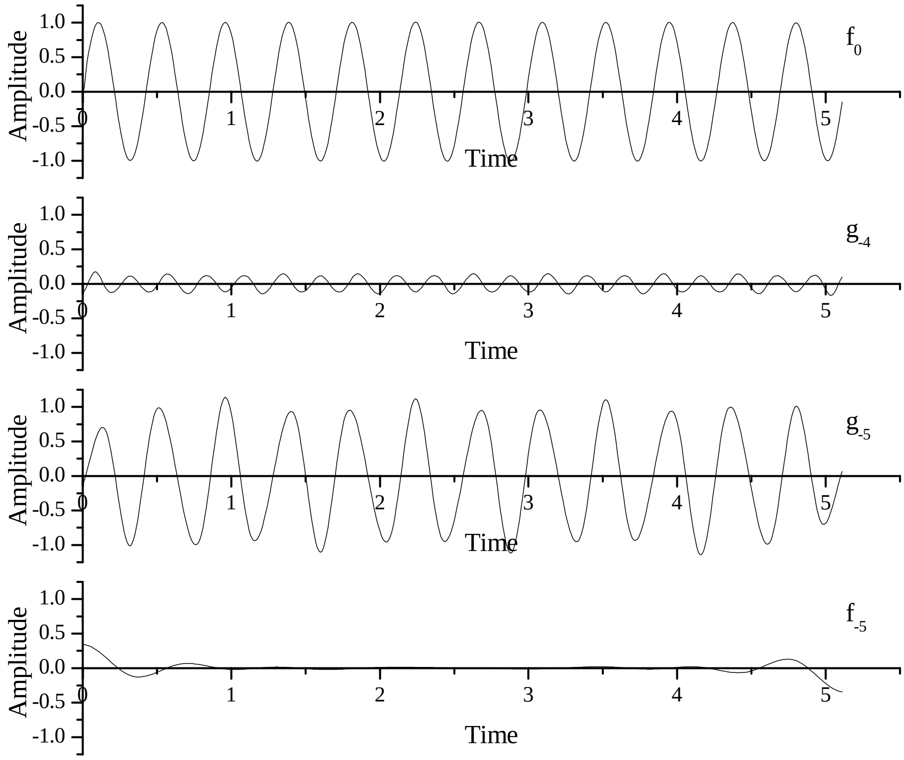


Figure 2 $\sin(2\pi \cdot 2.34375 \cdot t)$

3.6 Uniqueness of the decomposition –Orthogonality

The scaling function holds the following two-scale relation.

$$\phi_{(x)} = \sum_k p_k \cdot \phi_{(2x-k)} \tag{3.1}$$

Since $f_{j(x)}$ is shown as Equation (1.15), the scaling function for the j-th rank is $\phi_{\left(2^j x-k\right)}$.

$$f_{j(x)} = \sum_k c_k^{(j)} \cdot \phi_{\left(2^j x-k\right)} \tag{1.15}$$

The space V_j of $\phi_{\left(2^j x-k\right)}$, the space of the function that is the linear combination of $\phi_{\left(2^j x-k\right)}$, includes $f_{j(x)}$.

$$f_{j(x)} \in V_j$$

From Equation (3.1), $\phi_{\left(2^j x\right)} = \sum_k p_k \cdot \phi_{\left(2^{j+1} x-k\right)}$ by replacing x by $2^j x$. Since $\phi_{\left(2^j x\right)}$ is the linear combination of $\phi_{\left(2^{j+1} x-k\right)}$, $\phi_{\left(2^j x\right)} \in V_{j+1}$. Thus, $\phi_{\left(2^j x\right)}$ is also a part of V_{j+1}. Therefore, following system of sets is obtained.

$$\cdots \subset V_{j-1} \subset V_j \subset V_{j+1} \cdots$$

On the other hand, the mother wavelet has following two relations.

$$\psi_{(x)} = \sum_k q_k \cdot \phi_{(2x-k)} \tag{1.13}$$

$$g_{j(x)} = \sum_k d_k^{(j)} \cdot \psi_{\left(2^j x-k\right)} \tag{1.9}$$

Therefore, the space W_j of $\psi_{\left(2^j x-k\right)}$, the space of the function that is the linear combination of $\psi_{\left(2^j x-k\right)}$, includes $g_{j(x)}$.

$$g_{j(x)} \in W_j$$

Since $f_{j(x)} = f_{j-1(x)} + g_{j-1(x)}$, $V_j = V_{j-1} \oplus W_{j-1}$. In other words, W_{j-1} is compliment of V_{j-1} in the space of V_j. By repeating this procedure, the following relation is obtained.

$$V_0 = W_{-1} \oplus W_{-2} \oplus \cdots$$

$$f_{(x)} = \sum_j \sum_k d_k^{(j)} \psi_{\left(2^j x-k\right)} \tag{1.8}$$

In order to hold the relation of Equation (1.8) uniquely, $\psi_{(x)}$ must be basis function. Two-scale sequences p_k and q_k are defined so that $\psi_{(x)}$ becomes basis function. The details will be mentioned in Section 5.4, but the number of elements of p_k and q_k are not finite for the biorthogonal wavelet. Therefore, a finite number of elements is usually used as an approximation. Thus, to be precise, the components decomposed with the biorthogonal wavelet are not orthogonal to each other. It is, however, generally known that the result is practically very well, if an appropriate number of elements to be considered are applied.

4 Sub-band decomposition

The decomposition and reconstitution algorisms can be computed easily with digital-filtering technique.

4.1 Upsampling

The upsampling $c_k^{\uparrow}$ is the calculation to add zero data to every other step as follows.

$$\begin{aligned} c_{2k}^{\uparrow} &= c_k \\ c_{2k+1}^{\uparrow} &= 0 \end{aligned} \qquad (7.23)$$

The upsampling of the polynomial $X_{(z)}$ shown in Equation (7.24) is in the fashion shown in Equation (7.25).

$$X_{(z)} = \sum_k c_k \cdot z^k \qquad (7.24)$$

$$\begin{aligned} \sum_k c_k^{\uparrow} \cdot z^k &= \sum_k \left(c_{2k} \cdot z^{2k} + c_{2k+1} \cdot z^{2k+1} \right) \qquad (\textit{odd and even parts}) \\ &= \sum_k c_k \cdot z^{2k} \\ &= X_{\left(z^2\right)} \end{aligned}$$

The number of data points of $c_k^{\uparrow}$, $n_c^{\uparrow}$ is the double of the number of data points of c_k, n_c.

$$n_c^{\uparrow} = 2n_c$$

4.2 Downsampling

The downsampling $c_k^{\downarrow}$ is the calculation to take every other data as follows.

$$c_k^{\downarrow} = c_{2k} \qquad (7.25)$$

For the polynomial shown in Equation (7.24), the following equation can be obtained.

$$\begin{aligned} \frac{1}{2}\left[X_{\left(z^{1/2}\right)} + X_{\left(-z^{1/2}\right)} \right] &= \frac{1}{2}\sum_k c_k \left[\left(z^{1/2}\right)^k + \left(-z^{1/2}\right)^k \right] \\ &= \frac{1}{2}\sum_k c_{2k} \left[\left(z^{1/2}\right)^{2k} + \left(-z^{1/2}\right)^{2k} \right] + \frac{1}{2}\sum_k c_{2k+1} \left[\left(z^{1/2}\right)^{2k+1} + \left(-z^{1/2}\right)^{2k+1} \right] \ (\textit{odd and evenparts}) \\ &= \frac{1}{2}\sum_k c_{2k} \left[z^k + z^k \right] + \frac{1}{2}\sum_k c_{2k+1} \left[\left(z^{1/2}\right)^{2k+1} - \left(z^{1/2}\right)^{2k+1} \right] \\ &= \frac{1}{2}\sum_k c_{2k} \left[z^k + z^k \right] \\ &= \sum_k c_{2k} \cdot z^k \\ &= \sum_k c_k^{\downarrow} \cdot z^k \end{aligned}$$

The number of data points of $c_k^{\downarrow}$, $n_c^{\downarrow}$ is the half of the number of data points of c_k, n_c.

$$n_c^{\downarrow} = \mathrm{int}\left(\frac{n_c}{2} + 0.5\right)$$

4.3 Discrete convolution

Discrete convolution function $(a*b)_k$ is defined as Equation (7.27).

$$(a*b)_k = \sum_{\ell} a_{k-\ell} \cdot b_{\ell} \tag{7.27}$$

The discrete convolution is the product of two polynomials as Equation (7.28).

$$A_{(z)} = \sum_{i=i_a}^{i_a+n_a-1} a_k \cdot z^k$$

$$B_{(z)} = \sum_{i=i_b}^{i_b+n_b-1} b_k \cdot z^k$$

$$\begin{aligned}
\sum_k (a*b)_k \cdot z^k &= \sum_k \sum_{\ell} a_{k-\ell} \cdot b_{\ell} \cdot z^k \\
&= \sum_k \sum_{\ell} a_{k-\ell} \cdot b_{\ell} \cdot z^{k-\ell+\ell} \\
&= \sum_k \sum_{\ell} a_{k-\ell} \cdot z^{k-\ell} \cdot b_{\ell} \cdot z^{\ell} \\
&= \sum_k \sum_{\ell} a_k \cdot z^k \cdot b_{\ell} \cdot z^{\ell} \qquad (k-\ell \to k) \\
&= \sum_k a_k \cdot z^k \sum_{\ell} \cdot b_{\ell} \cdot z^{\ell} \\
&= A_{(z)} \cdot B_{(z)}
\end{aligned} \tag{7.28}$$

$$A_{(z)} \cdot B_{(z)} = \sum_{i=i_a+i_b}^{i_a+i_b+n_a+n_b-2} (a*b)_k \cdot z^k$$

4.4 Reconstitution algorism

$$\begin{aligned}
\left(a*b^{\uparrow}\right)_k &= \sum_{\ell} a_{k-\ell} \cdot b_{\ell}^{\uparrow} \\
&= \sum_{\ell} a_{k-2\ell} \cdot b_{2\ell}^{\uparrow} + \sum_{\ell} a_{k-2\ell-1} \cdot b_{2\ell+1}^{\uparrow} \ (\textit{odd and even parts}) \\
&= \sum_{\ell} a_{k-2\ell} \cdot b_{2\ell}^{\uparrow} \\
&= \sum_{\ell} a_{k-2\ell} \cdot b_{\ell}
\end{aligned}$$

The reconstitution algorism is as Equation (3.24)

$$c_k^{(j)} = \sum_{\ell} c_{\ell}^{(j-1)} \cdot p_{k-2\ell} + \sum_{\ell} d_{\ell}^{(j-1)} \cdot q_{k-2\ell} \tag{3.24}$$

Therefore, the reconstitution algorism can be calculated as follows.

$$c_k^{(j)} = \left(p * c^{(j-1)\uparrow}\right)_k + \left(q * d^{(j-1)\uparrow}\right)_k$$

4.5 Decomposition algorism

$$\begin{aligned}(a*b)_k^{\downarrow} &= (a*b)_{2k} \\ &= \sum_{\ell} a_{2k-\ell} \cdot b_{\ell}\end{aligned}$$

The decomposition algorism is as Equation (1.16).

$$\begin{aligned}c_k^{(j-1)} &= \frac{1}{2}\sum_{\ell\in Z} g_{2k-\ell} \cdot c_{\ell}^{(j)} \\ d_k^{(j-1)} &= \frac{1}{2}\sum_{\ell\in Z} h_{2k-\ell} \cdot c_{\ell}^{(j)}\end{aligned} \tag{1.16}$$

Therefore, the decomposition algorism can be calculated as follows.

$$\begin{aligned}c_k^{(j-1)} &= \frac{1}{2}\left(g * c^{(j)}\right)_k^{\downarrow} \\ d_k^{(j-1)} &= \frac{1}{2}\left(h * c^{(j)}\right)_k^{\downarrow}\end{aligned}$$

5 Sequences p_k, q_k, g_k, and h_k, and interpolating sequence $c_k^{(0)}$

5.1 Required minimum mathematical background

5.1.1 Auto-correlation function

The auto-correlation function of *f*, *F* is shown as Equation (6.18).

$$F_{(x)} = \int_{-\infty}^{\infty} f_{(x+y)} \cdot \bar{f}_{(y)} dy \tag{6.18}$$

On the other hand, the Fourier transform and the Inverse Fourier transform are shown in Equation (6.1) and (6.2) respectively.

$$\hat{f}_{(\omega)} = \int_{-\infty}^{\infty} e^{-i\omega x} \cdot f_{(x)} \cdot dx \tag{6.1}$$

$$f_{(x)} = \frac{1}{2\pi} \int_{-\infty}^{\infty} e^{i\omega x} \cdot \hat{f}_{(\omega)} \cdot d\omega \tag{6.2}$$

Following equation is derived from Equation (6.18) with replacing "*y*" by "*y-x*" (*dy =dy*) and Equation (6.1) and (6.2).

$$\begin{aligned}
F_{(x)} &= \int_{-\infty}^{\infty} \bar{f}_{(y-x)} \cdot f_{(y)} \cdot dy \\
&= \int_{-\infty}^{\infty} \left[\frac{1}{2\pi} \int_{-\infty}^{\infty} e^{-i\omega(y-x)} \cdot \bar{\hat{f}}_{(\omega)} \cdot d\omega \right] \cdot f_{(y)} \cdot dy \\
&= \frac{1}{2\pi} \int_{-\infty}^{\infty} \left[\int_{-\infty}^{\infty} e^{-i\omega y} \cdot f_{(y)} \cdot dy \right] \cdot \bar{\hat{f}}_{(\omega)} \cdot e^{i\omega x} \cdot d\omega \\
&= \frac{1}{2\pi} \int_{-\infty}^{\infty} \hat{f}_{(\omega)} \cdot \bar{\hat{f}}_{(\omega)} \cdot e^{i\omega x} \cdot d\omega \\
&= \frac{1}{2\pi} \int_{-\infty}^{\infty} \left| \hat{f}_{(\omega)} \right|^2 \cdot e^{i\omega x} \cdot d\omega \\
&= \frac{1}{2\pi} \int_{-\infty}^{\infty} \hat{F}_{(x)} \cdot e^{i\omega x} \cdot d\omega
\end{aligned}$$

Thus, Equation (6.19) is obtained.

$$\hat{F}_{(x)} = \left| \hat{f}_{(\omega)} \right|^2 \tag{6.19}$$

The auto-correlation function for the scaling function ϕ is shown in Equation (7.5).

$$F_{\phi(x)} = \int_{-\infty}^{\infty} \phi_{(x+y)} \cdot \bar{\phi}_{(y)} dy \tag{7.5}$$

Equation (7.6) is obtained from Equation (6.19).

$$\hat{F}_{\phi(x)} = \left| \hat{\phi}_{(\omega)} \right|^2 \tag{7.6}$$

If ϕ is compactly supported (ϕ is not infinite sequence), following Euler-Forbenius sequence can be defined.

$$E_{\phi(z)} = \sum_{k=-n}^{n} F_{\phi(k)} \cdot z^k \qquad (7.8)$$

5.1.2 Laurent polynomial

The sequences p_k, q_k, g_k, and h_k for the wavelet transform can be defined as Laurent polynomial as follow.

$$P_{(z)} = \frac{1}{2}\sum_{k\in Z} p_k \cdot z^k \qquad (7.1)$$

$$Q_{(z)} = \frac{1}{2}\sum_{k\in Z} q_k \cdot z^k$$
$$G_{(z)} = \frac{1}{2}\sum_{k\in Z} g_k \cdot z^k \qquad (7.11)$$
$$H_{(z)} = \frac{1}{2}\sum_{k\in Z} h_k \cdot z^k$$

They hold the following equations for the two-scale relation.

$$G_{(z)} = \frac{E_{\phi(z)}}{E_{\phi\left(z^2\right)}} \overline{P}_{(z)}$$
$$Q_{(z)} = -z^{2m-1} \cdot E_{\phi(-z)} \cdot \overline{P}_{(-z)} \qquad (7.12)$$
$$H_{(z)} = -z^{-2m+1} \cdot \frac{P_{(-z)}}{E_{\phi\left(z^2\right)}}$$
$$z = e^{-i\omega/2}$$

where, m is a given integer. Note that the complex conjugate of $P_{(z)}$, $\overline{P}_{(z)}$ is in the following fashion.

$$\overline{P}_{(z)} = \frac{1}{2}\sum_{k\in Z} \overline{p}_k \cdot z^{-k}$$

5.2 Orthogonal wavelet (Daubechies wavelet)

5.2.1 Fundamental characteristics

If a mother wavelet $\psi_{(x)}$ is orthogonal to its integer translate as follows, $\psi_{(x)}$ is called orthogonal wavelet.

$$\left\langle \psi \mid \psi_{(\cdot-n)} \right\rangle = \int_{-\infty}^{\infty} \psi_{(x)} \cdot \overline{\psi}_{(x-n)} dx \propto \delta_{n,0}$$

where, δ is called delta function, which satisfies the following equations.

$$\delta_{x,y} = \begin{cases} 1 & x = y \\ 0 & x \neq y \end{cases}$$

The orthogonal mother wavelet $\psi_{(x)}$ is usually generated from an orthogonal scaling function $\phi_{(x)}$ with Equation (1.13).

$$\psi_{(x)} = \sum_k q_k \cdot \phi_{(2x-k)} \tag{1.13}$$

$$\left\langle \phi \mid \phi_{(-n)} \right\rangle = \int_{-\infty}^{\infty} \phi_{(x)} \cdot \overline{\phi}_{(x-n)} dx = \int_{-\infty}^{\infty} \phi_{(x)} \cdot \overline{\phi}_{(x)} dx = \|\phi\|^2$$

Usually, following normalization is applied.

$$\|\phi\|^2 = 1 \tag{8.1}$$

Thus, $\psi_{(x)}$ and $\phi_{(x)}$ become orthonormal as Equation (8.2) and (8.3).

$$\left\langle \psi \mid \psi_{(-n)} \right\rangle = \delta_{n,0} \tag{8.2}$$

$$\left\langle \phi \mid \phi_{(-n)} \right\rangle = \delta_{n,0} \tag{8.3}$$

Therefore, the following equation can be derived from $F_{\phi(x)}$ of Equation (7.5) with Equation (8.1)

$$F_{\phi(n)} = \int_{-\infty}^{\infty} \phi_{(x+n)} \cdot \overline{\phi}_{(y)} dy = \delta_{n,0}$$

And then, the Euler-Forbenius sequence of Equation (7.8) becomes 1 as Equation (8.7).

$$E_{\phi(z)} = \sum_{k=-n}^{n} F_{\phi(z)} \cdot z^k = F_{\phi(0)} \cdot z^0 = z^0 = 1 \tag{8.7}$$

Finally, Equation (8.13) is obtained from Equation (7.12).

$$\begin{aligned} G_{(z)} &= \overline{P}_{(z)} \\ Q_{(z)} &= -z^{2m-1} \cdot \overline{P}_{(-z)} \\ H_{(z)} &= -z^{-2m+1} \cdot P_{(-z)} \end{aligned} \tag{8.13}$$

where, m is a given integer.

$$\begin{aligned} G_{(z)} &= \overline{P}_{(z)} \\ &= \frac{1}{2} \sum_{k \in Z} \overline{p}_k \cdot z^{-k} \\ &= \frac{1}{2} \sum_{k \in Z} \overline{p}_{-k} \cdot z^{k} \\ &= \frac{1}{2} \sum_{k \in Z} g_k \cdot z^{k} \end{aligned}$$

$$
\begin{aligned}
Q_{(z)} &= -z^{2m-1} \cdot \overline{P}_{(-z)} \\
&= -z^{2m-1} \cdot \frac{1}{2} \sum_{k \in Z} \overline{p}_k \cdot (-z)^{-k} \\
&= -\frac{1}{2} \sum_{k \in Z} (-1)^{-k} \cdot \overline{p}_k \cdot z^{2m-1-k} \\
&= -\frac{1}{2} \sum_{k \in Z} (-1)^{k-2m+1} \cdot \overline{p}_{2m-1-k} \cdot z^{k} \qquad (2m-1-k \to k) \\
&= \frac{1}{2} \sum_{k \in Z} (-1)^{k} \cdot \overline{p}_{2m-1-k} \cdot z^{k} \\
&= \frac{1}{2} \sum_{k \in Z} q_k \cdot z^{k}
\end{aligned}
$$

$$
\begin{aligned}
H_{(z)} &= -z^{-2m+1} \cdot P_{(-z)} \\
&= -z^{-2m+1} \cdot \frac{1}{2} \sum_{k \in Z} p_k \cdot (-z)^{k} \\
&= -\frac{1}{2} \sum_{k \in Z} (-1)^{-k} \cdot p_k \cdot z^{k-2m+1} \\
&= -\frac{1}{2} \sum_{k \in Z} (-1)^{k+2m-1} \cdot p_{k+2m-1} \cdot z^{k} \qquad (k-2m+1 \to k) \\
&= \frac{1}{2} \sum_{k \in Z} (-1)^{k} \cdot p_{k+2m-1} \cdot z^{k} \\
&= \frac{1}{2} \sum_{k \in Z} h_k \cdot z^{k}
\end{aligned}
$$

Therefore, q_k, g_k, and h_k are calculated from p_k with Equation (8.14)

$$
\begin{aligned}
g_k &= \overline{p}_{-k} \\
q_k &= (-1)^k \cdot \overline{p}_{2m-1-k} \\
h_k &= (-1)^k \cdot p_{k+2m-1}
\end{aligned}
\tag{8.14}
$$

$\overline{p}_k$ is equal to p_k if p_k is not complex but real, since $\overline{p}_k$ is the complex conjugate of p_k.

5.3 Get $\phi_{(n)}$

Here, the $\phi_{(n)}$ for the integer points will be calculated.

Since $\phi_{(n)} = \sum_k p_k \cdot \phi_{(2n-k)} \quad n = 0, \cdots, 2m-1$ and $p_k \neq 0 \quad k = 0, 1, \cdots, 2m-1$

$$
\begin{aligned}
\phi_{(0)} &= p_0 \cdot \phi_{(0)} \\
\phi_{(1)} &= p_0 \cdot \phi_{(2)} + p_1 \cdot \phi_{(1)} + p_2 \cdot \phi_{(0)} \\
\phi_{(2)} &= p_0 \cdot \phi_{(4)} + p_1 \cdot \phi_{(3)} + p_2 \cdot \phi_{(2)} + p_3 \cdot \phi_{(1)} + p_4 \cdot \phi_{(0)} \\
&\vdots
\end{aligned}
$$

$$\phi_{(i)} = \sum_{j=ST}^{ED} p_j \cdot \phi_{(2i-j)}$$

$$\vdots$$

$$\phi_{(2m-1)} = \sum_{j=ST}^{ED} p_j \cdot \phi_{(4m-2-j)}$$

Where,

$$0 \le j \ \ and \ \ 2*i - j \le 2m-1 \quad \Leftrightarrow 0 \le j \ \ and \ \ 2\cdot(i-m)+1 \le j$$

Therefore, $ST = \begin{cases} 0 & i \le m-1 \\ 2\cdot(i-m)+1 & i > m-1 \end{cases}$

$$j \le 2m-1 \ and \ 0 \le 2*i - j \quad j \le 2m-1 \ and \ j \le 2*i$$

Therefore, $ED = \begin{cases} 2i & i \le m-1 \\ 2m-1 & i > m-1 \end{cases}$

Therefore,

$$\{\phi\} = [\text{Related to } p_k] \cdot \{\phi\}$$

On the other hand, the following equation is true for an orthogonal wavelet.

$$\sum_k \phi_k = 1$$

Therefore,

$$\begin{Bmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{Bmatrix} = \left(\begin{bmatrix} & & & 0 \\ & [\text{Related to } p_k] & & \vdots \\ & & & 0 \\ 1 & \cdots & \cdots & 1 \end{bmatrix} - \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & \ddots & \ddots & \vdots \\ \vdots & \ddots & 1 & 0 \\ 0 & \cdots & 0 & 0 \end{bmatrix} \right) \cdot \begin{Bmatrix} \phi_1 \\ \vdots \\ \vdots \\ \phi_n \end{Bmatrix}$$

Here, the row for ϕ_0 is neglected since ϕ_0 is always zero because of $\phi_{(0)} = p_0 \cdot \phi_{(0)}$.
By solving this equation, $\phi_{(n)}$ can be obtained.

