已知

$$V_J = W_{J-1} \oplus W_{J-2} \oplus \cdots \oplus W_0 \oplus V_0.$$

現在我們研究如何把一個在 V_j 中的元素分解到 $W_{j-1} \oplus V_{j-1}$. 這個步驟可以遞迴執行, 一直分解到足夠低的解析度為止. 反過來, 如何把 W_{j-1} 和 V_{j-1} 的元素再合成回到 V_j.

定義 $P_j : L^2(\mathbb{R}) \to V_j$ 是個 L^2-正交投影. 如果已知 $f(x)$ 的數學表達式, 我們只能處理 $f(x)$ 在有限區間內, 以有限解析度所表現的量. 現在假設只考慮 $f(x)$ 在 $\phi(x)$ 的函蓋之內的部份, 也就是, $[0,1]$. 將其投影在足夠細的解析度上, 例如 V_J, $J > 0$. 由於

$$P_J f(x) = \sum_{k=0}^{2^J-1} (f, \phi_{Jk}) \phi_{Jk}(x),$$

我們得到一組代表 $f(x)$ 的係數 $\mathbf{v} = (v_1, v_2, \ldots, v_N)^T$. 其中 $N = 2^J$, $v_k = (f, \phi_{J,k-1})$.

如果預備處理的資料已經是離散形式的, 可以想像它是某一個在 $L^2(\mathbb{R})$ 的函數在某個 V_J 上的投影. 為了目前的方便, 我們假設輸入的資料必定是個 $\mathbb{R}^N$ 中的向量, 而 $N = 2^J$.

以後我們必然會面臨 $\log_2 N \notin \mathbb{Z}$ 的情形. 一個最簡單的處理方式是, 將此向量補零, 使其長度變成 2^J. 這麼作的缺點是, 如逼近理論所述, 在銜接零的地方很可能有個不連續點, 導至每個 W_j 裡有個很大的投影係數. 另一個作法是, 做鏡射補足 2^J. 這樣作, 至少結果是個連續函數. 但是, 和補零一樣, 我們必須增加計算量. 例如原輸入向量的長度如果是 1025, 我們要補 1023 個數才有 2048$= 2^{11}$. 幾乎增加了一倍的輸入量. 這時候, 比較合理的作法是把輸入向量拆成一筆長度 1024 的資料和一筆長度 1 的資料. 這種想法的最複雜情形是 $N = 2^J - 1$ 時, 我們將要把輸入向量分解成 2^{J-1}, 2^{J-2}, $\ldots$ 1 等 J 個段落. 這時候, 補足次方的作法又顯得比較有效率了. 所以, 在程式設計上, 必須作個正確的判斷. 但是, 總而言之, 我們只須要知道當 $N = 2^J$ 時的作法, 其他的是可以解決的細節.

由於

$$\begin{aligned} V_j &= \operatorname{span}\{\phi_{jk}(x) \mid 0 \le k \le 2^j - 1\} \\ &= \operatorname{span}\{\phi_{j-1,k}(