5.3.1 interpolating sequence $c_k^{(0)}$

The discrete Fourier transform of data sequence $\theta_k \ k = 1,2,\cdots,N$, $\hat{\theta}_k$ is shown in the following fashion.

$$\hat{\theta}_n = \sum_{k=1}^{N} \theta_k \cdot e^{-2\pi i\cdot(k-1)(n-1)/N} \tag{6.33}$$

The discrete inverse Fourier transform of $\hat{\theta}_k$ is shown in the following fashion.

$$\theta_k = \frac{1}{N} \sum_{n=1}^{N} \hat{\theta}_n \cdot e^{2\pi i\cdot(k-1)(n-1)/N} \tag{6.33}$$

The original signal $f_{(x)}$ is shown in the following fashion with the initial sequence (for

0-th rank) $c_k^{(0)}$.

$$f_{(x)} = f_{0(x)} = \sum_{k=1}^{N} c_k^{(0)} \cdot \phi_{(x-k)}$$

Then,

$$\begin{aligned}
\hat{f}_{0(m)} &= \sum_{n=1}^{N} f_{0(n)} \cdot e^{-2\pi i \cdot (m-1)(n-1)/N} \\
&= \sum_{n=1}^{N} \sum_{k=1}^{N} c_k^{(0)} \cdot \phi_{(n-k)} \cdot e^{-2\pi i \cdot (m-1)(n-1)/N} \\
&= \sum_{k=1}^{N} c_k^{(0)} \sum_{n=1}^{N} \phi_{(n-k)} \cdot e^{-2\pi i \cdot (m-1)(n-1)/N} \\
&= \sum_{k=1}^{N} c_k^{(0)} \cdot e^{-2\pi i \cdot (m-1)(k-1)/N} \sum_{n=1}^{N} \phi_{(n-k)} \cdot e^{-2\pi i \cdot (m-1)(n-k)/N} \\
&= \sum_{k=1}^{N} c_k^{(0)} \cdot e^{-2\pi i \cdot (m-1)(k-1)/N} \sum_{n=1}^{N} \phi_{(n-1)} \cdot e^{-2\pi i \cdot (m-1)(n-1)/N} \quad (n-k \to n-1) \\
&= \hat{c}_m^{(0)} \cdot \hat{\phi}_{(n-1)}
\end{aligned}$$

$$\Leftrightarrow \hat{c}_m^{(0)} = \frac{\hat{f}_{0(m)}}{\hat{\phi}_{(n-1)}}$$

Thus, $c_k^{(0)}$ is calculated as the inverse Fourier transform of the equation above.

$$c_k^{(0)} = \frac{1}{N} \sum_{m=1}^{N} \hat{c}_m^{(0)} \cdot e^{2\pi i \cdot (k-1)(m-1)/N} \qquad (6.35)$$

5.3.2 p_k for Daubechies wavelet

Daubechies wavelet is one of the orthogonal wavelets, which holds following relation.

$$\int_{-\infty}^{\infty} x^{\ell} \cdot \psi_{(x)} dx = 0 \qquad \ell = 0,1,\cdots,N-1 \tag{8.15}$$

It is very complicated to calculate p_k mathematically. Therefore, p_k is usually given as a table. Table 1 shows the $p_{(k)}$ to calculate p_k.

$$p_k = \sqrt{2} \cdot p_{(k)}$$

Table 1 $p_{(k)}$ for p_k of Daubechies wavelet

N	2	3	4	6	8	10
P(0)	0.482962913144534	0.332670552950080	0.230377813308890	0.111540743350110	0.054415842243110	0.026670057900550
P(1)	0.836516303737807	0.806891509311090	0.714846570552910	0.494623890398450	0.312871590914320	0.188176800077630
P(2)	0.224143868042013	0.459877502118490	0.630880767929860	0.751133908021100	0.675630736297320	0.527201188931580
P(3)	-0.129409522551260	-0.135011020010250	-0.027983769416860	0.315250351709200	0.585354683654220	0.688459039453440
P(4)	-	-0.085441273882030	-0.187034811719090	-0.226264693965440	-0.015829105256380	0.281172343660570
P(5)	-	0.035226291885710	0.030841381835560	-0.129766867567270	-0.284015542961580	-0.249846424327160
P(6)	-	-	0.032883011666890	0.097501605587320	0.000472484573910	-0.195946274377290
P(7)	-	-	-0.010597401785070	0.027522865530310	0.128747426620490	0.127369340335750
P(8)	-	-	-	-0.031582039317490	-0.017369301001810	0.093057364603550
P(9)	-	-	-	0.000553842201160	-0.044088253930800	-0.071394147166350
P(10)	-	-	-	0.004777257510950	0.013981027917400	-0.029457536821840
P(11)	-	-	-	-0.001077301085310	0.008746094047410	0.033212674059360
P(12)	-	-	-	-	-0.004870352993450	0.003606553566990
P(13)	-	-	-	-	-0.000391740373380	-0.010733175483300
P(14)	-	-	-	-	0.000675449406450	0.001395351747070
P(15)	-	-	-	-	-0.000117476784120	0.001992405295190
P(16)	-	-	-	-	-	-0.000685856694960
P(17)	-	-	-	-	-	-0.000116466855130
P(18)	-	-	-	-	-	0.000093588670320
P(19)	-	-	-	-	-	-0.000013264202890

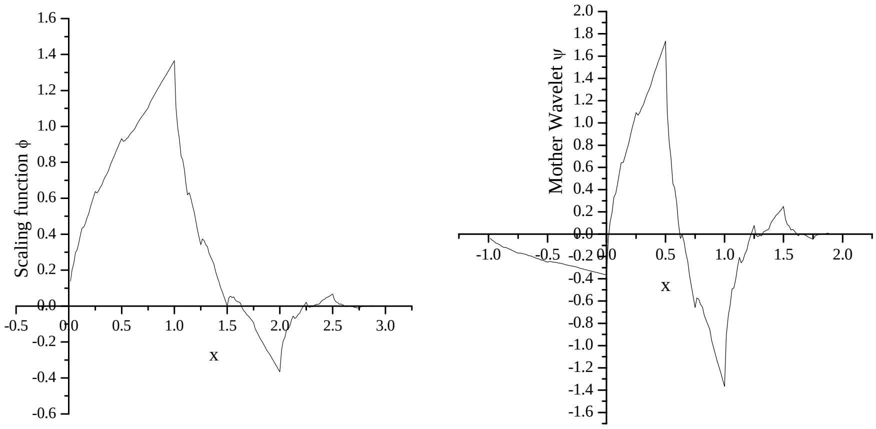
From Equation (8.16), each sequence can be calculated as follows.

$$\begin{aligned} p_k &= \sqrt{2} \cdot p_{(k)} && k = 0,1,\cdots,2N-1 \\ g_k &= p_{-k} && k = 1-2N,\cdots,0 \\ q_k &= (-1)^k \cdot p_{1-k} && k = 2-2N,\cdots,0,1 \\ h_k &= (-1)^k \cdot p_{1+k} && k = -1,0,1,\cdots 2N-2 \end{aligned}$$

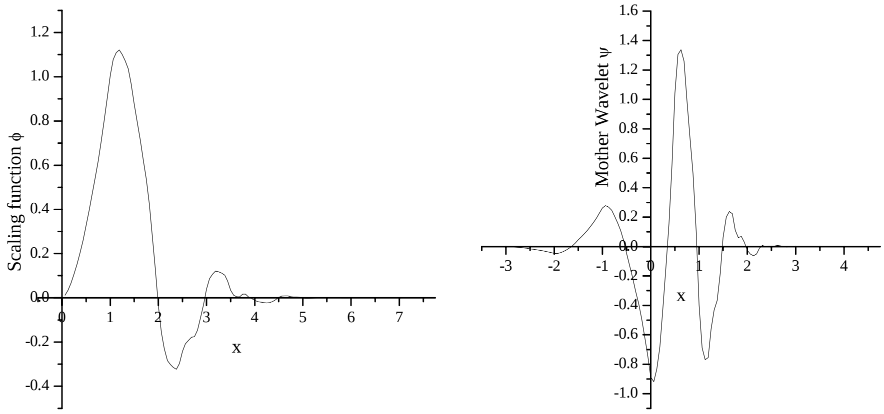
Here,

$$h_k = q_{-k}$$

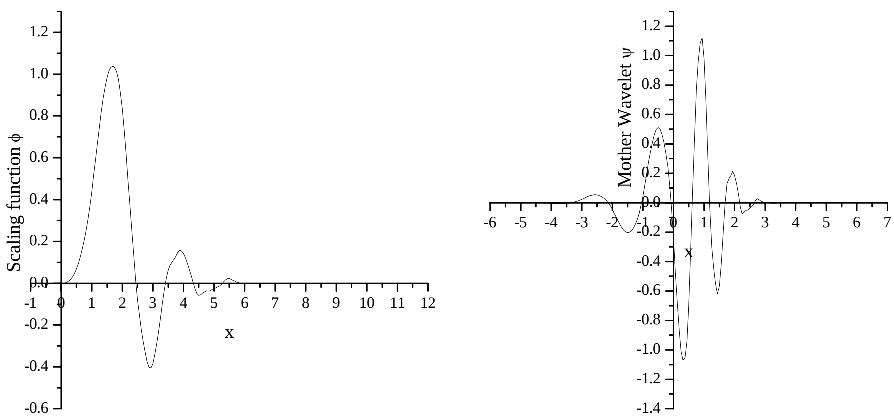
The scaling functions and mother wavelets for Daubechies with N of 2, 4, 6, 8, and 10 are shown in Figure 3.



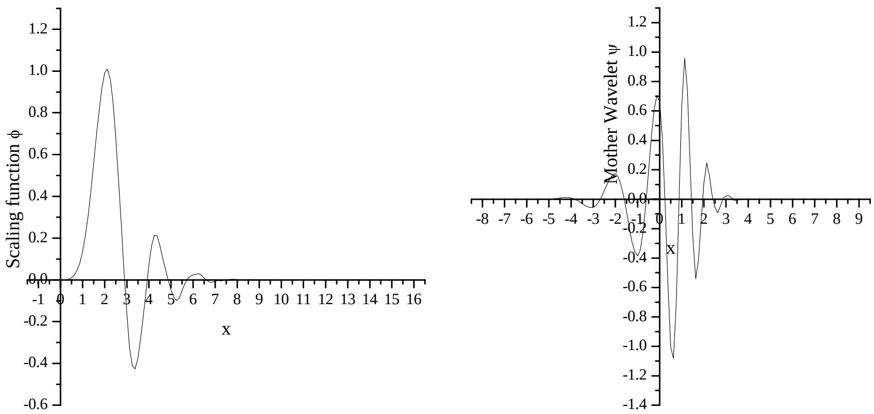
(a) Daubechie 2



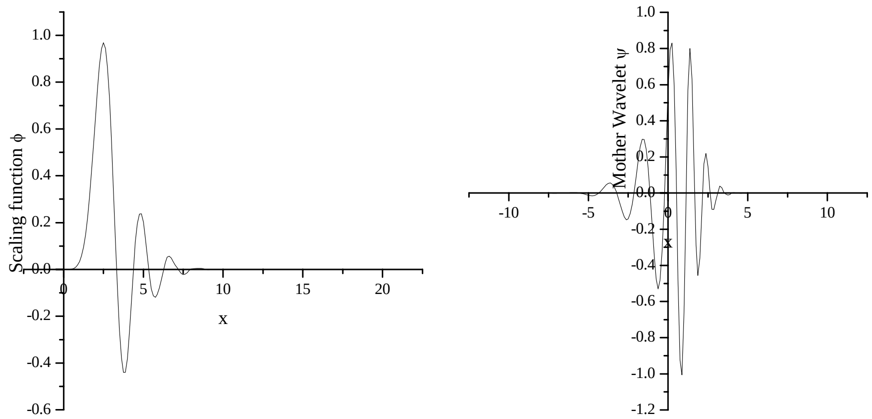
(b) Daubechie 4



(c) Daubechie 6



(d) Daubechie 8



(e) Daubechie 10

Figure 3 Scaling function and mother wavelet for Daubechies

In order to calculate Figure 3, $\phi_{(i)}$ was calculated by solving $\phi_{(i)} = \sum_{k=0}^{2N-1} p_k \phi_{(2i-k)}$. Then $\psi_{(i)}$ was calculated with $\psi_{(i)} = \sum_k q_k \cdot \phi_{(2i-k)}$. Then multi-scaling relation mentioned in section 3.2 (Equation 3.11) was used to get better resolution of the figures.

$$\phi_{\left(\frac{n}{2^j}\right)} = \sum_k p_k^{(j)} \cdot \phi_{(n-\ell)} \tag{3.11}$$

5.4 Biorthogonal wavelet (Cardinal B-spline wavelet)

5.4.1 Definition of cardinal B-spline function

$N_{1(x)}$ is defined as Equation (9.1).

$$N_{1(x)} = \begin{cases} 1 & 0 \le x < 1 \\ 0 & x < 0 \, or \, 1 \le x \end{cases} \tag{9.1}$$

$N_{1(x)}$ is the same as Haar's scaling function. With Equation (9.1), the cardinal B-spline function of m-th order ((m-1)-th rank) is defined as Equation (9.2).

$$\begin{aligned} N_{m(x)} &= \left(N_{m-1} * N_1\right)_{(x)} \\ &= \int_{-\infty}^{\infty} N_{m-1(x-t)} \cdot N_{1(t)} dt \\ &= \int_0^1 N_{m-1(x-t)} dt \end{aligned} \tag{9.2}$$

Finally, $N_{m(x)}$ can be shown as Equation (9.3).

$$N_{m(x)} = \left(\underbrace{N_1 * N_1 * \cdots * N_1}_{m \ elements} \right)_{(x)} \tag{9.3}$$

Another definition of $N_{m(x)}$ can be also made with the truncated power function x_+. The truncated power function is defined as Equation (9.4).

$$\begin{aligned} x_+ &= \max(0, x) \\ x_+^m &= (x_+)^m \end{aligned} \tag{9.4}$$

With (9.4), the cardinal B-spline function is defined as Equation (9.7). This $N_{m(x)}$ is applied for the scaling function

$$N_{m(x)} = \frac{1}{(m-1)!} \sum_{k=0}^{m} (-1)^k \binom{m}{k} (x-k)_+^{m-1} \tag{9.7}$$

where, $\binom{m}{k} = {}_mC_k = \dfrac{m!}{k!(m-k)!}$

The support of $N_{m(x)}$ becomes as follows from Equation (9.7).

$$\operatorname{supp} N_{m(x)} = [0 \quad m] \tag{9.11}$$

5.4.2 Sequence p_k

As mentioned earlier, the sequence p_k can be defined as Laurent polynomial as follow.

$$P_{(z)} = \frac{1}{2} \sum_{k \in Z} p_k \cdot z^k \tag{7.1}$$

The Fourier transform of $\phi_{(x)}$ is derived as Equation (7.2).

$$
\begin{aligned}
\hat{\phi}_{(\omega)} &= \int_{-\infty}^{\infty} e^{-i\omega x} \cdot \phi_{(x)} \cdot dx \\
&= \int_{-\infty}^{\infty} e^{-i\omega x} \cdot \sum_k p_k \cdot \phi_{(2x-k)} \cdot dx \qquad (Equation\,(3.1)) \\
&= \int_{-\infty}^{\infty} e^{-i\omega\frac{y+k}{2}} \cdot \sum_k p_k \cdot \phi_{(y)} \cdot \frac{dy}{2} \qquad (2x-k \to y,\ dx = \frac{dy}{2}) \qquad (7.2) \\
&= \frac{1}{2}\sum_k p_k \cdot e^{-\frac{i\omega}{2}k} \int_{-\infty}^{\infty} e^{-i\frac{\omega}{2}y} \cdot \phi_{(y)} \cdot dy \\
&= P_{\left(e^{-\frac{i\omega}{2}}\right)} \cdot \hat{\phi}_{\left(\frac{\omega}{2}\right)}
\end{aligned}
$$

The cardinal B-spline $N_{m(x)}$ also holds two-scale relation as Equation (9.24).

$$
N_{m(x)} = \sum_k p_k \cdot N_{m(2x-k)} \qquad (9.24)
$$

Fourier transform of $N_{1(x)}$ (m=1, Haar's scaling function) is derived as Equation (9.22).

$$
\begin{aligned}
\hat{N}_{1(\omega)} &= \int_{-\infty}^{\infty} e^{-i\omega x} \cdot N_{1(x)} \cdot dx \\
&= \int_0^1 e^{-i\omega x} \cdot dx \\
&= \left[\frac{e^{-i\omega x}}{-i\omega}\right]_0^1 \qquad (9.22) \\
&= \frac{1-e^{-i\omega}}{i\omega}
\end{aligned}
$$

Fourier transform of convolution Equation (6.12) can be calculated as Equation (6.13).

$$
(f * g)_{(x)} = \int_{-\infty}^{\infty} f_{(x-y)} \cdot g_{(y)} \cdot dy \qquad (6.12)
$$

$$
\begin{aligned}
\left(\widehat{f * g}\right)_{(\omega)} &= \int_{-\infty}^{\infty} e^{-i\omega x} \cdot (f * g)_{(x)} \cdot dx \\
&= \int_{-\infty}^{\infty} e^{-i\omega x} \cdot \int_{-\infty}^{\infty} f_{(x-y)} \cdot g_{(y)} \cdot dy \cdot dx \\
&= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} e^{-i\omega(x-y)} \cdot f_{(x-y)} \cdot e^{-i\omega y} \cdot g_{(y)} \cdot dy \cdot dx \qquad (6.13) \\
&= \int_{-\infty}^{\infty} e^{-i\omega(y)} \cdot f_{(y)} dy \cdot \int_{-\infty}^{\infty} e^{-i\omega y} \cdot g_{(y)} \cdot dy \qquad (x-y \to y) \\
&= \hat{f}_{(\omega)} \cdot \hat{g}_{(\omega)}
\end{aligned}
$$

Therefore, Fourier transform of $N_{m(x)}$ is derived as Equation (9.23) from (9.3), (9.22), and (6.13).

$$
\begin{aligned}
\hat{N}_{m(\omega)} &= Fourier\ of \left(\underbrace{N_1 * N_1 * \cdots * N_1}_{m} \right)_{(x)} \\
&= \underbrace{\hat{N}_{1(\omega)} \cdot \hat{N}_{1(\omega)} \cdot \cdots \cdot \hat{N}_{1(\omega)}}_{m} \\
&= \left(\frac{1-e^{-i\omega}}{i\omega} \right)^m
\end{aligned}
\tag{9.23}
$$

On the other hand, from Equation (7.2), following Equation can be derived.

$$
\hat{N}_{m(\omega)} = P_{\left(e^{-\frac{i\omega}{2}}\right)} \cdot \hat{N}_{m\left(\frac{\omega}{2}\right)}
$$

With Equation (9.23),

$$
\left(\frac{1-e^{-i\omega}}{i\omega} \right)^m = P_{\left(e^{-\frac{i\omega}{2}}\right)} \cdot \left(\frac{1-e^{-i\frac{\omega}{2}}}{i\frac{\omega}{2}} \right)^m
$$

$$
\begin{aligned}
\Leftrightarrow P_{\left(e^{-\frac{i\omega}{2}}\right)} &= \left(\frac{1-e^{-i\omega}}{i\omega} \right)^m \cdot \left(\frac{i\frac{\omega}{2}}{1-e^{-i\frac{\omega}{2}}} \right)^m \\
&= \left(\frac{1}{2} \frac{1-e^{-i\omega}}{1-e^{-i\frac{\omega}{2}}} \right)^m \\
&= \left(\frac{1}{2} \frac{\left(1-e^{-i\frac{\omega}{2}}\right)\left(1+e^{-i\frac{\omega}{2}}\right)}{1-e^{-i\frac{\omega}{2}}} \right)^m \\
&= \frac{1}{2^m} \left(1+e^{-i\frac{\omega}{2}} \right)^m \\
&= \frac{1}{2^m} \sum_{k=0}^{m} \binom{m}{k} \cdot e^{-i\frac{\omega}{2}k} \\
&= \frac{1}{2} \left\{ \frac{1}{2^{m-1}} \sum_{k=0}^{m} \binom{m}{k} \cdot \left(e^{-i\frac{\omega}{2}} \right)^k \right\} \\
&= \frac{1}{2} \sum_{k=0}^{m} p_k \cdot \left(e^{-i\frac{\omega}{2}} \right)^k
\end{aligned}
$$

Therefore,

$$
p_k = \frac{1}{2^{m-1}} \sum_{k=0}^{m} \binom{m}{k} \quad k = 0,1,\cdots,m
$$

5.4.3 Sequence q_k

The auto-correlation of $N_{m(x)}$ is calculated as follows with Equation (6.18).

$$
\begin{aligned}
F_{N_m(x)} &= \int_{-\infty}^{\infty} N_{m(x+y)} \cdot \overline{N}_{m(y)} dy \\
&= \frac{1}{2\pi}\int_{-\infty}^{\infty} e^{i\omega x} \cdot \hat{F}_{(\omega)} \cdot d\omega \\
&= \frac{1}{2\pi}\int_{-\infty}^{\infty} e^{i\omega x} \cdot \left|\hat{N}_{m(\omega)}\right|^2 \cdot d\omega && (\because Equation(6.19)) \\
&= \frac{1}{2\pi}\int_{-\infty}^{\infty} e^{i\omega x} \cdot \left|\frac{1-e^{-i\omega}}{i\omega}\right|^{2m} \cdot d\omega && (\because Equation(9.23)) \\
&= \frac{1}{2\pi}\int_{-\infty}^{\infty} e^{i\omega x} \cdot \left\{\left(\frac{1-e^{-i\omega}}{i\omega}\right)\left(\frac{1-e^{i\omega}}{-i\omega}\right)\right\}^m \cdot d\omega && (\because \left|\frac{1-e^{-i\omega}}{i\omega}\right|^2 = \left(\frac{1-e^{-i\omega}}{i\omega}\right)\left(\frac{1-e^{i\omega}}{-i\omega}\right)) \\
&= \frac{1}{2\pi}\int_{-\infty}^{\infty} e^{i\omega x} \cdot \left\{\left(\frac{1-e^{-i\omega}}{i\omega}\right)\left(\frac{1-e^{-i\omega}}{i\omega}\cdot e^{i\omega}\right)\right\}^m \cdot d\omega \\
&= \frac{1}{2\pi}\int_{-\infty}^{\infty} e^{i\omega(x+m)} \cdot \left(\frac{1-e^{-i\omega}}{i\omega}\right)^{2m} \cdot d\omega \\
&= \frac{1}{2\pi}\int_{-\infty}^{\infty} e^{i\omega(x+m)} \cdot \hat{N}_{2m(\omega)} \cdot d\omega \\
&= N_{2m(x+m)} && (\because f_{(x)} = \frac{1}{2\pi}\int_{-\infty}^{\infty} e^{i\omega x} \cdot \hat{f}_{(\omega)} \cdot d\omega)
\end{aligned}
$$

Since $\operatorname{supp} N_{2m(x)} = [0 \quad 2m]$, $\operatorname{supp} F_{N_m(x)} = [-m \quad m]$.

Therefore, Euler-Forbenius sequence becomes as follows.

$$
\begin{aligned}
E_{N_m(z)} &= \sum_{k=-m+1}^{m-1} F_{N_m(k)} \cdot z^k && (\because Equation(7.8)) \\
&= \sum_{k=-m+1}^{m-1} N_{2m(m+k)} \cdot z^k
\end{aligned}
\qquad (9.27)
$$

From Equation (7.11) and (7.12),

$$
\begin{aligned}
Q_{(z)} &= \frac{1}{2}\sum_{k\in Z} q_k \cdot z^k \\
&= -z^{2m-1} \cdot E_{N_m(-z)} \cdot \overline{P}_{(-z)} \\
&= (-z)^{2m-1} \cdot E_{N_m(-z)} \cdot \frac{1}{2}\frac{1}{2^{m-1}}\sum_{\ell=0}^{m}\binom{m}{\ell}\cdot(-z)^{-\ell} && (\because \overline{P}_{(z)} = \frac{1}{2}\frac{1}{2^{m-1}}\sum_{k=0}^{m}\binom{m}{k}\cdot z^{-k}) \\
&= (-z)^{2m-1} \cdot \sum_{k=-m+1}^{m-1} N_{2m(m+k)} \cdot (-z)^k \cdot \frac{1}{2^m}\sum_{\ell=0}^{m}\binom{m}{\ell}\cdot(-z)^{-\ell} && (\because E_{N_m(-z)} = \sum_{k=-m+1}^{m-1} N_{2m(m+k)} \cdot (-z)^k) \\
&= \sum_{k=-m+1}^{m-1} N_{2m(m+k)} \cdot (-z)^{k+m-1} \cdot \frac{1}{2^m}\sum_{\ell=0}^{m}\binom{m}{\ell}\cdot(-z)^{m-\ell} \\
&= \sum_{k=0}^{2m-2} N_{2m(k+1)} \cdot (-z)^k \cdot \frac{1}{2^m}\sum_{\ell=0}^{m}\binom{m}{m-\ell}\cdot(-z)^{\ell} && (k+m-1 \to k, m-\ell \to \ell) \\
&= \frac{1}{2}\sum_{k=0}^{2m-2}\sum_{\ell=0}^{m}\frac{1}{2^{m-1}} N_{2m(k+1)} \cdot \binom{m}{\ell}\cdot(-z)^{k+\ell} && (\because \binom{m}{m-\ell} = \binom{m}{\ell}) \\
&= \frac{1}{2}\sum_{k=0}^{3m-2}\frac{1}{2^{m-1}}\sum_{\ell=0}^{m} N_{2m(k-\ell+1)} \cdot \binom{m}{\ell}\cdot(-z)^{k} && (k+\ell \to k)
\end{aligned}
$$

Therefore,

$$
q_k = \frac{1}{2^{m-1}}\sum_{\ell=0}^{m} N_{2m(k-\ell+1)} \cdot \binom{m}{\ell} \qquad k = 0,1,\cdots,3m-2 \qquad (9.32)
$$

Then the mother wavelet $\psi_{N_m(x)}$ can be calculated as follows.

$$\begin{aligned}\psi_{N_m(x)} &= \sum_k q_k \cdot N_{m(2x-k)} \\ &= \sum_{k=0}^{3m-2} q_k \cdot N_{m(2x-k)}\end{aligned} \tag{9.33}$$

Since $\mathrm{supp} N_m = [0 \quad m]$,

$$\mathrm{supp}\,\psi_{N_m(x)} = [0 \quad 2m-1]$$

5.4.4 Eular-Frobenius Polynomial

The Eular-Frobenius polynomial is shown as Equation (9.27) as described earlier.

$$E_{N_m(z)} = \sum_{k=-m+1}^{m-1} N_{2m(m+k)} \cdot z^k \tag{9.27}$$

On the other hand, the Eular-Frobenius polynomial of n-th rank is defined as follows.

$$E_{n(z)} = n! \cdot \sum_{k \in Z} N_{n+1\left(\frac{n+1}{2}+k\right)} \cdot z^{k+\lfloor n/2 \rfloor} \tag{9.28}$$

where, $\lfloor x \rfloor$ means the maximum integer that does not exceed x.

If n is odd number, n can be replaced by $2m-1$ as follows.

$$\begin{aligned}E_{2m-1(z)} &= (2m-1)! \cdot \sum_{k \in Z} N_{2m(m+k)} \cdot z^{k+m-1} \\ &= (2m-1)! \cdot \sum_{k \in Z} N_{2m(k+1)} \cdot z^k \\ &= (2m-1)! \cdot \sum_{k=0}^{2m-2} N_{2m(k+1)} \cdot z^k \qquad (\because \mathrm{supp} N_{2m} = [0 \quad 2m])\end{aligned} \tag{9.29}$$

The following equation can be derived from Equation (9.27) with Equation (9.29).

$$\begin{aligned}E_{N_m(z)} &= \sum_{k=-m+1}^{m-1} N_{2m(m+k)} \cdot z^k \\ &= \sum_{k=0}^{2m-2} N_{2m(k+1)} \cdot z^k \cdot \frac{1}{z^{m-1}} \\ &= E_{2m-1(z)} \cdot \frac{1}{(2m-1)! \cdot z^{m-1}}\end{aligned} \tag{9.30}$$

5.4.5 Sequences g_k, and h_k

Laurent polynomial

From Equation (7.11) and (7.12), g_k, and h_k are calculated as follows.

$$G_{(z)} = \frac{1}{2} \sum_{k \in Z} g_k \cdot z^k = \frac{E_{N_m(z)}}{E_{N_m\left(z^2\right)}} \overline{P_{(z)}}$$

$$H_{(z)} = \frac{1}{2} \sum_{k \in Z} h_k \cdot z^k = -z^{-2m+1} \cdot \frac{P_{(-z)}}{E_{N_m\left(z^2\right)}}$$

$$\Leftrightarrow G_{(z)} = \frac{1}{2}\sum_{k\in Z} g_k \cdot z^k$$

$$= \frac{E_{N_m(z)}}{E_{N_m\left(z^2\right)}}\overline{P}_{(z)}$$

$$= \frac{\sum_{k=-m+1}^{m-1} N_{2m(m+k)} \cdot z^k}{E_{N_m\left(z^2\right)}}\overline{P}_{(z)} \qquad (\because E_{N_m(z)} = \sum_{k=-m+1}^{m-1} N_{2m(m+k)} \cdot z^k)$$

$$= \frac{\sum_{k=-m+1}^{m-1} N_{2m(m+k)} \cdot z^k}{E_{N_m\left(z^2\right)}}\frac{1}{2}\sum_{\ell\in Z}\overline{p}_\ell \cdot z^{-\ell} \qquad (\because \overline{P}_{(z)} = \frac{1}{2}\sum_{k\in Z}\overline{p}_k \cdot z^{-k})$$

$$= \frac{\sum_{k=-m+1}^{m-1} N_{2m(m+k)} \cdot z^k}{E_{N_m\left(z^2\right)}}\frac{1}{2^m}\sum_{\ell=0}^{m}\binom{m}{\ell} \cdot z^{-\ell} \qquad (\because p_k = \frac{1}{2^{m-1}}\sum_{k=0}^{m}\binom{m}{k})$$

$$= \frac{1}{2^m}\frac{\sum_{k=-m+1}^{m-1} N_{2m(m+k)} \cdot z^k \cdot \sum_{\ell=0}^{m}\binom{m}{\ell} \cdot z^{-\ell}}{E_{N_m\left(z^2\right)}}$$

$$H_{(z)} = \frac{1}{2}\sum_{k\in Z} h_k \cdot z^k$$

$$= -z^{-2m+1} \cdot \frac{P_{(-z)}}{E_{N_m\left(z^2\right)}}$$

$$= -z^{-2m+1} \cdot \frac{1}{E_{N_m\left(z^2\right)}}\frac{1}{2}\sum_{k\in Z} p_k \cdot (-z)^k \qquad (\because P_{(-z)} = \frac{1}{2}\sum_{k\in Z} p_k \cdot (-z)^k)$$

$$= -z^{-2m+1} \cdot \frac{1}{E_{N_m\left(z^2\right)}}\frac{1}{2^m}\sum_{k=0}^{m}\binom{m}{k} \cdot (-z)^k \qquad (\because p_k = \frac{1}{2^{m-1}}\sum_{k=0}^{m}\binom{m}{k})$$

From Equation (9.30),

$$\frac{1}{E_{N_m(z)}} = \frac{(2m-1)! \cdot z^{m-1}}{E_{2m-1(z)}}$$

$1/E_{N_m(z)}$ is, however, infinite sequence if m is greater than 2. Therefore, the following approximation is applied for $1/E_{N_m(z)}$.

$$\frac{1}{E_{N_m(z)}} = \sum_{k=-\infty}^{\infty}\alpha_k \cdot z^k \to \sum_{k=-n}^{n}\alpha_k \cdot z^k \qquad (9.42)$$

The number of sequence considered *n* should be determined so that the difference between the reconstituted and original signal becomes acceptable.

From Equation (9.29), $E_{2m-1(z)}$ is the (2m-2)-th order polynomial.

$$E_{2m-1(z)} = (2m-1)! \cdot \sum_{k=0}^{2m-2} N_{2m(k+1)} \cdot z^k \qquad (9.29)$$

Symmetric property

From Equation (9.1), the following relation can be obtained, since $N_{1(x)}$ is symmetric to the *x* of 1/2.

$$N_{1\left(\frac{1}{2}+x\right)} = N_{1\left(\frac{1}{2}-x\right)}$$

If the following relation is true,

$$N_{m-1\left(\frac{m-1}{2}+x\right)} = N_{m-1\left(\frac{m-1}{2}-x\right)}$$

The following relation is also true with Equation (9.2).

$$\begin{aligned}
N_{m\left(\frac{m}{2}+x\right)} &= \int_0^1 N_{m-1\left(\frac{m}{2}+x-t\right)} dt && (\because N_{m(x)} = \int_0^1 N_{m-1(x-t)} dt) \\
&= \int_0^1 N_{m-1\left(\frac{m-1}{2}+\frac{1}{2}+x-t\right)} dt \\
&= \int_0^1 N_{m-1\left(\frac{m-1}{2}-\frac{1}{2}-x+t\right)} dt && (\because N_{m-1\left(\frac{m-1}{2}+x\right)} = N_{m-1\left(\frac{m-1}{2}-x\right)}) \\
&= \int_0^1 N_{m-1\left(\frac{m}{2}-1-x+t\right)} dt \\
&= \int_0^1 N_{m-1\left(\frac{m}{2}-x-u\right)} du && (1-t \rightarrow u, dt = -du, 0 to 1 \rightarrow 1 to 0) \\
&= N_{m\left(\frac{m}{2}-x\right)}
\end{aligned}$$

Therefore,

$$N_{2m-1(m-1+x)} = N_{2m-1(m-1-x)}$$

Therefore, $N_{2m-1(x)}$ is symmetric to the *x* of (*m*-1). Then the factors for $z^{\pm k}$ of $\frac{N_{2m-1(x)}}{z^{m-1}}$ are the same. Due to the symmetric property of the B-spline function, if a_i is a solution of $E_{2m-1(z)} = 0$, a_i^{-1} is also a solution.

$$N_{2m-1(z)} = \prod_{i=1}^{m-1} (z - a_i)(z - a_i^{-1}) \qquad (9.43)$$

Approximation of $1/E_{N_{m(z)}}$

$\frac{z^{m-1}}{N_{2m-1(z)}}$ can be shown in the following fashion from Equation (9.43).

$$\begin{aligned}
\frac{z^{m-1}}{N_{2m-1(z)}} &= z^{m-1} \prod_{i=1}^{m-1} \frac{1}{(z-a_i)(z-a_i^{-1})} \\
&= \sum_{i=1}^{m-1} C_i \left[\frac{1}{1-a_i \cdot z} + \frac{a_i \cdot z^{-1}}{1-a_i \cdot z^{-1}} \right]
\end{aligned}$$

where

$$C_i = \prod_{k=1}^{m-1} \frac{1}{\left(a_k - a_i^{-1}\right)} \prod_{i \neq j} \frac{a_j}{a_i - a_j} \quad m \geq 3 \tag{9.45}$$

$$C_1 = \frac{1}{\left(a_1 - a_i^{-1}\right)} \qquad m = 2 \tag{9.46}$$

This relation can be proved as follows.

i) m=2

$$\frac{z}{N_{3(z)}} = \frac{z}{(z - a_1)\left(z - a_1^{-1}\right)}$$

$$= \frac{-a_1}{\left(1 - a_1 \cdot z^{-1}\right)(1 - a_1 \cdot z)}$$

$$= \left\{ \frac{1}{1 - a_1 \cdot z} + \frac{a_1 \cdot z^{-1}}{1 - a_1 \cdot z^{-1}} \right\} \frac{-a_1}{1 - a_1^2}$$

$$= \frac{1}{a_1 - a_1^{-1}} \left\{ \frac{1}{1 - a_1 \cdot z} + \frac{a_1 \cdot z^{-1}}{1 - a_1 \cdot z^{-1}} \right\}$$

$$= C_1 \left\{ \frac{1}{1 - a_1 \cdot z} + \frac{a_1 \cdot z^{-1}}{1 - a_1 \cdot z^{-1}} \right\}$$

ii) m>2

The following equation can be obtained with partial fraction.

$$\frac{z}{(z - a_i)\left(z - a_i^{-1}\right)} = \frac{1}{a_i - a_i^{-1}} \left\{ \frac{1}{1 - a_i \cdot z} + \frac{a_i \cdot z^{-1}}{1 - a_i \cdot z^{-1}} \right\}$$

The following two are defined.

$$A_i = \frac{1}{1 - a_i \cdot z}$$

$$B_i = \frac{a_i \cdot z^{-1}}{1 - a_i \cdot z^{-1}}$$

Then

$$A_i \cdot A_j = \frac{1}{1 - a_i \cdot z} \cdot \frac{1}{1 - a_j \cdot z}$$

$$= \left(\frac{1}{1 - a_i \cdot z} \cdot a_i - \frac{1}{1 - a_j \cdot z} \cdot a_j \right) \cdot \frac{1}{a_i - a_j}$$

$$= \frac{a_i}{a_i - a_j} A_i + \frac{a_j}{a_j - a_i} A_j$$

$$A_i \cdot B_j = \frac{1}{1 - a_i \cdot z} \cdot \frac{a_j \cdot z^{-1}}{1 - a_j \cdot z^{-1}}$$

$$= \left(\frac{1}{1 - a_i \cdot z} \cdot a_i \cdot a_j + \frac{a_j \cdot z^{-1}}{1 - a_j \cdot z^{-1}} \right) \cdot \frac{1}{1 - a_i \cdot a_j}$$

$$= \frac{a_i \cdot a_j}{1 - a_i \cdot a_j} A_i + \frac{1}{1 - a_i \cdot a_j} B_j$$

$$
\begin{aligned}
B_i \cdot B_j &= \frac{a_i \cdot z^{-1}}{1-a_i \cdot z^{-1}} \cdot \frac{a_j \cdot z^{-1}}{1-a_j \cdot z^{-1}} \\
&= \left(\frac{a_i \cdot z^{-1}}{1-a_i \cdot z^{-1}} \cdot a_j - \frac{a_j \cdot z^{-1}}{1-a_j \cdot z^{-1}} \cdot a_i \right) \cdot \frac{1}{a_i - a_j} \\
&= \frac{a_j}{a_i - a_j} B_i + \frac{a_i}{a_j - a_i} B_j
\end{aligned}
$$

Therefore,

$$
\begin{aligned}
(A_i + B_i) \cdot (A_j + B_j) &= A_i \cdot (A_j + B_j) + B_i \cdot (A_j + B_j) \\
&= \left(\frac{a_i}{a_i - a_j} + \frac{a_i \cdot a_j}{1 - a_i \cdot a_j} \right) \cdot A_i + \left(\frac{1}{1 - a_i \cdot a_j} + \frac{a_j}{a_i - a_j} \right) \cdot B_i \qquad \textit{Only } A_i \textit{ and } B_i \textit{ part} \\
&= \frac{a_i (1 - a_j^2)}{(a_i - a_j)(1 - a_i \cdot a_j)} \cdot A_i + \frac{a_i (1 - a_j^2)}{(a_i - a_j)(1 - a_i \cdot a_j)} \cdot B_i \qquad \textit{Only } A_i \textit{ and } B_i \textit{ part} \\
&= \frac{a_j (a_j^{-1} - a_j)}{(a_i - a_j)(a_i^{-1} - a_j)} (A_i + B_i) \qquad \textit{Only } A_i \textit{ and } B_i \textit{ part} \\
&= \frac{1}{(a_j - a_i^{-1})} \frac{a_j}{(a_i - a_j)} \cdot (a_j - a_j^{-1}) \cdot (A_i + B_i) \qquad \textit{Only } A_i \textit{ and } B_i \textit{ part}
\end{aligned}
$$

On the other hand,

$$
\begin{aligned}
\frac{z^{k-1}}{N_{2k-1(z)}} &= z^{k-1} \prod_{i=1}^{k-1} \frac{1}{(z - a_i)(z - a_i^{-1})} \\
&= \prod_{i=1}^{k-2} \frac{z}{(z - a_i)(z - a_i^{-1})} \\
&= \prod_{i=1}^{k-2} \frac{1}{a_i - a_i^{-1}} \left\{ \frac{1}{1 - a_i \cdot z} + \frac{a_i \cdot z^{-1}}{1 - a_i \cdot z^{-1}} \right\} \\
&= \left\{ \prod_{i=1}^{k-2} \frac{1}{a_i - a_i^{-1}} \right\} \left\{ \prod_{i=1}^{k-2} \left\{ \frac{1}{1 - a_i \cdot z} + \frac{a_i \cdot z^{-1}}{1 - a_i \cdot z^{-1}} \right\} \right\} \\
&= \left\{ \prod_{i=1}^{k-2} \frac{1}{a_i - a_i^{-1}} \right\} \left\{ \prod_{i=1}^{k-2} \{ A_i + B_i \} \right\}
\end{aligned}
$$

Considering just the factor of $(A_i + B_i)$,

$$\left\{\prod_{\ell=1}^{k-2}\frac{1}{a_\ell - a_\ell^{-1}}\right\}(A_i + B_i)\left\{\prod_{\ell=1,\,\ell\neq i}^{k-2}\{A_\ell + B_\ell\}\right\}$$

$$= \left\{\prod_{\ell=1}^{k-2}\frac{1}{a_\ell - a_\ell^{-1}}\right\}(A_i + B_i)(A_1 + B_1)\left\{\prod_{\ell=2,\,\ell\neq i}^{k-2}\{A_\ell + B_\ell\}\right\}$$

$$= \left\{\prod_{\ell=1}^{k-2}\frac{1}{a_\ell - a_\ell^{-1}}\right\}\left\{\frac{1}{(a_1 - a_i^{-1})}\frac{a_1}{(a_i - a_1)}\cdot\left(a_1 - a_1^{-1}\right)\right\}\cdot(A_i + B_i)\left\{\prod_{\ell=2,\,\ell\neq i}^{k-2}\{A_\ell + B_\ell\}\right\}$$

$$= \left\{\prod_{\ell=1}^{k-2}\frac{1}{a_\ell - a_\ell^{-1}}\right\}\left\{\frac{1}{(a_1 - a_i^{-1})}\frac{a_1}{(a_i - a_1)}\cdot\left(a_1 - a_1^{-1}\right)\right\}\left\{\frac{1}{(a_2 - a_i^{-1})}\frac{a_1}{(a_i - a_2)}\cdot\left(a_2 - a_2^{-1}\right)\right\}\cdot(A_i + B_i)\left\{\prod_{\ell=3,\,\ell\neq i}^{k-2}\{A_\ell + B_\ell\}\right\}$$

$$= \left\{\prod_{\ell=1}^{k-2}\frac{1}{a_\ell - a_\ell^{-1}}\right\}\left\{\prod_{\ell=1,\,\ell\neq i}^{k-2}\frac{1}{(a_\ell - a_i^{-1})}\frac{a_\ell}{(a_i - a_\ell)}\cdot\left(a_\ell - a_\ell^{-1}\right)\right\}\cdot(A_i + B_i)$$

$$= \frac{1}{a_i - a_i^{-1}}\left\{\prod_{\ell=1,\,\ell\neq i}^{k-2}\frac{1}{(a_\ell - a_i^{-1})}\frac{a_\ell}{(a_i - a_\ell)}\right\}\cdot(A_i + B_i)$$

$$= \prod_{\ell=1}^{k-2}\frac{1}{(a_\ell - a_i^{-1})}\prod_{\ell=1,\,\ell\neq i}^{k-2}\frac{a_\ell}{(a_i - a_\ell)}\cdot(A_i + B_i)$$

$$= C_i\cdot(A_i + B_i)$$

Therefore,

$$\frac{z^{k-1}}{N_{2k-1(z)}} = \left\{\prod_{i=1}^{k-2}\frac{1}{a_i - a_i^{-1}}\right\}\left\{\prod_{i=1}^{k-2}\{A_i + B_i\}\right\} = \sum_{i=1}^{m-1}C_i\cdot\left[\frac{1}{1 - a_i\cdot z} + \frac{a_i\cdot z^{-1}}{1 - a_i\cdot z^{-1}}\right]$$

With this equation, the approximation of $\frac{z^{m-1}}{N_{2m-1(z)}}$ is defined as follows.

$$\frac{z^{m-1}}{N_{2m-1(z)}} = \sum_{k=-n}^{n}\left(\sum_{i=1}^{m-1}C_i\cdot a_i^{|k|}\right)\cdot z^k \qquad (9.44)$$

Therefore,

$$\frac{1}{N_{N_m(z)}} = (2m-1)!\frac{z^{m-1}}{N_{2m-1(z)}} = (2m-1)!\sum_{k=-n}^{n}\left(\sum_{i=1}^{m-1}C_i\cdot a_i^{|k|}\right)\cdot z^k$$

Then,

$$\alpha_k = (2m-1)!\sum_{i=1}^{m-1}C_i\cdot a_i^{|k|}$$

Sequences g_k, and h_k

g_k, and h_k are obtained as the factor sequence to z of the following equations. The n of greater than 8 generally gives acceptable accuracy.

$$G_{(z)} = \frac{1}{2^m}\left\{\sum_{k=-m+1}^{m-1} N_{2m(m+k)} \cdot z^k\right\} \cdot \left\{\sum_{\ell=0}^{m}\binom{m}{\ell} \cdot z^{-\ell}\right\} \cdot \left\{(2m-1)! \sum_{k=-n}^{n}\left(\sum_{i=1}^{m-1} C_i \cdot a_i^{|k|}\right) \cdot z^{2k}\right\}$$

$$H_{(z)} = -z^{-2m+1} \cdot \left\{\frac{1}{2^m}\sum_{k=0}^{m}\binom{m}{k} \cdot (-z)^k\right\}\left\{(2m-1)! \sum_{k=-n}^{n}\left(\sum_{i=1}^{m-1} C_i \cdot a_i^{|k|}\right) \cdot z^{2k}\right\}$$

The support for g_k, and h_k are as follows.

$$g_k \qquad \begin{aligned} k &= -m+1-m-2n, \cdots\cdots, m-1+0+2n \\ &= -2m-2n+1, \cdots\cdots, m+2n-1 \end{aligned}$$

$$h_k \qquad \begin{aligned} k &= -2m+1+0-2n, \cdots\cdots, -2m+1+m+2n \\ &= -2m-2n+1, \cdots\cdots, -m+2n+1 \end{aligned}$$

5.4.6 Sequence $c_k^{(0)}$

The basic spline for the m-th order is defined as Equation (9.35).

$$L_{m(j)} = \delta_{j,0} = \begin{cases} 1 & j = 0 \\ 0 & j \neq 0 \end{cases} \tag{9.35}$$

$L_{m(x)}$ is a linear constitution of N_m as Equation (9.36).

$$L_{m(x)} = \sum_{k=-\infty}^{\infty} \beta_k^{(m)} \cdot N_{m\left(x+\frac{m}{2}-k\right)} \tag{9.36}$$

The sequence $\beta_k^{(m)}$ should be chosen to hold Equation (9.35). In order to choose $\beta_k^{(m)}$, following polynomial is defined.

$$B_{m(z)} = \sum_k \beta_k^{(m)} \cdot z^k \tag{9.37}$$

As mentioned earlier, the Eular-Frobenius polynomial of n-th rank is defined as follows.

$$E_{n(z)} = n! \cdot \sum_{k \in Z} N_{n+1\left(\frac{n+1}{2}+k\right)} \cdot z^{k+\lfloor n/2 \rfloor} \tag{9.28}$$

By replacing n by m-1,

$$E_{m-1(z)} = (m-1)! \cdot \sum_{k \in Z} N_{m\left(\frac{m}{2}+k\right)} \cdot z^{k+\lfloor (m-1)/2 \rfloor}$$

$$\Leftrightarrow \frac{E_{m-1(z)}}{z^{\lfloor (m-1)/2 \rfloor}} = (m-1)! \cdot \sum_{k \in Z} N_{m\left(\frac{m}{2}+k\right)} \cdot z^k$$

$$\Leftrightarrow \sum_{k \in Z} N_{m\left(\frac{m}{2}+k\right)} \cdot z^k = \frac{E_{m-1(z)}}{(m-1)! \cdot z^{\lfloor (m-1)/2 \rfloor}}$$

From Equation (9.35) and (9.36),,

$$L_{m(j)} = \sum_{k=-\infty}^{\infty} \beta_k^{(m)} \cdot N_{m\left(j+\frac{m}{2}-k\right)} = \delta_{j,0}$$

By multiplying z^j to both sides and summing up to j,

$$\sum_j L_{m(j)} = \sum_j \sum_k \beta_k^{(m)} \cdot N_{m\left(j+\frac{m}{2}-k\right)} \cdot z^j = \sum_j \delta_{j,0} \cdot z^j = \delta_{0,0} \cdot z^0 = 1$$

The left side can be transformed as follows.

$$\begin{aligned}
\sum_j \sum_k \beta_k^{(m)} \cdot N_{m\left(j+\frac{m}{2}-k\right)} \cdot z^j &= \sum_k \beta_k^{(m)} \cdot z^k \sum_j N_{m\left(j+\frac{m}{2}-k\right)} \cdot z^{j-k} \\
&= \sum_k \beta_k^{(m)} \cdot z^k \sum_j N_{m\left(j+\frac{m}{2}\right)} \cdot z^j \qquad (j-k \to j) \\
&= B_{m(z)} \frac{E_{m-1(z)}}{(m-1)! \cdot z^{\lfloor (m-1)/2 \rfloor}}
\end{aligned}$$

Therefore,

$$B_{m(z)} = \frac{(m-1)! \cdot z^{\lfloor (m-1)/2 \rfloor}}{E_{m-1(z)}} \tag{9.38}$$

Here, only the case when *m* is even number is considered. If *m* is odd number, it becomes more complicated to calculate $\beta_k^{(m)}$.

$$\begin{aligned}
B_{2m(z)} &= \frac{(2m-1)! \cdot z^{\lfloor (2m-1)/2 \rfloor}}{E_{2m-1(z)}} \\
&= (2m-1)! \cdot \frac{z^{(m-1)}}{E_{2m-1(z)}} \\
&= (2m-1)! \cdot \sum_{k=-n}^{n} \left(\sum_{i=1}^{m-1} C_i \cdot a_i^{|k|} \right) \cdot z^k \qquad (\because \frac{z^{m-1}}{N_{2m-1(z)}} = \sum_{k=-n}^{n} \left(\sum_{i=1}^{m-1} C_i \cdot a_i^{|k|} \right) \cdot z^k, Equation\,(9.44)) \\
&= \sum_{k=-n}^{n} \alpha_k \cdot z^k \qquad (\because \alpha_k = (2m-1)! \sum_{i=1}^{m-1} C_i \cdot a_i^{|k|})
\end{aligned}$$

By using basic spline, interpolating function $f_{0(x)}$ can be derived as follows with the original signal $f_{(x)}$.

$$\begin{aligned}
f_{0(x)} &= \sum_k f_{(x)} \cdot L_{m(x-k)} \\
&= \sum_k f_{(x)} \cdot \sum_\ell \beta_\ell^{(m)} \cdot N_{m\left(x-k+\frac{m}{2}-\ell\right)}
\end{aligned}$$

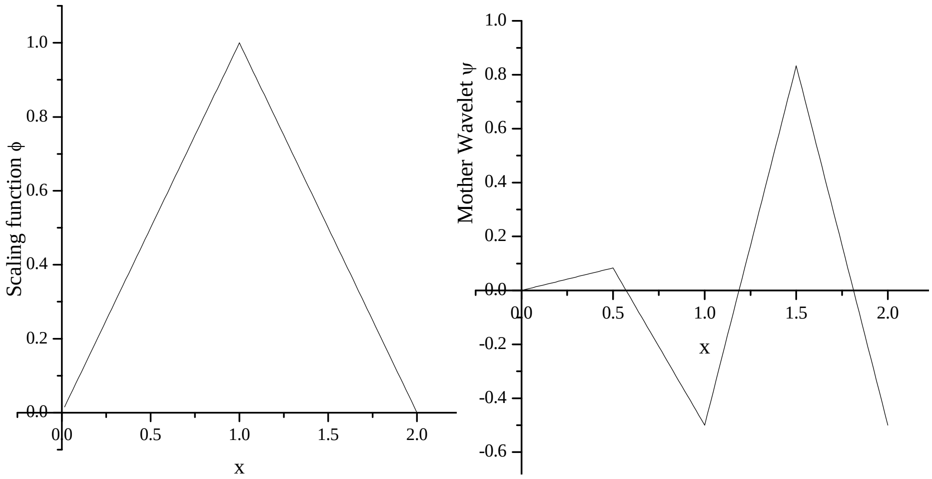
Since only even number is considered for m,

$$
\begin{aligned}
f_{0(x)} &= \sum_k f_{(x)} \cdot \sum_\ell \beta_\ell^{(m)} \cdot N_{m\left(x-k+\frac{m}{2}-\ell\right)} \\
&= \sum_k f_{(x)} \cdot \sum_\ell \beta^{(2m)}_{\ell+\frac{m}{2}-k} \cdot N_{m(x-\ell)} \qquad \left(k-\frac{m}{2}+\ell \to \ell\right) \\
&= \sum_\ell \left(\sum_k f_{(x)} \cdot \beta^{(2m)}_{\ell+\frac{m}{2}-k} \right) \cdot N_{2m(x-\ell)} \\
&= \sum_\ell c_\ell^{(0)} \cdot N_{m(x-\ell)}
\end{aligned}
\tag{9.39}
$$

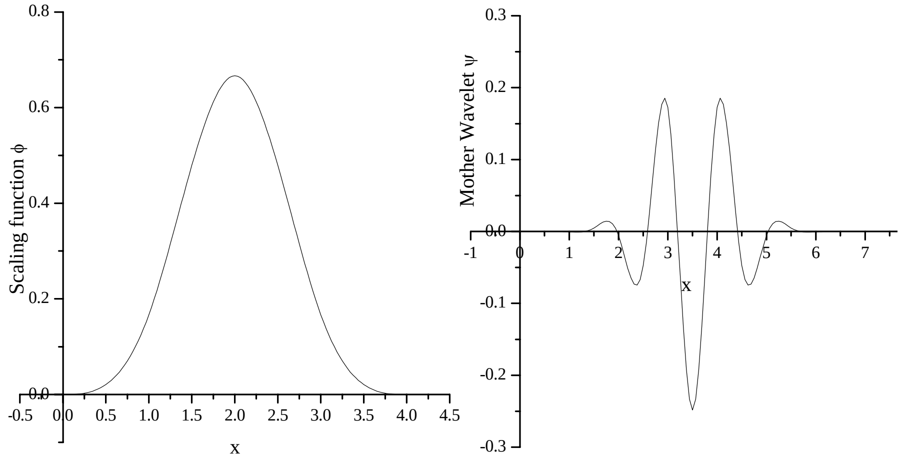
where,

$$
c_\ell^{(0)} = \sum_k f_{(\ell)} \cdot \beta^{(2m)}_{\ell+\frac{m}{2}-k} \tag{9.40}
$$

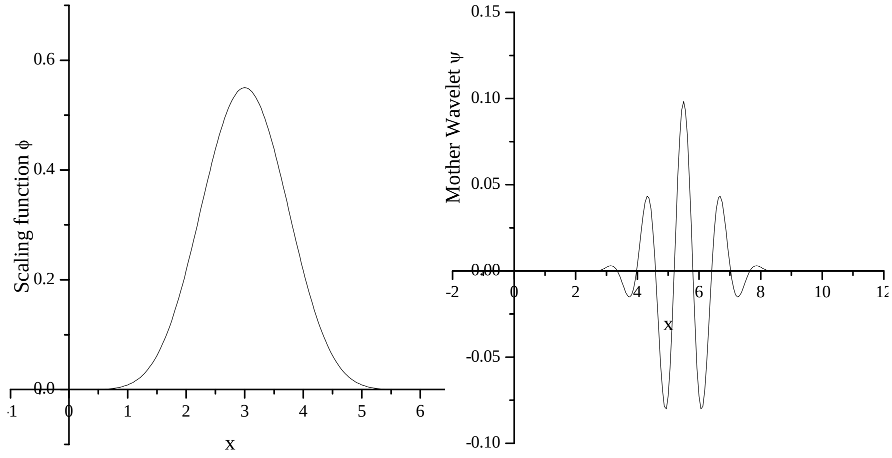
The scaling functions and mother wavelets for cardinal B-spline with m of 2, 4, and 6 are shown in Figure 4.



(a) cardinal B-spline 2



(b) cardinal B-spline 4



(c) cardinal B-spline 6

Figure 4 Scaling function and mother wavelet for cardinal B-spline

6 Coding of the Wavelet transform software

6.1 Required software and how to get source code

The software (classes or subroutines) is coded on the following developing framework.

Microsoft Development Environment 2003	Version 7.1.3019
Microsoft .Net Framework 1.1	Version 1.1.4322 SP1
Microsoft Visual Basic .Net	

The source code or executable file can be requested by sending e-mail to Kusunoki (kusunoki@kenken.go.jp). The softwares listed above are not needed to just run the executable file. Only Microsoft .Net Framework is needed for the executable file. It is distributed by Microsoft for free.

6.2 Summary of equations

For the original signal $f_{(x)}$, the interpolating function $f_{0(x)}$ is calculated first.

[Daubechies]

Inverse Fourier transform of the following equation.

$$\hat{c}_m^{(0)} = \frac{\hat{f}_{0(m)}}{\hat{\phi}_{(n-1)}} = \frac{\hat{f}_{(m)}}{\hat{\phi}_{(n-1)}}$$

[B-spline]

$$c_\ell^{(0)} = \sum_k f_{(\ell)} \cdot \beta^{(2m)}_{\ell+\frac{m}{2}-k}$$

$$B_{m(z)} = \sum_k \beta_k^{(m)} \cdot z^k = \sum_{k=-n}^{n} \alpha_k \cdot z^k$$

$$\alpha_k = (2m-1)! \sum_{i=1}^{m-1} C_i \cdot a_i^{|k|}$$

$$C_1 = \frac{1}{\left(a_1 - a_i^{-1}\right)} \qquad m = 2\,, \quad C_i = \prod_{k=1}^{m-1} \frac{1}{\left(a_k - a_i^{-1}\right)} \prod_{i \neq j} \frac{a_j}{a_i - a_j} \quad m \geq 3$$

$$N_{2m-1(x)} = \prod_{i=1}^{m-1} (z - a_i)\left(z - a_i^{-1}\right)$$

$$N_{m(x)} = \frac{1}{(m-1)!} \sum_{k=0}^{m} (-1)^k \binom{m}{k} (x-k)_+^{m-1}\,, \quad \binom{m}{k} = {}_m C_k = \frac{m!}{k! \cdot (m-k)!}$$

Then the sequences p_k, q_k, g_k, and h_k for the wavelet transform should be defined. [Daubechies]

$$p_k = \sqrt{2} \cdot p_{(k)} \qquad k = 0,1,\cdots,2N-1$$
$$g_k = p_{-k} \qquad k = 1-2N,\cdots,0$$
$$q_k = (-1)^k \cdot p_{1-k} \qquad k = 2-2N,\cdots,0,1$$
$$h_k = (-1)^k \cdot p_{1+k} \qquad k = -1,0,1,\cdots 2N-2$$

Table 1 $p_{(k)}$ for p_k of Daubechies wavelet

N	2	3	4	6	8	10
P(0)	0.482962913144534	0.332670552950080	0.230377813308890	0.111540743350110	0.054415842243110	0.026670057900550
P(1)	0.836516303737807	0.806891509311090	0.714846570552910	0.494623890398450	0.312871590914320	0.188176800077630
P(2)	0.224143868042013	0.459877502118490	0.630880767929860	0.751133908021100	0.675630736297320	0.527201188931580
P(3)	-0.129409522551260	-0.135011020010250	-0.027983769416860	0.315250351709200	0.585354683654220	0.688459039453440
P(4)	-	-0.085441273882030	-0.187034811719090	-0.226264693965440	-0.015829105256380	0.281172343660570
P(5)	-	0.035226291885710	0.030841381835560	-0.129766867567270	-0.284015542961580	-0.249846424327160
P(6)	-	-	0.032883011666890	0.097501605587320	0.000472484573910	-0.195946274377290
P(7)	-	-	-0.010597401785070	0.027522865530310	0.128747426620490	0.127369340335750
P(8)	-	-	-	-0.031582039317490	-0.017369301001810	0.093057364603550
P(9)	-	-	-	0.000553842201160	-0.044088253930800	-0.071394147166350
P(10)	-	-	-	0.004777257510950	0.013981027917400	-0.029457536821840
P(11)	-	-	-	-0.001077301085310	0.008746094047410	0.033212674059360
P(12)	-	-	-	-	-0.004870352993450	0.003606553566990
P(13)	-	-	-	-	-0.000391740373380	-0.010733175483300
P(14)	-	-	-	-	0.000675449406450	0.001395351747070
P(15)	-	-	-	-	-0.000117476784120	0.001992405295190
P(16)	-	-	-	-	-	-0.000685856694960
P(17)	-	-	-	-	-	-0.000116466855130
P(18)	-	-	-	-	-	0.000093588670320
P(19)	-	-	-	-	-	-0.000013264202890

[B-spline]

$$p_k = \frac{1}{2^{m-1}} \sum_{k=0}^{m} \binom{m}{k} \quad k = 0,1,\cdots,m$$

$$q_k = \frac{1}{2^{m-1}} \sum_{\ell=0}^{m} N_{2m(k-\ell+1)} \cdot \binom{m}{\ell} \qquad k = 0,1,\cdots,3m-2$$

$$G_{(z)} = \frac{1}{2^m} \left\{ \sum_{k=-m+1}^{m-1} N_{2m(m+k)} \cdot z^k \right\} \cdot \left\{ \sum_{\ell=0}^{m} \binom{m}{\ell} \cdot z^{-\ell} \right\} \cdot \left\{ (2m-1)! \sum_{k=-n}^{n} \left(\sum_{i=1}^{m-1} C_i \cdot a_i^{|k|} \right) \cdot z^{2k} \right\}$$

$$H_{(z)} = -z^{-2m+1} \cdot \left\{ \frac{1}{2^m} \sum_{k=0}^{m} \binom{m}{k} \cdot (-z)^k \right\} \left\{ (2m-1)! \sum_{k=-n}^{n} \left(\sum_{i=1}^{m-1} C_i \cdot a_i^{|k|} \right) \cdot z^{2k} \right\}$$

$$g_k \qquad k = -2m-2n+1,\cdots\cdots,m+2n-1$$

$$h_k \qquad k = -2m-2n+1,\cdots\cdots,-m+2n+1$$

$$\alpha_k = (2m-1)! \sum_{i=1}^{m-1} C_i \cdot a_i^{|k|}$$

$$C_1 = \frac{1}{(a_1 - a_i^{-1})} \qquad m=2\ , \quad C_i = \prod_{k=1}^{m-1} \frac{1}{(a_k - a_i^{-1})} \prod_{i \neq j} \frac{a_j}{a_i - a_j} \quad m \geq 3$$

$$N_{2m-1(x)} = \prod_{i=1}^{m-1} (z - a_i)(z - a_i^{-1})$$

$$N_{m(x)} = \frac{1}{(m-1)!} \sum_{k=0}^{m} (-1)^k \binom{m}{k} (x-k)_+^{m-1}\ , \quad \binom{m}{k} = {}_mC_k = \frac{m!}{k!\cdot(m-k)!}$$

Mother wavelet $\psi_{(n)}$ and scaling function $\phi_{(n)}$ for integer points can be calculated as follows.

[Daubechies]

$$\begin{Bmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{Bmatrix} = \left(\begin{bmatrix} & [\text{Related to } p_k] & & 0 \\ & & & \vdots \\ & & & 0 \\ 1 & \cdots & \cdots & 1 \end{bmatrix} - \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & \ddots & \ddots & \vdots \\ \vdots & \ddots & 1 & 0 \\ 0 & \cdots & 0 & 0 \end{bmatrix} \right) \cdot \begin{Bmatrix} \phi_1 \\ \vdots \\ \vdots \\ \phi_n \end{Bmatrix}, \quad \phi_0 = 0\ , \quad n = 2N-1$$

$$\phi_{(i)} = \sum_{j=ST}^{ED} p_j \cdot \phi_{(2i-j)}\ , \quad ST = \begin{cases} 0 & i \leq m-1 \\ 2\cdot(i-m)+1 & i > m-1 \end{cases}, \quad ED = \begin{cases} 2i & i \leq m-1 \\ 2m-1 & i > m-1 \end{cases}$$

$$\operatorname{supp}\phi = [0 \quad 2N-1]$$

$$\psi_{(n)} = \sum_{k=2-2N}^{1} q_k \cdot \phi_{(2n-k)} \qquad \operatorname{supp}\psi = [1-N \quad N]$$

[B-spline]

$$\psi_{N_m(n)} = \sum_{k=0}^{3m-2} q_k \cdot N_{m(2n-k)} \qquad \operatorname{supp}\psi_{N_m} = [0 \quad 2m-1]$$

$$\phi_{N_m(n)} = N_{m(n)} = \frac{1}{(m-1)!}\sum_{k=0}^{m}(-1)^k\binom{m}{k}(n-k)_+^{m-1} \quad \binom{m}{k} = {}_mC_k = \frac{m!}{k!(m-k)!}, \quad \mathrm{supp}\, N_m = [0 \quad m]$$

If internally divided points are needed to draw more accurate figures of scaling function and mother wavelet,

$$\psi_{\left(\frac{n}{2^j}\right)} = \sum_k q_k^{(j)} \cdot \phi_{(n-k)}$$

$$q_k^{(j)} = \sum_\ell q_\ell^{(j-1)} \cdot q_{k-2\ell}\ , \quad q_k^{(1)} = q_k$$

$$\phi_{\left(\frac{n}{2^j}\right)} = \sum_k p_k^{(j)} \cdot \phi_{(n-\ell)}$$

$$p_k^{(j)} = \sum_\ell p_{k-2\ell} \cdot p_\ell^{(j)}\ , \quad p_k^{(1)} = p_k$$

The decomposition algorism is shown as follows.

$$c_k^{(j-1)} = \frac{1}{2}\left(g * c^{(j)}\right)_k^{\downarrow}$$

$$d_k^{(j-1)} = \frac{1}{2}\left(h * c^{(j)}\right)_k^{\downarrow}$$

$$c_k^{\downarrow} = c_{2k} \qquad n_c^{\downarrow} = \mathrm{int}\left(\frac{n_c}{2} + 0.5\right)$$

$$(a*b)_k = \sum_\ell a_{k-\ell} \cdot b_\ell \qquad A_{(z)} \cdot B_{(z)} = \sum_{i=i_a+i_b}^{i_a+i_b+n_a+n_b-2} (a*b)_k \cdot z^k$$

The reconstitution algorism is shown as follows.

$$c_k^{(j)} = \left(p * c^{(j-1)\uparrow}\right)_k + \left(q * d^{(j-1)\uparrow}\right)_k$$

$$\begin{matrix} c_{2k}^{\uparrow} = c_k \\ c_{2k+1}^{\uparrow} = 0 \end{matrix} \qquad n_c^{\uparrow} = 2n_c$$

Then $f_{j(x)}$ and $g_{j(x)}$ are calculated with following equations.

$$f_{j(x)} = \sum_k c_k^{(j)} \cdot \phi_{\left(2^j x - k\right)}$$

$$g_{j(x)} \equiv \sum_k d_k^{(j)} \cdot \psi_{\left(2^j x - k\right)}$$

The j-th rank has $n/2^j$ data points, where n is the number of data points of original signal. Sometimes number of data points need to be constant n for each rank. In that case, following equations are used to calculate for the internally divided points.

$$f_{j\left(\frac{n}{2^{j+\ell}}\right)} = \sum_k c_k^{(j+\ell)} \cdot \phi_{(n-k)}$$

$$c_k^{(j+i)} = \sum_\ell c_\ell^{(j+i-1)} \cdot p_{k-2\ell}$$

$$g_{j\left(\frac{n}{2^{j+\ell}}\right)} = \sum_k d_k^{(j+\ell)} \cdot \phi_{(n-k)}$$

$$d_k^{(j+i)} = \sum_\ell d_\ell^{(j+i-1)} \cdot q_{k-2\ell}$$

6.3 Classes for Wavelet

Seven classes are defined for the Wavelet transform as shown in Figure 5.

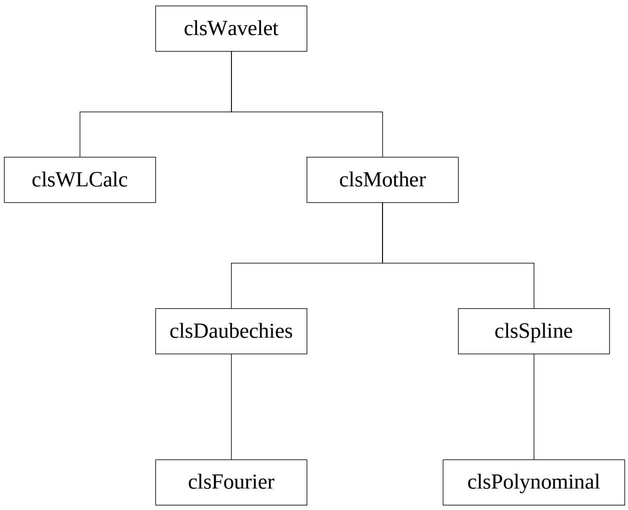


Figure 5 Seven classes

Each classes are;

clsWavelet

This is the main class. This class needs to be declared in the software. Other six classes are internal classes, and they are defined in the main class or some other classes. You don’t have to care about other classes when coding.

clsWLCalc

This class is to decompose and reconstitute the wavelet components.

clsMother

This class is the main class for mother wavelets. clsWavelet call clsMother to deal with mother wavelet and scaling function.

clsDaubechies

This class is for Daubechies wavelet.

clsSpline

This class is for cardinal B-spline wavelet.

clsPolynominal

This class is to calculate polynomial functions.

In the source code, if $a_{(i)}$ is a array of factors of Laurent polynomial such as

$$A_{(z)} = \frac{1}{2}\sum_{k=0}^{j} a_{(k)} \cdot z^{k+m}$$

$a_{(0)}$ is the smallest exponent of *z, m,* then $a_{(1)}$ is the factor for z^m, $a_{(2)}$ for z^{m+1} and so on.

6.4 clsDaubechie

6.4.1 Properties

Parameter	R&W	The order of Daubechies. Only 2, 4, 6, 8, and 10 is available
SuppMaxFai	R	The upper end of support of ϕ. (=2N-1)
SuppMaxPsai	R	The upper end of support of ψ. (=N)
SuppMinFai	R	The lower end of support of ϕ. (=0)
SuppMinPsai	R	The lower end of support of ψ. (=1-N)

Local varies for the properties.

DaubechiePramam=Parameter

6.4.2 GetParam(P(), Q(), G(), H())

This subroutine calculate sequences p_k, q_k, g_k, and h_k. The property of “Parameter” should be set.

$$p_k = \sqrt{2} \cdot p_{(k)} \qquad k = 0,1,\cdots,2N-1$$
$$g_k = p_{-k} \qquad k = 1-2N,\cdots,0$$
$$q_k = (-1)^k \cdot p_{1-k} \qquad k = 2-2N,\cdots,0,1$$
$$h_k = (-1)^k \cdot p_{1+k} \qquad k = -1,0,1,\cdots 2N-2$$

```
Public Sub GetParam(ByRef P() As Double, ByRef Q() As Double, ByRef G() As Double, ByRef
H() As Double)
        '--------------------------------------------------
        '   Scaling function for Daubechie
        '
        '   N=2～10
        '   P(k)                [    0,2N-1], 2N points
        '   Q(k)=(-1)^k*P(1-k)  [2-2N,    1], 2N points
        '   G(k)=P(k)           [1-2N,    0], 2N points
        '   H(k)=Q(k)           [-1,   2N-2], 2N points
        '--------------------------------------------------
```

```
Dim i, k, Pn As Integer
ReDim P(DaubechieParam * 2), Q(DaubechieParam * 2)
ReDim G(DaubechieParam * 2), H(DaubechieParam * 2)
P(0) = 0
Q(0) = 2 - 2 * DaubechieParam
G(0) = -(P(0) + UBound(P) - 1)
H(0) = -(Q(0) + UBound(Q) - 1)

Select Case DaubechieParam
    Case 2
        P(1) = 0.482962913144534
        P(2) = 0.836516303737807
        P(3) = 0.224143868042013
        P(4) = -0.12940952255126
    Case 3
        P(1) = 0.33267055295008
        P(2) = 0.80689150931109
        P(3) = 0.45987750211849
        P(4) = -0.13501102001025
        P(5) = -0.08544127388203
        P(6) = 0.03522629188571
    Case 4
        P(1) = 0.23037781330889
        P(2) = 0.71484657055291
        P(3) = 0.63088076792986
        P(4) = -0.02798376941686
        P(5) = -0.18703481171909
        P(6) = 0.03084138183556
        P(7) = 0.03288301166689
        P(8) = -0.01059740178507
    Case 6
        P(1) = 0.11154074335011
        P(2) = 0.49462389039845
        P(3) = 0.7511339080211
        P(4) = 0.3152503517092
        P(5) = -0.22626469396544
        P(6) = -0.12976686756727
        P(7) = 0.09750160558732
        P(8) = 0.02752286553031
        P(9) = -0.03158203931749
        P(10) = 0.00055384220116
        P(11) = 0.00477725751095
        P(12) = -0.00107730108531
    Case 8
        P(1) = 0.05441584224311
        P(2) = 0.31287159091432
        P(3) = 0.67563073629732
        P(4) = 0.58535468365422
        P(5) = -0.01582910525638
        P(6) = -0.28401554296158
        P(7) = 0.00047248457391
        P(8) = 0.12874742662049
        P(9) = -0.01736930100181
        P(10) = -0.0440882539308
        P(11) = 0.0139810279174
        P(12) = 0.00874609404741
        P(13) = -0.00487035299345
        P(14) = -0.00039174037338
        P(15) = 0.00067544940645
        P(16) = -0.00011747678412
    Case 10
        P(1) = 0.02667005790055
        P(2) = 0.18817680007763
        P(3) = 0.52720118893158
        P(4) = 0.68845903945344
```

```
                P(5) = 0.28117234366057
                P(6) = -0.24984642432716
                P(7) = -0.19594627437729
                P(8) = 0.12736934033575
                P(9) = 0.09305736460355
                P(10) = -0.07139414716635
                P(11) = -0.02945753682184
                P(12) = 0.03321267405936
                P(13) = 0.00360655356699
                P(14) = -0.0107331754833
                P(15) = 0.00139535174707
                P(16) = 0.00199240529519
                P(17) = -0.00068585669496
                P(18) = -0.00011646685513
                P(19) = 0.00009358867032
                P(20) = -0.00001326420289
        End Select

        For i = 1 To 2 * DaubechieParam
            P(i) = P(i) * 2 ^ 0.5
        Next i

        For i = 1 To 2 * DaubechieParam
            k = Q(0) + i - 1
            Pn = 1 - k
            Pn = Pn + 1 - P(0)
            Q(i) = (-1) ^ k * P(Pn)
        Next i
        For i = 1 To 2 * DaubechieParam
            G(i) = P(2 * DaubechieParam - i + 1)
            H(i) = Q(2 * DaubechieParam - i + 1)
        Next i

    End Sub
```

6.4.3 GetFai(Fai())

This subroutine calculate $\phi_{(i)}$ at integer points i by solving the first degree equations. Fai(i) is $\phi_{(i)}$. In order to solve the equation, clsMathKusu.SolveLinearEquation is used. The subroutine is also shown below.

$$\begin{Bmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{Bmatrix} = \left(\begin{bmatrix} & [\text{Related to } p_k] & & 0 \\ & & & \vdots \\ & & & 0 \\ 1 & \cdots & \cdots & 1 \end{bmatrix} - \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & \ddots & \ddots & \vdots \\ \vdots & \ddots & 1 & 0 \\ 0 & \cdots & 0 & 0 \end{bmatrix} \right) \cdot \begin{Bmatrix} \phi_1 \\ \vdots \\ \vdots \\ \phi_n \end{Bmatrix}, \quad \phi_0 = 0, \quad n = 2N-1$$

$$\phi_{(i)} = \sum_{j=ST}^{ED} p_j \cdot \phi_{(2i-j)}, \quad ST = \begin{cases} 0 & i \le m-1 \\ 2\cdot(i-m)+1 & i > m-1 \end{cases}, \quad ED = \begin{cases} 2i & i \le m-1 \\ 2m-1 & i > m-1 \end{cases}$$

$$\operatorname{supp}\phi = [0 \quad 2N-1]$$

$$\psi_{(n)} = \sum_{k=2-2N}^{1} q_k \cdot \phi_{(2n-k)} \qquad \operatorname{supp}\psi = [1-N \quad N]$$

```
    Public Sub GetFai(ByRef Fai() As Double)
        '-------------------------------------------------
        '
```

```
        '   Calculate Fai(x) for Daubechie Wavelet
        '   at integer points.
        '-----------------------------------------------------
        Dim P(), Q(), G(), H() As Double
        Dim i, ii, St, Ed As Integer
        Dim Matrix(Me.SuppMaxFai - 1, Me.SuppMaxFai - 1) As Double
        Dim Vector(Me.SuppMaxFai - 1) As Double
        Dim clsMath As New clsMathKusu
        GetParam(P, Q, G, H)

        For i = 1 To Me.SuppMaxFai - 2
            If 2 * i < Me.SuppMaxFai Then
                St = 0
                Ed = 2 * i
            Else
                St = 2 * i - Me.SuppMaxFai
                Ed = Me.SuppMaxFai
            End If
            For ii = St To Ed
                If 2 * i - ii < Me.SuppMaxFai Then Matrix(i, 2 * i - ii) = _
                         Matrix(i, 2 * i - ii) + P(ii + 1)
            Next
            Matrix(i, i) = Matrix(i, i) - 1
        Next
        For i = 1 To Me.SuppMaxFai - 1
            Matrix(Me.SuppMaxFai - 1, i) = 1
        Next
        Vector(Me.SuppMaxFai - 1) = 1

        clsMath.SolveLinearEquation(Matrix, Vector, Fai)
        ReDim Preserve Fai(Me.SuppMaxFai + 1)
        For i = Me.SuppMaxFai To 1 Step -1
            Fai(i) = Fai(i - 1)
        Next
        Fai(0) = 0
    End Sub
```

SolveLinearEquation(Matrix(,), Vector(), Result())

This subroutine solves first degree equations.

$$[Matrix] \cdot \{x\} = \{Vector\}$$

$$\Leftrightarrow \{x\} = [Matrix]^{-1} \cdot \{Vector\}$$

```
    Public Function SolveLinearEquation(ByVal Matrix(,) As Double, ByVal Vector() As
Double, ByRef Result() As Double) As Integer
        '------------------------------------------------------
        '   Solve [Matrix]×{result}={Vector}
        '   Input Data
        '       Matrix      Factor matrix of the equation
        '       Vector()    Right side vector
        '
        '   Output Data
        '       Result()    solution
        '   Return Value
        '       =0  No error
        '       <>0 Error number for subroutine MInver
        '------------------------------------------------------
        Dim MatSize, Err As Integer
        Dim Det As Double
        MatSize = UBound(Vector)
        ReDim Result(MatSize)
```

```
        MInver(Matrix, MatSize, 0.000000001, Det, Err, MatSize)
        SolveLinearEquation = Err
        If Err <> 0 Then Exit Function

        MatTimesVect(Matrix, Vector, MatSize, Result, MatSize)

    End Function
```

MInver(Matrix(,), UseMatSize, Eps, Det, Err, MatSize)

This subroutine calculates inverse matrix. Eps should be small enough to get better accuracy. If some error occurs, error code will return in Err, otherwise Err is zero.

```
    Public Sub MInver(ByRef Matrix(,) As Double, ByVal UseMatSize As Integer, ByVal Eps
As Double, ByRef Det As Double, ByRef Err As Integer, ByVal MatSize As Integer)
        '-------------------------------------------
        '   Calculate inverse Matrix
        '   Input Data
        '       Matrix      Original matrix (index start from 1)
        '       UseMatSize  Maximum index to be calculated
        '       Eps         Error Tolerance
        '       MatSize     Size of the matrix
        '
        '   Output Data
        '       Matrix      Inverse matrix
        '       Det         Determinant
        '       Err         Error number
        '-------------------------------------------

        Dim Work(500), iw, J, K, Lr, jj, i As Integer
        Dim M, N As Integer

        M = UseMatSize
        N = MatSize

        Dim W, Wmax, Pivot, Api As Double

        Dim A(N, M) As Double
        For J = 1 To M
            For jj = 1 To M
                A(J, jj) = Matrix(J, jj)
1020:       Next jj
1010:   Next J
        If (M < 1 Or M > 500 Or Eps <= 0.0#) Then
            Err = 999
            MsgBox("Matrix is Too Big Or Tolerance Less Than 0.0")
        ElseIf M = 1 Then
            If A(1, 1) = 0.0# Then
                MsgBox("Matrix(0,0)=0.0 !!!")
                Exit Sub
            End If
            A(1, 1) = 1 / A(1, 1)
        Else
            Err = 0
            Det = 1
            For i = 1 To M
                Work(i) = i
10:         Next i
            For K = 1 To M
                Wmax = 0
                For i = K To M
```

```
                    W = Abs(A(i, K))
                    If W > Wmax Then
                        Wmax = W
                        Lr = i
                    End If
20:             Next i
                Pivot = A(Lr, K)
                Api = Abs(Pivot)
                If Api <= Eps Then
                    Err = 1
                    Return
                End If
                If Lr <> K Then
                    iw = Work(K)
                    Work(K) = Work(Lr)
                    Work(Lr) = iw
                    For J = 1 To M
                        W = A(K, J)
                        A(K, J) = A(Lr, J)
                        A(Lr, J) = W
30:                 Next J
                End If
                For i = 1 To M
                    A(K, i) = A(K, i) / Pivot
40:             Next i
                For i = 1 To M
                    If i <> K Then
                        W = A(i, K)
                        If W <> 0 Then
                            For J = 1 To M
                                If J <> K Then A(i, J) = A(i, J) - W * A(K, J)
50:                         Next J
                            A(i, K) = -W / Pivot
                        End If
                    End If
60:             Next i
                A(K, K) = 1 / Pivot
70:         Next K
            For i = 1 To M
80:             K = Work(i)
                If K <> i Then
                    iw = Work(K)
                    Work(K) = Work(i)
                    Work(i) = iw
                    For J = 1 To M
                        W = A(J, i)
                        A(J, i) = A(J, K)
                        A(J, K) = W
90:                 Next J
                    GoTo 80
                End If
100:        Next i
        End If
        '
        For J = 1 To M
            For jj = 1 To M
                Matrix(J, jj) = A(J, jj)
            Next jj
        Next J

    End Sub
```

MatTimesVect(Matrix(,), Vector(), VectorSize, Result(), ResultSize)

This subroutine calculates product of matrix and vector. The size of matrix is automatically defined from the size of vector to be multiplied and result vector.

$$\left\{ \text{Result} \right\}\text{ResultSize} = \left[\quad Matrix \quad \right] \left\{ \text{Vector} \right\}\text{VectorSize}$$

```
    Public Sub MatTimesVect(ByVal Matrix(,) As Double, ByVal Vector() As Double, ByVal
VectorSize As Integer, ByRef Result() As Double, ByVal ResultSize As Integer)
        '---------------------------------------------------------
        '   Calculate [Matrix] * {Vector}
        '
        '   Input Data
        '       Matrix      Two dimensional matrix
        '       Vector      Vector to be multiplied
        '       VectorSize  Number of element of the vector
        '       ResultSize  Number of element of the result vector
        '
        '   Output Data
        '       Result()    Solution vector
        '---------------------------------------------------------
        Dim i, ii As Integer
        ReDim Result(ResultSize)

        For i = 1 To ResultSize
            For ii = 1 To VectorSize
                Result(i) = Result(i) + Matrix(i, ii) * Vector(ii)
            Next
        Next
    End Sub
```

6.4.4 GetCk0(F(), Ck0())

The $c_k^{(0)}$ ($f_{0(x)} = \sum_k c_k^{(0)} \cdot \phi_{(x-k)}$) is calculated by this subroutine with the following equation.

$$\hat{c}_m^{(0)} = \frac{\hat{f}_{0(m)}}{\hat{\phi}_{(n-1)}} = \frac{\hat{f}_{(m)}}{\hat{\phi}_{(n-1)}}$$

$c_k^{(0)}$ is the inverse Fourier transform of $\hat{c}_m^{(0)}$.

```
    Public Sub GetCk0(ByVal F() As Double, ByRef Ck0() As Double)
        '---------------------------------------------------
        '   Calculate Co(k) from
        '
        '   c̄m = f̄o(m) / Φ̄(1,1)(m)
        '   Input Data
        '       F()     Original signal(F(1)～F(N))、F(0) is the first Index
        '
        '   Output Data
        '       Ck0()   sequence  Ck(0) is the first index.
        '---------------------------------------------
        Dim FFT As New clsFourier
        Dim FReal(), FImage(), FaiReal(), FaiImage(), Ck0Image() As Double
```

```
        Dim FF(), Dt, Df, FaiReal2() As Double
        Dim i, MaxNum, MaxNum2, stp As Integer
        Dt = 1
        MaxNum = UBound(F)
        GetFai(FaiReal2)

        MaxNum2 = Log(MaxNum) / Log(2)
        MaxNum2 = 2 ^ MaxNum2

        ReDim FReal(MaxNum2), FImage(MaxNum2)
        For i = 1 To MaxNum2
            stp = Int(i / MaxNum)
            FReal(i) = F(i - stp * MaxNum)
        Next

        FFT.Fourier(FReal, FImage, FF, True, True, Dt, Df)

        ReDim FaiReal(MaxNum2), FaiImage(MaxNum2)

        For i = 1 To MaxNum2
            If i > UBound(FaiReal2) + 1 Or i < 1 Then
                FaiReal(i) = 0
            Else
                FaiReal(i) = FaiReal2(i - 1)
            End If
        Next


        FFT.Fourier(FaiReal, FaiImage, FF, True, True, Dt, Df)

        FFT.DivideImage(FReal, FImage, FaiReal, FaiImage, Ck0, Ck0Image)

        FFT.Fourier(Ck0, Ck0Image, FF, False, True, Dt, Df)
        ReDim Preserve Ck0(MaxNum)

    End Sub
```

6.5 clsFourier

This is the class to conduct Fourier transform. You can see other useful subroutines and functions, which are not used for the Wavelet transform.

6.5.1 Fourier(CReal(), CImage(), F(), Forward, Periodic, Dt, Df)

This is the subroutine for the Fourier transform. The actual FFT (fast Fourier transform) is conducted by subroutine FAST shown later.

If Forward is “True”, the subroutine calculates (forward) Fourier transform, if “False”, the subroutine calculates inverse Fourier transform.

[Forward transform]

To send

CReal() original data in temporal domain

CImage()	not needed
F()	not needed
Forward	True
Periodic	if "true" data is considered periodic
Dt	Time interval
Df	Not needed

To return

CReal()	Real part of the Fourier transformed data
CImage()	Image part of the Fourier transformed data
F()	Fourier amplitude
Forward	True
Periodic	none
Dt	Time interval
Df	Frequency interval

[Forward transform]

To send

CReal()	Real part of the Fourier transformed data
CImage()	Image part of the Fourier transformed data
F()	Not needed
Forward	False
Periodic	Not needed
Dt	Not needed
Df	Frequency interval

To return

CReal()	data in temporal domain
CImage()	none
F()	none
Forward	False
Periodic	none
Dt	Time interval
Df	Frequency interval

```
    Public Sub Fourier(ByRef CReal() As Double, ByRef CImage() As Double, ByRef F() As
Double, ByVal Forward As Boolean, ByVal Periodic As Boolean, ByRef Dt As Double, ByRef
Df As Double)
        Dim N, NT, K, NFold As Integer
        N = UBound(CReal)
        If Forward = True Then
            NT = 2
            Do While NT < N
                NT = NT * 2
            Loop
            ReDim Preserve CReal(NT), CImage(NT)
            If Periodic = True Then
                For K = N + 1 To NT
                    CReal(K) = CReal(K - N)
                Next
            End If
            Fast(NT, CReal, CImage, -1)
            NFold = NT / 2
            ReDim F(NFold)
            For K = 1 To NFold
                F(K - 1) = (CReal(K) ^ 2 + CImage(K) ^ 2) ^ 0.5 * Dt
            Next
            Df = 1 / (NT * Dt)
        Else
            NT = 2
            Do While NT < N
                NT = NT * 2
            Loop
            ReDim Preserve CReal(NT), CImage(NT)
            For K = 1 To NT
                CReal(K) = CReal(K) / NT
                CImage(K) = CImage(K) / NT
            Next
            Fast(NT, CReal, CImage, 1)
            Dt = 1 / NT / Df
        End If

    End Sub
```

Fast(N, Realx(), ImageX(), IND)

This subroutine conducts Fast Fourier Transform (FFT).

```
    Private Sub Fast(ByVal N As Integer, ByRef Realx() As Double, ByRef ImageX() As Double,
ByVal IND As Integer)
        '----------------------------------------------------------------
        '   Subroutine for Fast Fourier Transfer
        '
        '   Input Data
        '       N           Number of Data, which must be 2^i
        '       IND         =+1 : Inverse Fourier, =-1 : Fourier
        '       RealX()     Real value
        '       ImageX()    Image value
        '
        '   Output Data
        '       RealX()     Real value (Fourier value is multiplied by N)
        '       ImageX()    Image value (Fourier value is multiplied by N)
        '
        '-----------------------------------------------------------------
        Dim Temp(1), Temp1(1), Theta(1) As Single
        Dim Mfast As Integer
        Dim K, Kmax, IStep As Integer
        Dim i, j As Integer
        j = 1
```

```vb
        For i = 1 To N
            If i < j Then
                Temp(0) = Realx(j)
                Temp(1) = ImageX(j)
                Realx(j) = Realx(i)
                ImageX(j) = ImageX(i)
                Realx(i) = Temp(0)
                ImageX(i) = Temp(1)
            End If
            Mfast = N / 2
Line120:
            If j <= Mfast Then GoTo Line130
            j = j - Mfast
            Mfast = Mfast / 2
            If Mfast >= 2 Then GoTo Line120
Line130:
            j = j + Mfast
        Next i

        Kmax = 1
Line150:
        If Kmax >= N Then
            Exit Sub
        End If
        IStep = Kmax * 2
        For K = 1 To Kmax
            Theta(0) = 0.0#
            Theta(1) = 3.141593 * (IND * (K - 1.0#) / Kmax)
            For i = K To N Step IStep
                j = i + Kmax
                Temp1(0) = Exp(Theta(0)) * Cos(Theta(1))
                Temp1(1) = Exp(Theta(0)) * Sin(Theta(1))
                Temp(0) = Realx(j) * Temp1(0) - ImageX(j) * Temp1(1)
                Temp(1) = Realx(j) * Temp1(1) + ImageX(j) * Temp1(0)
                Realx(j) = Realx(i) - Temp(0)
                ImageX(j) = ImageX(i) - Temp(1)
                Realx(i) = Realx(i) + Temp(0)
                ImageX(i) = ImageX(i) + Temp(1)
            Next i
        Next K
        Kmax = IStep
        GoTo Line150

    End Sub
```

6.5.2 DivideImage(XReal(), XImage(), DividerReal(), DividerImage(), ResultReal(), ResultImage())

This subroutine conducts division of image value.

```vb
    Public Function DivideImage(ByVal XReal() As Double, ByVal XImage() As Double, ByVal
DividerReal() As Double, ByVal DividerImage() As Double, ByRef ResultReal() As Double,
ByRef ResultImage() As Double) As Boolean
        '-------------------------------------------------------------------
        '
        '   x+yi = (a+bi)/(c+di)
        '
        '       x=(ac+bd)/Z
        '       y=(bc-ad)/Z
        '       Z=c^2+d^2
        '
```

```
        '-----------------------------------------------------------------------
        Dim MaxNum, i As Integer
        Dim Factor As Double
        MaxNum = UBound(XReal)
        If MaxNum <> UBound(XImage) Or MaxNum <> UBound(DividerReal) Or MaxNum <>
UBound(DividerImage) Then
            DivideImage = False
            Exit Function
        End If
        ReDim ResultReal(MaxNum), ResultImage(MaxNum)

        DivideImage = True

        For i = 1 To MaxNum
            Factor = DividerReal(i) ^ 2 + DividerImage(i) ^ 2
            If Factor <> 0 Then
                ResultReal(i) = (XReal(i) * DividerReal(i) + XImage(i) *
DividerImage(i)) / Factor
                ResultImage(i) = (XImage(i) * DividerReal(i) - XReal(i) *
DividerImage(i)) / Factor
            Else
                DivideImage = False
                Exit For
            End If
        Next

    End Function
```

6.6 clsPolynomial

This class is to conduct calculations with polynomials. This class is called from the clsSpline to calculate cardinal B-spline.

6.6.1 Solve(Num, A(), X())

This subroutine solves the following equation.

$$\sum_{k=0}^{n} A_{(k)} \cdot x^{k} = 0$$

"Num" is the number of the solutions to be calculated. The solution is calculated in order closer to zero.

```
    Public Sub Solve(ByVal Num As Integer, ByVal A() As Double, ByRef X() As Double)
        '-------------------------------------
        '   Solve F(x)=Sigma{A(k)*x^k}=0
        '   A()     Factor matrix
        '   Num     Num. Of solution to be calculated. If <0 or >order, calculate all
        '   return data
        '   X()     solutions
        '-------------------------------------

        FindX(Num, A, X)
    End Sub
```

FindX(Num, A, X)

This is to solve polynomial with the Newton Method. The solutions are calculated in order closer to zero.

As for the Newton method, the solution of equation $f_{(x)} = 0$, x_s is calculated as follows.

i) The initial value for x_s, x_{init} is assumed.

ii) The modifier for x_s, Δx_s is calculated as follows.

$$\Delta x_s = \frac{f_{(x_S)}}{f'_{(x_S)}}$$

iii) x_s is modified as follows.

$$x_s \leftarrow x_s - \Delta x_s$$

iv) If $\left| f_{(x_S)} \right| < tol$, x_s is the solution. If not, go back to ii).

When one solution is obtained, the order of the polynomial can be reduced as follows.

$$
\begin{array}{c|ccccc}
 & x^n \times & x^{n-1} \times & \cdots & & x^0 \times \\
 & A_{(n)} & A_{(n-1)} & \cdots & & A_{(0)} \\
x_s & & B_{(n-1)} \cdot x_s & \cdots & & B_{(0)} \cdot x_s \\
\hline
 & B_{(n-1)} & B_{(n-2)} & \cdots & B_{(0)} & B_{(-1)} \\
 & = A_{(n)} & = A_{(n-1)} - B_{(n-2)} \cdot x_s & & & = A_{(0)} - B_{(0)} \cdot x_s \\
 & & & & & = 0
\end{array}
$$

where, $A_{(i)}$ is the factor of the polynomial before reduced, and $B_{(i)}$ is after reduced. Since x_s is one solution, $B_{(-1)} = A_{(0)} - B_{(0)} \cdot x_s$ must be small enough. Here, the iv) of the Newton method is replaced by the following.

iv) If $\left| B_{(-1)} \right| < tol$, x_s is the solution. If not, go back to ii).

```
    Private Function FindX(ByVal Num As Integer, ByVal A() As Double, ByRef X() As Double)
As Boolean
        '-------------------------------------
        '   Solve polynomial with Newton method
        '   Input Data
        '   Num     Num. Of solution to be calculated. If <0 or >order, calculate all
        '   A()     Factor of polynomial
        '   return data
        '   X()     Solution
        '-------------------------------------
        Dim XInit As Double, B() As Double, Tol As Double
        Dim i, ii As Integer
        Tol = 0.0000001
        XInit = 0
        If Num <= 0 Or Num > UBound(A) Then Num = UBound(A)
        ReDim X(Num)
        For i = 1 To Num
            X(i) = Converge(XInit, Tol, A, B)
            ReDim A(UBound(B))
            For ii = 0 To UBound(B)
```

```
                A(ii) = B(ii)
            Next
        Next
    End Function
```

Converge (XInit, Tol, A(), B())

This function returns the solution closest to the XInit. B() is also the return value, which is the factor of one order reduced polynomial.

```
    Private Function Converge(ByVal XInit As Double, ByVal Tol As Double, ByVal A() As
Double, ByRef B() As Double) As Double
        '--------------------------------------
        '   Solve polynomial with Newton method
        '   XInit is the initial value for x.
        '   Input Data
        '   XInit   initial value
        '   Tol     F(x)=Tol tolerance
        '   A()     factor of polynomial
        '   Output Data
        '   B()     one order reduced factor of polynomial
        '   RETURN
        '   approximate solution
        '--------------------------------------
        Dim X, XOld As Double
        XOld = XInit
        X = XOld - Modifier(XOld, A)
        Do While Abs(CalcB0(X, A, B)) > Tol
            XOld = X
            X = XOld - Modifier(XOld, A)
        Loop
        Converge = X
    End Function
```

Modifier(x, A())

This function returns the following value to modify the value of Newton method.

$$\Delta x_s = \frac{f_{(x_s)}}{f'_{(x_s)}}$$

```
    Private Function Modifier(ByVal x As Double, ByVal A() As Double) As Double
        '--------------------------------------
        '   calculate the modification factor for the Newton method
        '   F(x)/F'(x)
        '--------------------------------------
        If DFx(x, A) = 0 Then
            Modifier = 0
            Exit Function
        End If
        Modifier = Fx(x, A) / DFx(x, A)
    End Function
```

DFx(x, A())

This function returns the differentiation of $f_{(x)} = \sum_{k=0}^{N} A_{(k)} \cdot x^k$.

```
Private Function DFx(ByVal x As Double, ByVal A() As Double) As Double
    '-------------------------------------
    '   F'(x)=i*A(i)*x^(i-1)   i=1~N
    '-------------------------------------
    Dim i As Integer
    DFx = 0
    For i = 1 To UBound(A)
        DFx = DFx + A(i) * x ^ (i - 1) * i
    Next
End Function
```

Fx(x, A())

This function returns the value of $f_{(x)} = \sum_{k=0}^{N} A_{(k)} \cdot x^k$.

```
Private Function Fx(ByVal x As Double, ByVal A() As Double) As Double
    '-------------------------------------
    '   F(x)=A(i)*x^i    i=0~N
    '-------------------------------------
    Dim i As Integer
    Fx = 0
    For i = 0 To UBound(A)
        Fx = Fx + A(i) * x ^ i
    Next
End Function
```

CalcB0(X, A(), B())

This function returns the value of $B_{(-1)} = A_{(0)} - B_{(0)} \cdot x_s$. At the same time, the factor for the one order reduced polynomial, $B_{(i)}$ is also calculated.

```
Private Function CalcB0(ByVal X As Double, ByVal A() As Double, ByRef B() As Double)
As Double
    '-------------------------------------
    '   Calculate the factor for the one order reduced polynomial, B()
    '    and B(-1)
    '-------------------------------------
    Dim i As Integer, C As Double
    ReDim B(UBound(A) - 1)
    B(UBound(B)) = A(UBound(A))
    For i = UBound(A) To 2 Step -1
        B(i - 2) = B(i - 1) * X + A(i - 1)
    Next
    CalcB0 = B(0) * X + A(0)
End Function
```

6.6.2 Multiply(A(), B(), C())

This subroutine calculates following product of polynomials.

$$\left(\sum_{k=1}^{N_A} A_{(k)} \cdot x^{(k-1)+A(0)}\right) \times \left(\sum_{k=1}^{N_B} B_{(k)} \cdot x^{(k-1)+B(0)}\right) = \sum_{k=1}^{N_C} C_{(k)} \cdot x^{(k-1)+C(0)}$$

Note that $A_{(0)}$, $B_{(0)}$, and $C_{(0)}$ are not the factors for x^0, but indicate the lowest order of x in the polynomials.

```
Public Sub Multiply(ByVal A() As Double, ByVal B() As Double, ByRef C() As Double)
    Dim i, j, NumC, OrdA, OrdB, OrdC As Integer
    ReDim C(0)
    C(0) = A(0) + B(0)

    For i = 1 To UBound(A)
        OrdA = A(0) + i - 1
        For j = 1 To UBound(B)
            OrdB = B(0) + j - 1
            OrdC = OrdA + OrdB
            NumC = OrdC - C(0) + 1
            If NumC > UBound(C) Then ReDim Preserve C(NumC)
            C(NumC) = C(NumC) + A(i) * B(j)
        Next
    Next
End Sub
```

6.7 clsSpline

This is the class to conduct calculations for the cardinal B-spline wavelet.

6.7.1 Properties

Parameter	R&W	The order of cardinal B-spline
ApproxBeta	R&W	β_k is zero if $\lvert k \rvert > approxbeta$
Approx	R&W	α_k is zero if $\lvert k \rvert > approx$
Periodic	R&W	if "true", the original signal is considered "Periodic" signal.
SuppMaxFai	R	The upper end of support of ϕ. (=2N-1)
SuppMaxPsai	R	The upper end of support of ψ. (=N)
SuppMinFai	R	The lower end of support of ϕ. (=0)
SuppMinPsai	R	The lower end of support of ψ. (=1-N)

Local varies for the properties.

LM=Parameter

BetaTol=ApproxBeta

MaxN=Approx

6.7.2 GetParam(P(), Q(), G(), H())

This subroutine calculate sequences p_k, q_k, g_k, and h_k.

```
    Public Sub GetParam(ByRef P() As Double, ByRef Q() As Double, ByRef G() As Double,
ByRef H() As Double)
        CalcP(P, LM)
        CalcQ(Q, LM)
        CalcG(G, LM, MaxN)
        CalcH(H, LM, MaxN)
    End Sub
```

CalcP(P(), M)

This subroutine calculates the sequence “P_k” with the following equation.

$$p_k = \frac{1}{2^{m-1}} \sum_{k=0}^{m} \binom{m}{k} \qquad k = 0,1,\cdots,m$$

```
    Private Sub CalcP(ByRef P() As Double, ByVal M As Integer)
        '-------------------------------------------------
        '   p(k)=1/2^(m-1) * mCk    [0,m] Num. Of Data: m+1
        '-------------------------------------------------
        Dim k As Integer
        ReDim P(M + 1)
        P(0) = 0
        For k = 0 To M
            P(k + 1) = 1.0 / 2 ^ (M - 1) * Binomial(M, k)
        Next
    End Sub
```

Binomial(M, K).

This function calculates binomial equation and returns the result.

```
    Private Function Binomial(ByVal M As Integer, ByVal K As Integer) As Double
        '-------------------------------------------------
        '   Calculate binomial equation
        '   mCk=m!/k!/(m-k)!
        '-------------------------------------------------
        Dim i As Integer
        Binomial = 1
        For i = M - K + 1 To M
            Binomial = Binomial * i
        Next
        For i = 1 To K
            Binomial = Binomial / i
        Next
    End Function
```

CalcQ(Q(), M)

This subroutine calculates the sequence “q_k” with the following equation.

$$q_k = \frac{1}{2^{m-1}} \sum_{\ell=0}^{m} N_{2m(k-\ell+1)} \cdot \binom{m}{\ell} \qquad k = 0,1,\cdots,3m-2$$

```
    Private Sub CalcQ(ByRef Q() As Double, ByVal M As Integer)
```

```
        '------------------------------------------------
        '                          m
        '   q(k)=(-1)^k / 2^(m-1) * Sigma{mCL*N2m(k+1-L)}
        '                          L=0
        '
        '          [0,3m-2],Num of Data: 3m-1
        '------------------------------------------------
        Dim k, L As Integer
        ReDim Q(3 * M - 1)
        Q(0) = 0
        For k = 0 To 3 * M - 2
            Q(k + 1) = 0
            For L = 0 To M
                Q(k + 1) = Q(k + 1) + Binomial(M, L) * Nm(2 * M, k + 1 - L)
            Next
            Q(k + 1) = Q(k + 1) * (-1) ^ k / 2 ^ (M - 1)
        Next
    End Sub
```

Nm(M, X)

This function calculates the m-th order cardinal B-spline value at X.

$$N_{m(x)} = \frac{1}{(m-1)!}\sum_{k=0}^{m}(-1)^k\binom{m}{k}(x-k)_+^{m-1}$$

```
    Private Function Nm(ByVal M As Integer, ByVal X As Double) As Double
        '-------------------------------------------------
        '   the cardinal B-spline of m-th order
        '
        '                m
        '   Nm(x)=1/(m-1)!*Sigma{(-1)^k*mCk*((x-k)+)^(m-1)}
        '                k=0
        '
        '-------------------------------------------------
        Dim k As Integer
        Nm = 0
        For k = 0 To M
            Nm = Nm + (-1) ^ k * Binomial(M, k) * CutoffPower(X - k, M - 1)
        Next
        Nm = Nm / Factorial(M - 1)
    End Function
```

CutoffPower

This function calculates a truncated power function as follows, and returns the result.

$$x_+^m = (x_+)^m$$

$$x_+ = \max(0,\ x)$$

```
    Private Function CutoffPower(ByVal X, ByVal M) As Double
        '------------------------------------------------
        '   (x+)^m
        '   x+=max{0,x}
        '------------------------------------------------
        If X < 0 Then
            CutoffPower = 0
            Exit Function
        End If
```

```
        CutoffPower = X ^ M
    End Function
```

Factorial

This function calculates the factorial and returns the result.

$$x! = x \cdot (x-1) \cdot \cdots \cdot 2 \cdot 1$$

```
    Private Function Factorial(ByVal X As Integer) As Integer
        '-------------------------------------------------
        '   x!
        '-------------------------------------------------
        Dim i
        Factorial = 1
        For i = 1 To X
            Factorial = Factorial * i
        Next
    End Function
```

CalcG

This subroutine calculates the sequence "g_k" with the following equations.

$$\begin{aligned}\Leftrightarrow G_{(z)} &= \frac{1}{2}\sum_{k\in Z} g_k \cdot z^k \\ &= \frac{E_{N_m(z)}}{E_{N_m\left(z^2\right)}}\overline{P}_{(z)} \\ &= \frac{1}{2^m}\frac{\sum_{k=-m+1}^{m-1} N_{2m(m+k)} \cdot z^k \cdot \sum_{\ell=0}^{m}\binom{m}{\ell} \cdot z^{-\ell}}{E_{N_m\left(z^2\right)}}\end{aligned}$$

$$\alpha_k = (2m-1)!\sum_{i=1}^{m-1} C_i \cdot a_i^{|k|}$$

```
    Private Sub CalcG(ByRef G() As Double, ByVal M As Integer, ByVal Order As Integer)
        '---------------------------------------------
        '
        '   G(z)=ENm(z)/ENm(z^2)*P̄(z) = 1/2 Sigma Gk * z^k
        '
        '   1/ENm(z^2)=(2m-1)!*Sigma αk * z^2k
        '   ENm(z)=E2m-1(z)/(2m-1)!/z^m-1
        '   P(z)=(1+z^-1)^m/2^m
        '
        '   Calculate Gk
        '
        '   Input Data
        '       M       Spline order
        '       Order   Alpha k is assumed to be zero when |k|>Order.
        '
        '   Output
        '       G       Gk
        '---------------------------------------------
        Dim Alpha() As Double, Enm() As Double, PkB() As Double, GD() As Double
        Dim i, j As Integer
```

```
        Dim Poly As New clsPolynomial
        Dim Dummy As Double
        Alphak(Order, M, Alpha)
        GetENm(M, Enm)

        CalcPkB(M, PkB)

        Poly.Multiply(Alpha, Enm, GD)
        Poly.Multiply(GD, PkB, G)
        For i = 1 To UBound(G)
            G(i) = G(i) * 2
        Next
    End Sub
```

AlphaK(N, M, Alpha)

This subroutine calculates the approximation of $1/E_{Nm}(z^2)$, α_k. The sequence has infinite elements, therefore if $|k| > N$, α_k is assumed to be zero in order to make the sequence finite.

$$\alpha_k = (2m-1)!\sum_{i=1}^{m-1} C_i \cdot a_i^{|k|}$$

$$C_1 = \frac{1}{(a_1 - a_i^{-1})} \qquad m = 2\,, \quad C_i = \prod_{k=1}^{m-1} \frac{1}{(a_k - a_i^{-1})} \prod_{i \neq j} \frac{a_j}{a_i - a_j} \quad m \geq 3$$

$$N_{2m-1(z)} = \prod_{i=1}^{m-1} (z - a_i)(z - a_i^{-1})$$

```
    Private Sub Alphak(ByVal N As Integer, ByVal M As Integer, ByRef Alpha() As Double)
        '-------------------------------------------------
        '              n
        '   1/ENm(z^2)=Sigma  αk z^(2k)
        '             k=-n
        '
        '   Alpha(0) is the smallest order of z (=-2n)
        '   Num. Of data elements: 4n+1
        '   Input data
        '   N   Order of approximation
        '   M   order of cardinal B-spline
        '-------------------------------------------------
        Dim i, k As Integer
        Dim A(), Factor As Double
        ReDim Alpha(N * 2 * 2 + 1)
        Factor = Factorial(2 * M - 1)
        E2m1Solve(M, M - 1, A)
        Alpha(0) = -N * 2
        For k = -N To N Step 1
            Alpha(k * 2 + 2 * N + 1) = 0
            For i = 1 To M - 1
                Alpha(k * 2 + N * 2 + 1) = Alpha(k * 2 + N * 2 + 1) + Ci(i, M, A) * A(i)
^ Abs(k) * Factor
            Next
        Next
    End Sub
```

E2m1Solve(M, Num, X())

This subroutine solve the following equation to get solutions a_i. “M” is the order of the cardinal B-spline, Num is the number of solutions to get, and X() is the vector of solutions.

$$N_{2m-1(z)} = \prod_{i=1}^{m-1} (z - a_i)(z - a_i^{-1}) = 0$$

```
Private Sub E2m1Solve(ByVal M As Integer, ByVal Num As Integer, ByRef X() As Double)
    '-------------------------------------------------
    '   Solve
    '   E2m-1(z)=0
    '   The number of solutions to get is Num.
    '   Solution is found in order of closer to zero
    '-------------------------------------------------
    Dim A() As Double, i As Integer
    Dim Polynomial As New clsPolynomial

    E2m1Factor(M, A)

    Polynomial.Solve(Num, A, X)

End Sub
```

E2m1Factor(M, A())

This subroutine calculates the factor of z of E_{2m-1} as follows. “M” is the order of the cardinal B-spline and A() is the vector of the factor.

$$E_{2m-1} = (2m-1)! \cdot \sum_{k=0}^{2m-2} N_{2m(k+1)} \cdot z^k$$

$$= \sum_{k=0}^{2m-2} A_{(k+1)} \cdot z^k$$

```
Private Sub E2m1Factor(ByVal M As Integer, ByRef A() As Double)
    '-------------------------------------------------
    '                       2m-2
    '   E2m-1(z)=(2m-1)!ΣN2m(k+1) z^k
    '                       k=0
    '
    '    Calculate the factors for z of E2m-1.
    '   E2m-1(z)=ΣA(k) z^k
    '
    '-------------------------------------------------
    ReDim A(2 * M - 2)
    Dim k As Integer, Factor As Double
    '        Dim Spline As New clsSpline
    Factor = Factorial(2 * M - 1)
    For k = 0 To 2 * M - 2
        A(k) = Nm(2 * M, k + 1) * Factor
    Next
End Sub
```

Ci(i, M, A())

This function calculates the factor C_i to calculate α_k. C_i is calculated as follows.

$$C_1 = \frac{1}{(a_1 - a_i^{-1})} \qquad m = 2\,, \quad C_i = \prod_{k=1}^{m-1} \frac{1}{(a_k - a_i^{-1})} \prod_{i \neq j} \frac{a_j}{a_i - a_j} \quad m \geq 3$$

```
    Private Function Ci(ByVal i As Integer, ByVal M As Integer, ByVal A() As Double) As
Double
        '------------------------------------------------
        '   Calculate Ci in order to approximate 1/ENm(z)
        '
        '   Input data
        '   i   element index to calculate Ci
        '   M   order of the cardinal B-spline
        '   A() Solutions of E2m-1(z)=0. m-1 solutions are needed
        '
        '   Output
        '   Ci
        '------------------------------------------------
        Dim k As Integer
        Ci = 1
        If M = 2 Then
            Ci = 1 / (A(1) - 1 / A(1))
        Else
            For k = 1 To M - 1
                Ci = Ci / (A(k) - 1 / A(i))
                If i <> k Then
                    Ci = Ci * A(k) / (A(i) - A(k))
                End If
            Next
        End If
    End Function
```

GetENm(M, Enm())

This subroutine calculates the factor for z^k for the $E_{N_{m(z)}}$. $E_{N_{m(0)}}$ is the smallest order of z.

$$E_{N_{m(z)}} = \sum_{k=-m+1}^{m-1} N_{2m(m+k)} \cdot z^k$$

$$E_{N_{m(0)}} = -m + 1$$

```
    Private Sub GetENm(ByVal M As Integer, ByRef Enm() As Double)
        '------------------------------------------------
        '         m-1
        '   Enm(z)=Sigma {N2m(m+k) * z^k}
        '        k=-m+1
        '
        '   Enm(0)  =-m+1
        '------------------------------------------------
```

```
        Dim k, stp As Integer
        ReDim Enm(2 * M - 2 + 1)
        Enm(0) = -M + 1
        stp = 1
        For k = -M + 1 To M - 1 Step 1
            Enm(stp) = Nm(2 * M, M + k)
            stp = stp + 1
        Next
    End Sub
```

CalcPkB(M, P())

This subroutine calculates the factor of z for the conjugate sequence of p_k.

$$\overline{P}_{(z)} = \frac{1}{2}\sum_{k\in Z}\overline{p}_k \cdot z^{-k}$$

```
    Private Sub CalcPkB(ByVal M As Integer, ByRef P() As Double)
        '---------------------------------------------------
        '
        '   P̄(z)=(1+z^-1)^m / 2^m  =Sigma p̄k*z^k
        '
        '   Input Data
        '       M   Order of the cardinal B-spline
        '
        '   Output
        '       P   Pk
        '---------------------------------------------------
        Dim i As Integer, Factor As Double
        ReDim P(M + 1)
        P(0) = -M
        Factor = 1 / 2 ^ M
        For i = 0 To M
            P(i + 1) = Factor * Binomial(M, i)
        Next
    End Sub
```

CalcH(H(), M, Order)

This subroutine calculates the sequence “h_k” with the following equation.

$$\begin{aligned} H_{(z)} &= \frac{1}{2}\sum_{k\in Z} h_k \cdot z^k \\ &= -z^{-2m+1} \cdot \frac{P_{(-z)}}{E_{N_m}\left(z^2\right)} \\ &= -z^{-2m+1} \cdot \frac{1}{E_{N_m}\left(z^2\right)} \frac{1}{2^m} \sum_{k=0}^{m} \binom{m}{k} \cdot (-z)^k \end{aligned}$$

```
Private Sub CalcH(ByRef H() As Double, ByVal M As Integer, ByVal Order As Integer)
    '----------------------------------------------
    '
    '   H(z)=-Z^(-2m+1) * P(-z) / ENm(z^2) = 1/2 Sigma hk * z^k
    '
    '   1/ENm(z^2)=(2m-1)!*Sigma Alphak * z^2k
    '   P(-z)=(1-z)^m/2^m
    '
    '
    '   Input Data
    '       M       Order of the cardinal B-spline
    '       Order   if |k|>Order, Alpha k is assumed to be zero.
    '
    '   Output
    '       H       hk
    '----------------------------------------------
    Dim Alpha() As Double, Pk() As Double
    Dim i As Integer, Factor As Double
    Dim Poly As New clsPolynominal
    Alphak(Order, M, Alpha)
    CalcPk(M, Pk)
    Poly.Multiply(Alpha, Pk, H)
    For i = 1 To UBound(H)
        H(i) = H(i) * (-2)
    Next
    H(0) = H(0) - 2 * M + 1
End Sub
```

CalcPk(M, P())

This subroutine calculates the factor of z^k for the p_k.

$$p_k = \frac{1}{2^{m-1}} \sum_{k=0}^{m} \binom{m}{k} \quad k = 0,1,\cdots,m$$

```
Private Sub CalcPk(ByVal M As Integer, ByRef P() As Double)
    '----------------------------------------------
    '
    '   P(z)=(1-z)^m / 2^m  =Sigma pk*z^k
    '
    '   Input Data
    '       M   Order of the cardinal B-spline
    '
    '   Output
    '       P   Pk
    '----------------------------------------------
    Dim i As Integer, Factor As Double
    ReDim P(M + 1)
    P(0) = 0
    Factor = 1 / 2 ^ M
    For i = 0 To M
        P(i + 1) = (-1) ^ i * Factor * Binomial(M, i)
    Next
End Sub
```

6.7.3 GetCk0(F(), Ck0())

The $c_k^{(0)}$ ($f_{0(x)} = \sum_k c_k^{(0)} \cdot \phi_{(x-k)}$) is calculated by this subroutine with the following equation.

$$c_\ell^{(0)} = \sum_k f_{(\ell)} \cdot \beta^{(2m)}_{\ell+\frac{m}{2}-k}$$

$$B_{m(z)} = \sum_k \beta_k^{(m)} \cdot z^k = \sum_{k=-n}^{n} \alpha_k \cdot z^k$$

$$\alpha_k = (2m-1)! \sum_{i=1}^{m-1} C_i \cdot a_i^{|k|}$$

$$C_1 = \frac{1}{\left(a_1 - a_i^{-1}\right)} \qquad m=2, \quad C_i = \prod_{k=1}^{m-1} \frac{1}{\left(a_k - a_i^{-1}\right)} \prod_{i \neq j} \frac{a_j}{a_i - a_j} \quad m \geq 3$$

$$N_{2m-1(x)} = \prod_{i=1}^{m-1} (z - a_i)\left(z - a_i^{-1}\right)$$

$$N_{m(x)} = \frac{1}{(m-1)!} \sum_{k=0}^{m} (-1)^k \binom{m}{k} (x-k)_+^{m-1}, \quad \binom{m}{k} = {}_mC_k = \frac{m!}{k!(m-k)!}$$

```
Public Sub GetCk0(ByVal F() As Double, ByRef Ck0() As Double)
    '-----------------------------------------------------
    '   According to the interpolating function, calculate
    '       Ck(0)=Sigma Beta k+2-L * f(L)
    '
    '   Input Data
    '       F()     Original signal (F(1)~F(N)), F(0)is the Index for the first data
    '
    '   Output Data
    '       Ck0()   Interpolating function. Ck(0)is the index for the first data.
    '-----------------------------------------------------
    Dim i, j, NumF, M As Integer
    Dim Beta() As Double, order As Double, Periodic As Boolean
    M = LM
    order = BetaTol
    Periodic = LPeriodic
    BetaK(M, order, Beta)
    NumF = UBound(F)
    ReDim Ck0(NumF)
    Ck0(0) = F(0)
    For i = 1 To NumF
        Ck0(i) = C0k(F(0) + i - 1, order, Beta, F, Periodic)
    Next
End Sub
```

BetaK(M, Order, Beta())

This subroutine calculates the β_k as follows.

$$B_{m(z)} = \sum_k \beta_k^{(m)} \cdot z^k = \sum_{k=-n}^{n} \alpha_k \cdot z^k$$

```vb
    Private Sub BetaK(ByVal M As Integer, ByVal Order As Integer, ByRef Beta() As Double)
        '-------------------------------------------------
        '   Calculate the beta of
        '
        '         (m-1)! z[(m-1)/2]   (2n-1)!z^(n-1)
        '   Bm(z)=-----------------=-----------------=Sigma Beta(k)Z^k
        '              Em-1(z)           E2n-1(z)
        '
        '   Input Data
        '       M       Order of the cardinal B-spline, must be odd number
        '       Order   if |k|>Order, Beta(k)=zero
        '
        '   Output Data
        '       Beta    Beta(k)   Beta(0) is the index for the first data
        '                         Num. Of elements: 2*order+1
        '-------------------------------------------------
        Dim N As Integer, A() As Double, i As Integer, k As Integer, Factor As Double
        N = M / 2
        ReDim Beta(Order * 2 + 1)
        Beta(0) = -Order
        If M = 2 Then
            Beta(Order + 1) = 1
        Else
            E2m1Solve(N, Order, A)  '   A() is the solution of E2n-1(z)=0
            Factor = Factorial(2 * N - 1)
            For k = -Order To Order Step 1
                Beta(k + Order + 1) = 0
                For i = 1 To N - 1
                    Beta(k + Order + 1) = Beta(k + Order + 1) + Factor * Ci(i, N, A) *
 A(i) ^ Abs(k)
                Next
            Next

        End If
    End Sub
```

C0K(k, Order, Beta(), F(), Periodic)

This subroutine actually calculate the $c_k^{(0)}$ with the following equation.

$$c_\ell^{(0)} = \sum_k f_{(\ell)} \cdot \beta^{(2m)}_{\ell+\frac{m}{2}-k}$$

```vb
    Private Function C0k(ByVal k As Integer, ByVal Order As Integer, ByVal Beta() As
 Double, ByVal F() As Double, ByVal Periodic As Boolean) As Double
        '-------------------------------------------------
        '   Calculate
        '   Ck(0)=Sigma Beta k+2-L×f(L)
        '   for calculating the interpolating function.
        '
        '   Input Data
        '       k       element number
        '       Order   if |k|>Order, Beta(k) is assumed to be zero
        '       beta()  Factors to calculate the interpolating equation
        '       F()     Original signal. F(0) is the index for the first data
        '       Periodic    If true, the original signal is periodic function.
        '-------------------------------------------------
        Dim i, j, NumF, St, Ed As Integer
        NumF = UBound(F)
        St = F(0)
        Ed = NumF + St - 1
```

```
        COk = 0
        For i = -Order To Order Step 1
            j = k + 2 - i
            If j < 0 Then
                If Periodic = True Then
                    Do While j < St
                        j = j + NumF
                    Loop
                Else
                    j = -1
                End If
            ElseIf j > Ed Then
                If Periodic = True Then
                    Do While j > Ed
                        j = j - NumF
                    Loop
                Else
                    j = -1
                End If
            End If
            If j <> -1 Then COk = COk + Beta(i + Order + 1) * F(j - St + 1)
        Next
    End Function
```

6.7.4 GetFai(Fai())

This subroutine calculates $\phi_{(i)}$ at integer points i. Fai(i) is $\phi_{(i)}$. The subroutine is also shown below.

$$\phi_{N_m(n)} = N_{m(n)} = \frac{1}{(m-1)!}\sum_{k=0}^{m}(-1)^k\binom{m}{k}(n-k)_+^{m-1} \quad \binom{m}{k} = {}_mC_k = \frac{m!}{k!(m-k)!}, \quad \mathrm{supp}\, N_m = [0 \quad m]$$

```
    Public Sub GetFai(ByRef Fai() As Double)
        '----------------------------------------------------
        '
        '   Calculate Fai(x) for the cardinal B-spline Wavelet
        '   at integer points.
        '   Fai(0):The index number for the first data.
        '----------------------------------------------------
        Dim i As Integer
        Dim Dx As Double
        ReDim Fai(Me.SuppMaxFai + 1)
        For i = 0 To Me.SuppMaxFai
            Dx = i
            Fai(i + 1) = Nm(LM, Dx)
        Next
    End Sub
```

6.8 clsMother

This class conducts calculation related to the mother wavelet. clsDaubechie and clsSpline are used in this class.

6.8.1 Properties

Parameter R&W The order of Daubechies. Only 2, 4, 6, 8, and 10 is available

ApproxAlpha	R&W	α_k is zero if $\lvert k \rvert > approx$
ApproxBeta	R&W	β_k is zero if $\lvert k \rvert > approxbeta$
Periodic	R&W	if "true", the original signal is considered "Periodic" signal.
MotherType	R&W	Type of Mother Wavelet.
SuppMaxFai	R	The upper end of support of ϕ. (=2N-1)
SuppMaxPsai	R	The upper end of support of ψ. (=N)
SuppMinFai	R	The lower end of support of ϕ. (=0)
SuppMinPsai	R	The lower end of support of ψ. (=1-N)

Local varies for the properties.

LM=Parameter

BetaTol=ApproxBeta

AlphaTol=ApproxAlpha

lclMother=MotherType

6.8.2 GetCk0(F(), Ck0())

The $c_k^{(0)}$ ($f_{0(x)} = \sum_k c_k^{(0)} \cdot \phi_{(x-k)}$) is calculated by this subroutine. F() is the original signal.

```
    Public Sub GetCk0(ByVal F() As Double, ByRef Ck0() As Double)
        '------------------------------------------------------------------
        '   Calculate the interpolating sequence {Co(k)}
        '------------------------------------------------------------------
        Select Case lclMother
            Case Mother.Daubechie
                Daub.GetCk0(F, Ck0)
            Case Mother.Spline
                Spline.GetCk0(F, Ck0)
        End Select

    End Sub
```

6.8.3 GetParam(P(), Q(), G(), H())

This subroutine calculate sequences p_k, q_k, g_k, and h_k.

```
    Public Sub GetParam(ByRef P() As Double, ByRef Q() As Double, ByRef G() As Double,
ByRef H() As Double)
        '------------------------------------------------------------------
        '   Calculate the sequences {pk},{qk},{gk},{hk}
        '------------------------------------------------------------------
        Select Case lclMother
            Case Mother.Daubechie
                Daub.GetParam(P, Q, G, H)
            Case Mother.Spline
                Spline.GetParam(P, Q, G, H)
        End Select
```

```
    End Sub
```

6.8.4 CalcFaiAtInt(Fai(), Psai())

This subroutine calculates $\phi_{(i)}$ and $\psi_{(i)}$ at integer points i. Fai(i) is $\phi_{(i)}$. Psai() is $\psi_{(i)}$. $\phi_{(0)}$ and $\psi_{(0)}$ are the index number for the first data.

```
    Public Sub CalcFaiAtInt(ByRef Fai() As Double, ByRef Psai() As Double)
        Dim P(), Q(), G(), H(), FaiN() As Double
        GetParam(P, Q, G, H)
        GetFaiN(FaiN)
        CalcFaiPsaiAtInt(1, Me.SuppMaxFai, Me.SuppMinFai, P, FaiN, Fai)
        CalcFaiPsaiAtInt(1, Me.SuppMaxPsai, Me.SuppMinPsai, Q, FaiN, Psai)

    End Sub
```

GetFaiN(FaiN())

This subroutine calculates the $\phi_{(i)}$ at the integer points, i.

```
    Private Sub GetFaiN(ByRef FaiN() As Double)
        Select Case Me.MotherType
            Case Mother.Daubechie
                Daub.GetFai(FaiN)
            Case Mother.Spline
                Spline.GetFai(FaiN)
        End Select
    End Sub
```

CalcFaiPsaiAtInt(j, SuppMax, SuppMin, P(), FaiN(), Fai())

This subroutine calculates $\phi_{(i)}$ or $\psi_{(i)}$ (depending on which sequence will be sent as P()).

```
    Private Sub CalcFaiPsaiAtInt(ByVal j As Integer, ByVal SuppMax As Integer, ByVal
SuppMin As Integer, ByVal P() As Double, ByVal FaiN() As Double, ByRef Fai() As Double)
        '-----------------------------------------------------------
        '   Fai(n)=Sigma{pk * Fai(2^j * n-k)}
        '
        '   Input Data
        '       j           Order
        '       P()         Pk
        '       SuppMax     Upper bound of n (max. value)
        '       SuppMin     Lower bound of n (min. value)
        '       faiN        Fai(n)
        '
        '   Output Data
        '       Fai()       Fai or Psai
        '       Fai(0)      The index for the first data
        '
        '   Note !
        '       Fai(n) will not be calculated and be zero if 2^j*n is not integer.
        '       Be careful if j is negative.
        '       In order to calculate all Fai(n/2^j) in order, use CalcFaiPsai
```

```
    '-----------------------------------------------------------
    Dim i, SuppM, PointNum, N As Integer
    Dim clsWL As New clsWLCalc

    SuppM = SuppMax - SuppMin
    ReDim Fai(SuppM + 1)
    Fai(0) = SuppMin
    For i = 0 To SuppM
        N = SuppMin + i
        Fai(i + 1) = CalcPkFaiAtInt(j, N, P, FaiN)
    Next
End Sub
```

CalcPkFaiAtInt(J, N, PkJ(), Fai())

This function calculates the ϕ with specific parameters, J, N, and PkJ() and returns it.

```
Private Function CalcPkFaiAtInt(ByVal J As Integer, ByVal N As Integer, ByVal PkJ()
As Double, ByVal Fai() As Double) As Double
    '---------------------------------------------------------
    '   Fai(N)=Sigma {Pk * Fai(2^j * n-k)}
    '
    '   Input Data
    '       j       Order
    '       N       n
    '       PkJ()   Sequence Pk
    '       Fai()   fai(n), Fai at the integer point n
    '   Note !
    '       Fai(n) will not be calculated and be zero if 2^j*n is not integer.
    '       Be careful if j is negative.
    '       In order to calculate all Fai(n/2^j) in order, use CalcPkFai
    '-----------------------------------------------------------
    Dim i, k, Pos, ii As Integer
    CalcPkFaiAtInt = 0
    For i = 1 To UBound(Fai) Step 1
        ii = Fai(0) + i - 1
        k = 2 ^ J * N - ii
        '           k = N - ii
        If N * 2 ^ J = Int(N * 2 ^ J) Then
            Pos = k - PkJ(0) + 1
            If Pos > 0 And Pos <= UBound(PkJ) Then
                CalcPkFaiAtInt = CalcPkFaiAtInt + PkJ(Pos) * Fai(i)
            End If
        End If
    Next
End Function
```

6.8.5 CalcFai(MinNum, XFai(), Fai(), XPSai(), PSai())

This subroutine calculates accurate $\phi_{(i)}$ and $\psi_{(i)}$ with the multi-scaling technique. The number of required data points is defined by MinNum. The number calculated actually should be 2^k, therefore, the MinNum is the number data points required at least.

```vb
    Public Sub CalcFai(ByVal MinNum As Integer, ByRef XFai() As Double, ByRef Fai() As
Double, ByRef XPsai() As Double, ByRef Psai() As Double)
        '-------------------------------------------------------------------
        '   Calculate accurate mother wavelet and scaling function
        '
        '   Input Data
        '       MinNum      Num. Of data points required at least
        '
        '   Output Data
        '       XFai()      X axis value for Fai
        '       Fai()       Fai
        '       XPsai()     X axis value for PSai
        '       Psai()      Psai
        '-------------------------------------------------------------------
        Dim P(), Q(), G(), H() As Double
        GetParam(P, Q, G, H)

        CalcFaiPsai(MinNum, Me.SuppMaxFai, Me.SuppMinFai, P, P, Fai, XFai)
        CalcFaiPsai(MinNum, Me.SuppMaxPsai, Me.SuppMinPsai, Q, P, Psai, XPsai)

    End Sub
```

CalcFaiPSai

This subroutine calculates accurate $\phi_{(i)}$ or $\psi_{(i)}$ with the multi-scaling technique depending on PInit(). The number of required data points is defined by MinNum. The number calculated actually should be 2^k, therefore, the MinNum is the number data points required at least.

$$\phi_{\left(\frac{n}{2^j}\right)} = \sum_k p_k^{(j)} \cdot \phi_{(n-k)}$$

$$\psi_{\left(\frac{n}{2^j}\right)} = \sum_k q_k^{(j)} \cdot \phi_{(n-k)}$$

```vb
    Private Sub CalcFaiPsai(ByVal MinNum As Integer, ByVal SuppMax As Integer, ByVal
SuppMin As Integer, ByVal PInit() As Double, ByVal P() As Double, ByRef Fai() As Double,
ByRef X() As Double)
        '-----------------------------------------------------------
        '   Calclate following equation to get accurate Fai and Psai
        '   Fai(n/2^j)=Sigma{pk(j) * Fai(n-k)}
        '
        '   Input Data
        '       MinNum      Required num of data points
        '       P()         Pk
        '       PInit()     Initial sequence, Pk or Qk
        '
        '   Output Data
        '       Fai()       Fai or Psai depending on PInit()
        '       X()         X axis value
        '----------------------------------------------------------
        Dim i, j, SuppM, PointNum, N As Integer
        Dim Pj() As Double, FaiN() As Double
        Dim clsWL As New clsWLCalc

        GetFaiN(FaiN)
        SuppM = SuppMax - SuppMin
        If MinNum = 0 Then
```

```
            MinNum = SuppM
        Else
            j = Int(Log(MinNum / SuppM) / Log(2) + 0.99999)
        End If
        If j < 1 Then j = 1
        PointNum = SuppM * 2 ^ j
        ReDim Fai(PointNum), X(PointNum)
        clsWL.CalcPkJ(P, PInit, Pj, j)
        For i = 0 To PointNum
            X(i) = SuppMin + i / 2 ^ j
            N = SuppMin * 2 ^ j + i
            Fai(i) = CalcPkFai(N, Pj, FaiN)
        Next
    End Sub
```

CalcPkFai(N, PkJ(), Fai())

This function calculates the following function with one specific n, and returns it.

$$CalcPkFai = \sum_k p_k^{(j)} \cdot \phi_{(n-k)}$$

```
    Private Function CalcPkFai(ByVal N As Integer, ByVal PkJ() As Double, ByVal Fai()
As Double) As Double
        '-------------------------------------------------------
        '   Calculate
        '   Fai(N/2^j)=Sigma{Pk(j) * Fai(n-k)}
        '
        '   Input Data
        '       N       n
        '       PkJ()   Sequence Pk(j)
        '       Fai()   The value of Fai(i) at integer data points
        '-------------------------------------------------------
        Dim i, k, Pos, ii As Integer
        CalcPkFai = 0
        For i = 1 To UBound(Fai) Step 1
            ii = Fai(0) + i - 1
            k = N - ii
            Pos = k - PkJ(0) + 1
            If Pos > 0 And Pos <= UBound(PkJ) Then
                CalcPkFai = CalcPkFai + PkJ(Pos) * Fai(i)
            End If
        Next
    End Function
```

6.8.6 GetFj(j, Ck(), FjNum, Fj())

This subroutine calculates the series of $f_{j(x)}$ with the following function.

$$f_{j(x)} = \sum_\ell c_\ell^{(j)} \cdot \phi_{\left(2^j x - \ell\right)}$$

```
    Public Sub GetFj(ByVal j As Integer, ByVal Ck() As Double, ByVal FjNum As Integer,
ByRef Fj() As Double)
        '--------------------------------------------------------------------
        '
        '   Fj(x)=Σck(j) * φ(2^j * x-k)
        '
        '   Input Data
        '       j       Rank of the Wavelet
        '       Ck()    Sequence for the j-th order
```

```
        '       FjNum   Num. Of data of Fj()
        '
        '   Output Data
        '       Fj()    Fj(x)
        '------------------------------------------------------------------
        Dim FaiN() As Double
        GetFaiN(FaiN)
        ReDim Fj(FjNum)
        CalcFaiPsaiInOrder(FjNum - 1, 0, Ck, FaiN, Fj)
    End Sub
```

CalcFaiPsaiInOrder(SuppMax, SuppMin, P(), FaiN(), Fai())

This subroutine calculate the following function. Depending on P() ($p_k^{(j)}$) and Fai() ($\phi_{(n-k)}$, the return sequence is ϕ , ψ , $f_{j(x)}$, or $g_{j(x)}$. If multi-scaling is applied (depending on $p_k^{(j)}$), the increment of these is not always 1 (in other words, they are not the value at integer points).

$$\phi_{\left(\frac{n}{2^j}\right)} = \sum_k p_k^{(j)} \cdot \phi_{(n-k)}$$

```
    Private Sub CalcFaiPsaiInOrder(ByVal SuppMax As Integer, ByVal SuppMin As Integer,
ByVal P() As Double, ByVal FaiN() As Double, ByRef Fai() As Double)
        '----------------------------------------------------------
        '   Fai(n/2^j)=Sigma {pk * φ(n-k)}
        '
        '   Input Data
        '       P()         Pk
        '       SuppMax     Upper support (Max of n)
        '       SuppMin     Lower support (Min of n)
        '       faiN()      Fai(n) at integer points
        '
        '   Output Data
        '       Fai()       Fai, Psai, Fj, or Gj. The increment is not always 1.
        '       Fai(0)      Index for the first element of Fai(1)
        '
        '---------------------------------------------------------
        Dim i, SuppM, PointNum, N As Integer
        Dim clsWL As New clsWLCalc

        SuppM = SuppMax - SuppMin
        ReDim Fai(SuppM + 1)
        Fai(0) = SuppMin
        For i = 0 To SuppM
            N = SuppMin + i
            Fai(i + 1) = CalcPkFai(N, P, FaiN)
        Next
    End Sub
```

6.8.7 GetGj(j, Dk(), GjNum, Gj())

This subroutine calculates $g_{j(x)}$ with following function.

$$g_{j(x)} = \sum_k d_k^{(j+\ell)} \cdot \phi_{\left(2^{j+\ell}x-k\right)}$$

```
    Public Sub GetGj(ByVal j As Integer, ByVal Dk() As Double, ByVal GjNum As Integer,
```

```
ByRef Gj() As Double)
        '--------------------------------------------------------------------
        '
        '   Gj(x)=Sigma Dk(j) * Psai(2^j * x-k)
        '
        '   Input Data
        '       j       Rank of wavelet
        '       Dk()    Factor sequence for the j-th rank
        '       GjNum   Num. Of data of Gj
        '
        '   Output Data
        '       Gj()    Gj(x)
        '--------------------------------------------------------------------
        Dim FaiN(), PSai() As Double
        Dim P(), Q(), G(), H() As Double
        GetParam(P, Q, G, H)
        GetFaiN(FaiN)
        CalcFaiPsaiAtInt(1, Me.SuppMaxPsai, Me.SuppMinPsai, Q, FaiN, PSai)
        ReDim Gj(GjNum)
        CalcFaiPsaiInOrder(GjNum - 1, 0, Dk, PSai, Gj)
    End Sub
```

6.9 clsWLCalc

This class is to conduct decomposition and reconstitution of the Wavelet transform.

6.9.1 Reconstitution(P(), Q(), Cold(), DOld(), C(), Periodic)

The $c_k^{(j)}$ is reconstituted from $c_k^{(j-1)}$ and $d_k^{(j-1)}$ with following function.

$$c_k^{(j)} = \left(p * c^{(j-1)\uparrow}\right)_k + \left(q * d^{(j-1)\uparrow}\right)_k$$

```
    Public Sub Reconstitution(ByVal P() As Double, ByVal Q() As Double, ByVal COld() As
Double, ByVal DOld() As Double, ByRef C() As Double, ByVal Periodic As Boolean)
        '----------------------------------------
        '   Reconstitution algorism
        '            ^         ^
        '   c(k)=(p*c|)(k)+(g*d|)(k)
        '   Input Data
        '       P()     Sequence pk
        '       Q()     Sequence qk
        '       COld()  One rank higher Ck
        '       DOld()  One rank higher Dk
        '       Periodic    if true, the signal is considered periodic
        '
        '   Output Data
        '       C()     reconstituted Ck
        '----------------------------------------
        Dim CUp(), DUp() As Double
        Dim CC(), DD() As Double
        Ak2LCL(P, COld, CC, Periodic)
        Ak2LCL(Q, DOld, DD, Periodic)
        SumUpVector(CC, DD, C)
        ChopSmallValue(C, 0.00000000000001)
    End Sub
```

Ak2LCL(A(), C(), AC(), Periodic)

This subroutine calculates following convolusion and upsampling.

$$AC_k = \sum A_{k-2\ell} \cdot C_\ell = \left(A * C^{\uparrow}\right)_k$$

```
    Private Sub Ak2LCL(ByVal A() As Double, ByVal C() As Double, ByRef AC() As Double,
ByVal Periodic As Boolean)
        '---------------------------------------------
        '
        '   ACk=Sigma Ak-2L * CL
        '              ^
        '      =(A*C|)k
        '
        '---------------------------------------------
        Dim CUp() As Double
        UpSampling(C, CUp, Periodic)
        Convolve(A, CUp, AC, Periodic)
    End Sub
```

UpSampling(A(), B(), Periodic)

This subroutine conducts upsampling.

$$c^{\uparrow}_{2k} = c_k$$

$$c^{\uparrow}_{2k+1} = 0$$

```
    Private Sub UpSampling(ByVal A() As Double, ByRef B() As Double, ByVal Periodic As
Boolean)
        '----------------------------------------
        '   Upsampling A()
        '   A(0) is the index of the first element
        '   B(2k)=A(k)
        '   B(2k+1)=0
        '   If Periodic=True, add last element of zero
        '----------------------------------------
        Dim i As Integer
        If Periodic = True Then
            ReDim B(2 * UBound(A))
        Else
            ReDim B(2 * UBound(A) - 1)
        End If
        B(0) = A(0) * 2
        For i = 1 To UBound(A)
            B(i * 2 - 1) = A(i)
        Next i
    End Sub
```

Convolve(A(), B(), C(), Periodic)

This subroutine conducts convolusion.

$$c_k = (a * b)_k = \sum_{\ell} a_{k-\ell} \cdot b_\ell$$

```
    Private Sub Convolve(ByVal A() As Double, ByVal B() As Double, ByRef C() As Double,
ByVal Periodic As Boolean)
        '-----------------------------------------------------
        '   Convolusion
        '   Ck=(A*B)k=Sigma Ak-L * BL
        '                L
        '
```

```
    '   If Periodic=true, C is considered as periodic signal
    '---------------------------------------------------------
    Dim i, ii, k, j, L As Integer, AA As Double, BB As Double
    Dim AAn As Integer, BBn As Integer
    If Periodic = True Then
        ReDim C(UBound(B))
    Else
        ReDim C(UBound(A) + UBound(B) - 1)
    End If
    C(0) = A(0) + B(0)
    If Periodic = True Then
        For ii = 1 To UBound(C)
            k = C(0) + ii - 1
            For i = 1 To UBound(A)
                AAn = A(0) + i - 1
                L = k - AAn
                BBn = L + 1 - B(0)
                Do While BBn <= 0
                    BBn = BBn + UBound(B)
                Loop
                Do While BBn > UBound(B)
                    BBn = BBn - UBound(B)
                Loop
                C(ii) = C(ii) + A(i) * B(BBn)
            Next
        Next
    Else
        For ii = 1 To UBound(C)
            k = C(0) + ii - 1
            For i = 1 To UBound(A)
                AAn = A(0) + i - 1
                L = k - AAn
                BBn = L + 1 - B(0)
                If BBn > 0 And BBn <= UBound(B) Then
                    C(ii) = C(ii) + A(i) * B(BBn)
                End If
            Next
        Next
    End If
End Sub
```

ChopSmallValue(A(), Tol)

This subroutine chops off small elements (less than Tol) at both ends of the sequence (A()). The convolusion can make a small computational error. This subroutine is used to chop off these elements.

```
Private Sub ChopSmallValue(ByRef A() As Double, ByVal Tol As Double)
    Dim i
    Do While Abs(A(1)) < Tol And UBound(A) > 1
        A(0) = A(0) + 1
        ShiftVector(A, 1)
    Loop
    Do While Abs(A(UBound(A))) < Tol And UBound(A) > 1
        ReDim Preserve A(UBound(A) - 1)
    Loop
End Sub
```

ShiftVector(Vector(), Pos)

This subroutine deletes an element at the position of Pos of the vector Vector().

```
    Private Sub ShiftVector(ByVal Vector() As Double, ByVal Pos As Integer)
        Dim i
        For i = Pos To UBound(Vector) - 1
            Vector(i) = Vector(i + 1)
        Next
        ReDim Preserve Vector(UBound(Vector) - 1)
    End Sub
```

6.9.2 Decomposition(G(), H(), Cold(), C(), D(), Periodic)

This subroutine conducts decomposition algorism as follows.

$$c_k^{(j-1)} = \frac{1}{2}\left(g * c^{(j)}\right)_k^{\downarrow}$$

$$d_k^{(j-1)} = \frac{1}{2}\left(h * c^{(j)}\right)_k^{\downarrow}$$

```
    Public Sub Decomposition(ByVal G() As Double, ByVal H() As Double, ByVal COld() As
Double, ByRef C() As Double, ByRef D() As Double, ByVal Periodic As Boolean)
        '------------------------------------------
        '   Decomposition algorism
        '   c(k)=1/2*(g*c)|(k)
        '   d(k)=1/2*(h*c)|(k)
        '
        '   Input Data
        '       G()     Decomposition sequence gk
        '       H()     Decomposition sequence hk
        '       COld()  Lower rank Ck
        '       Periodic    If true, the signal is considered periodic
        '
        '   Output Data
        '       C()     Decomposed Ck
        '       D()     Decomposed Dk
        '------------------------------------------
        Dim CDown(), DDown() As Double
        Convolve(G, COld, CDown, Periodic)
        Convolve(H, COld, DDown, Periodic)
        DownSampling(CDown, C)
        DownSampling(DDown, D)
        MultipleVector(C, 0.5)
        MultipleVector(D, 0.5)
    End Sub
```

DownSampling

This subroutine conducts downsampling.

$$c_k^{\downarrow} = c_{2k}$$

```
    Private Sub DownSampling(ByVal A() As Double, ByRef B() As Double)
        '------------------------------------------
        '   Downsample B()
        '   A(0) is index for the first element.
        '   B(k)=a(2k)
        '------------------------------------------
```

```
        Dim i As Integer, An As Integer, Bn As Integer
        ReDim B(Int(UBound(A) / 2 + 0.5))
        B(0) = Int(A(0) / 2 + 0.5)
        For i = 1 To UBound(A)
            An = A(0) + i - 1              'Actual index of each element
            If Int(An / 2) * 2 = An Then    'If odd element
                Bn = An / 2 - B(0) + 1
                B(Bn) = A(i)
            End If
        Next
    End Sub
```

MultipleVector

This function multiplies a factor to the all elements of vector A().

```
    Private Function MultipleVector(ByRef A() As Double, ByVal Factor As Double)
        Dim i
        For i = 1 To UBound(A)
            A(i) = A(i) * Factor
        Next
    End Function
```

6.9.3 CalcPkJ(P(), PInit(), Pj(), j)

This subroutine calculates multi-scaling algorism.

$$p_k^{(j)} = \sum_{\ell} p_{k-2\ell} \cdot p_{\ell}^{(j-1)}, \qquad p_k^{(1)} = p_k$$

$c_k^{(j+\ell)}$, $d_k^{(j+\ell)}$, $q_k^{(j)}$ are also calculated with this subroutine. They are depending on P() and PInit().

```
    Public Sub CalcPkJ(ByVal P() As Double, ByVal PInit() As Double, ByRef Pj() As Double,
ByVal j As Integer)
        '-------------------------------------------
        '   Pk(j)=Sigma {Pk-2L * PL(j-1)}
        '       Pk(1)=Pk
        '
        '   Input Data
        '       P()     Sequence Pk
        '       PInit() Initial sequence of Pj
        '       j       rank
        '
        '   Output Data
        '       Pj      j-th order pk(j)
        '-------------------------------------------
        Dim i, ii As Integer
        Dim PjDammy() As Double
        ReDim Pj(UBound(PInit))
        For i = 0 To UBound(PInit)
            Pj(i) = PInit(i)
        Next

        For i = 2 To j
            Ak2LCL(P, Pj, PjDammy, False)
            ReDim Pj(UBound(PjDammy))
```

```
            For ii = 0 To UBound(Pj)
                Pj(ii) = PjDammy(ii)
            Next
        Next
    End Sub
```

6.10 clsWaveLet

This is the main class to conduct Wavelet transform. This class should be defined in your program. Other classes are all used under this class. All properties must be set and signal to be transformed must be registered to the class before the wavelet transform is conducted.

6.10.1 Events

This class has four events.

```
Public Event PropertyChanged(ByVal PropertyName As String)
```

When some properties are changed, this event occurs.

```
Public Event Reconstituted(ByVal Depth As Integer)
```

When reconstitution is conducted, this event occurs.

```
Public Event Decomposed(ByVal Depth As Integer)
```

When decomposition is conducted, this event occurs.

```
Public Event FunctionRegistered()
```

When new signal is registered to the class, this event occurs.

6.10.2 Properties

Parameter	R&W	The order of Daubechies. Only 2, 4, 6, 8, and 10 is available
AlphaNum	R&W	α_k is zero if $\lvert k \rvert > approx$
BetaNum	R&W	β_k is zero if $\lvert k \rvert > approxbeta$
Periodic	R&W	if “true”, the original signal is considered “Periodic” signal.
MotherType	R&W	Type of Mother Wavelet.
MaxDepth	R	Max rank to be able to decompose
CalculationDepth	R&W	Max rank to decompose
TimeIncrement	R&W	Time increment of the signal

Frequency(j)	R	Frequency increment of the j-th rank
ReadyToDecompose	R	When true, the data is ready to decompose
AlreadyDecomposed	R	If true, the data has been decomposed already
AlreadyReconstituted	R	If true, the data has been reconstituted already
CalcGo	R&W	Wavelet transform is conducted only when CalcGo is True

Local varies for the properties.

lclMaxDepth=MaxDepth

CalcDepth=CalculationDepth

lclTimeIncrement=TimeIncrement

lclCalcGo=CalcGo

Local Varies

Mother	clsMother
WLCalc	clWLCalc
DecomposedFlag	if not decomposed, =false
ReconstitutedFlag	if not reconstituted, =false
lclF()	original signal
C()()	sequence Ck
D()()	sequence Dk

```
Public Enum MType As Integer
    Daubechie
    Spline
End Enum
```

6.10.3 GetFaiPsai(MinNum, XFai(), Fai(), XPsai(), Psai())

This subroutine calculates accurate mother wavelet and scaling function. MinNum is the minimum required number of data points. The number should be 2^k, if the number is not 2^k, the number will be automatically adjusted. Other arguments are:

XFai() X axis values for $\phi_{(x)}$

Fai() $\phi_{(x)}$

XPsai() X axis values for $\psi_{(x)}$

Psai() $\psi_{(x)}$

```
    Public Sub GetFaiPsai(ByRef MinNum As Integer, ByRef XFai() As Double, ByRef Fai()
As Double, ByRef XPsai() As Double, ByRef Psai() As Double)
        Mother.CalcFai(MinNum, XFai, Fai, XPsai, Psai)
    End Sub
```

6.10.4 RegisterFunction(F())

This function register the original signal to the class. This function must be called before conducting wavelet transform.

```
Public Function RegisterFunction(ByVal F() As Double) As Boolean
    ReDim lclF(UBound(F))
    Dim KeepFlag As Boolean
    Dim i As Integer
    KeepFlag = lclCalcGo
    For i = 0 To UBound(F)
        lclF(i) = F(i)
    Next
    If lclMaxDepth <> Int(Log(UBound(F)) / Log(2)) Then
        lclMaxDepth = Log(UBound(F)) / Log(2)
        lclCalcGo = False
        RaiseEvent PropertyChanged("MaxDepth")
    End If
    If lclMaxDepth < 1 Then
        RegisterFunction = False
    Else
        RegisterFunction = True
        DecomposedFlag = False
        ReconstitutedFlag = False
        lclCalcGo = KeepFlag
        RaiseEvent FunctionRegistered()
        RaiseEvent PropertyChanged("Function")
    End If
End Function
```

GetFunction(F())

The registered signal can be gotten with this function. If you change the value of returned function, the value in the class will be also changed. If you do not want to change the value in the class, “GetFunctionIndependently” should be used. This function is much faster than “GetFunctionIndependently”.

```
Public Function GetFunction(ByRef F() As Double) As Boolean
    If lclF Is Nothing Then Return False

    F = lclF
    Return True
End Function
```

GetFunctionIndependently(F())

The registered singal can be obtained with this function. Obtained signal is independent on the value in the class (not shared).

One override function is also defined.

```
    Public Function GetFuntionIndependently(ByRef F() As clsStructures.DblXYPoint) As
Boolean
```

```
    Public Function GetFuntionIndependently(ByRef F() As Double) As Boolean
        If lclF Is Nothing Then Return False

        ReDim F(UBound(lclF))
        Dim i As Integer
        For i = 0 To UBound(lclF)
            F(i) = lclF(i)
        Next
        Return True
    End Function
```

6.10.5 EnforcedDecomposition

Decomposition is conducted with this function, even if the decomposition is already made. “Decomposition” should be used if it should be checked that decomposition has done or not.

```
    Public Function EnforcedDecomposition() As Boolean
        DecomposedFlag = False
        Return Decomposition()
    End Function
```

6.10.6 Decomposition

This function decomposes the signal to the rank of CalcDepth. If the decomposition has been made with the same signal and parameters, nothing will be done.

```
    Public Function Decomposition() As Boolean
        Dim i, j As Integer
        Dim G(), H(), Q(), P() As Double

        If DecomposedFlag = True Then
            Decomposition = True
            Exit Function
        End If
        If lclMaxDepth = 0 Or UBound(lclF) = Nothing Then
            Decomposition = False
            Exit Function
        End If

        If CalcDepth > lclMaxDepth Then CalcDepth = lclMaxDepth
        ReDim C(0), D(0)
        Mother.GetParam(P, Q, G, H)
        Mother.GetCk0(lclF, C(0))
        For i = 1 To CalcDepth
            ReDim Preserve C(i), D(i)
            WLCalc.Decomposition(G, H, C(i - 1), C(i), D(i), Mother.Periodic)
        Next

        Decomposition = True
        RaiseEvent Decomposed(UBound(D))
```

```
        DecomposedFlag = True
        ReconstitutedFlag = False
    End Function
```

6.10.7 EnforcedReconstitution

Reconstitution is conducted with this function, even if the Reconstitution is already made. “Reconstitution” should be used if it should be checked that Reconstitution has done or not.

```
    Public Function EnforcedReconstitution() As Boolean
        ReconstitutedFlag = False
        Return Reconstitution()
    End Function
```

6.10.8 Reconstitution

This function reconstitutes the signal. If the reconstitution has been made with the same signal and parameters, nothing will be done.

```
    Public Function Reconstitution() As Boolean
        Dim i As Integer
        Dim G(), H(), Q(), P() As Double
        If ReconstitutedFlag = True Then
            Reconstitution = True
            Exit Function
        End If
        If UBound(D) = Nothing Or UBound(D) = 0 Then
            Reconstitution = False
            Exit Function
        End If
        Mother.GetParam(P, Q, G, H)

        For i = UBound(D) To 1 Step -1
            WLCalc.Reconstitution(P, Q, C(i), D(i), C(i - 1), Mother.Periodic)
        Next
        Reconstitution = True
        RaiseEvent Reconstituted(UBound(D))
        ReconstitutedFlag = True
    End Function
```

6.10.9 GetFj(L, j, Fj())

This function calculates $f_{j(x)}$. Another override function is also defined.

```
    Public Function GetFj(ByVal L As Integer, ByVal j As Integer, ByRef Fj() As
clsStructures.DblXYPoint) As Boolean
```

```
    Public Function GetFj(ByVal L As Integer, ByVal j As Integer, ByRef Fj() As Double)
As Boolean
```

```
        '--------------------------------------------------------------
        '   Calculate Fj
        '
        '       Fj=Sigma Ck(j+L)*Fai(2^(j+L)*x-k)
        '       Sigma Ck(j+L)=Sigma Sigma Pk-2L*CL(j+L-1)
        '   Fj is interpolated with L
        '
        '   Input Data
        '       L   Interpolating rank. If =-j, the number of data points is constant.
        '       j   wavelet rank
        '
        '   Output
        '       Fj()  Fj
        '--------------------------------------------------------------
        Dim FjNum As Integer
        If UBound(C(Abs(j))) = Nothing Then
            GetFj = False
            Exit Function
        End If
        If L > 0 Then
            Dim P(), Q(), G(), H(), Cj() As Double
            Mother.GetParam(P, Q, G, H)
            WLCalc.CalcPkJ(P, C(Abs(j)), Cj, L + 1)
            FjNum = UBound(lclF) * 2 ^ (j + L)
            Mother.GetFj(j + L, Cj, FjNum, Fj)
        Else
            FjNum = UBound(lclF) * 2 ^ j
            Mother.GetFj(j, C(Abs(j)), FjNum, Fj)
        End If
        GetFj = True
    End Function
```

6.10.10 GetGj

This function calculates $g_{j(x)}$. Another override function is also defined.

```
    Public Function GetGj(ByVal L As Integer, ByVal j As Integer, ByRef Gj() As
clsStructures.DblXYPoint)
```

```
    Public Function GetGj(ByVal L As Integer, ByVal j As Integer, ByRef Gj() As Double)
As Boolean
        '--------------------------------------------------------------
        '   Calculate Gj
        '
        '       Gj=Sigma dk(j)*Psai(2^j*x-k)
        '       Gj=Sigma dk(j+1)*Fai(2^(j+1)*x-k)
        '       Sigma dk(j+1)=Sigma Sigma qk-2L*dL(j)
        '       Gj=Sigma dk(j+L)*Fai(2^(j+L)*x-k)
        '       Sigma dk(j+L)=Sigma Sigma Pk-2L*dL(j+L-1)
        '
        '   Gj is interpolated with L.
        '
        '   Input Data
        '       L   Interpolating rank. If =-j, the num. Of data steps is constant.
        '       j   wavelet rank
        '
        '   Output
        '       Gj() Gj
        '--------------------------------------------------------------
```

```
        Dim GjNum As Integer
        If UBound(C(Abs(j))) = Nothing Then
            GetGj = False
            Exit Function
        End If
        If L > 0 Then
            Dim P(), Q(), G(), H(), Dj(), Dj2() As Double
            Dim i
            Mother.GetParam(P, Q, G, H)
            WLCalc.CalcPkJ(Q, D(Abs(j)), Dj, 2)
            If L > 1 Then
                WLCalc.CalcPkJ(P, Dj, Dj2, L)
                ReDim Dj(UBound(Dj2))
                For i = 0 To UBound(Dj)
                    Dj(i) = Dj2(i)
                Next
            End If
            GjNum = UBound(lclF) * 2 ^ (j + L)
            Mother.GetFj(j + L, Dj, GjNum, Gj)
        Else
            GjNum = UBound(lclF) * 2 ^ j
            Mother.GetGj(j, D(Abs(j)), GjNum, Gj)
        End If
        GetGj = True
    End Function
```

6.10.11 GetCj(j, Cj())

This function gives the sequence of c_k for the j-th rank.

```
    Public Function GetCj(ByVal j As Integer, ByVal Cj() As Double) As Boolean
        Dim i As Integer
        If j > UBound(C) Then
            GetCj = False
            Exit Function
        End If
        For i = 0 To UBound(C(j))
            Cj(i) = C(j)(i)
        Next
        GetCj = True
    End Function
```

6.10.12 GetDj(j, Dj())

This function gives the sequence of d_k for the j-th rank.

```
    Public Function GetDj(ByVal j As Integer, ByVal Dj() As Double) As Boolean
        Dim i As Integer
        If j > UBound(D) Then
            GetDj = False
            Exit Function
        End If
        For i = 0 To UBound(D(j))
            Dj(i) = D(j)(i)
        Next
        GetDj = True
    End Function
```

6.11 DblXYPoint

This is the type of a structure for vary. This structure is used for the data to draw graph.

```
Public Structure DblXYPoint
    Public X As Double
    Public Y As Double
End Structure
```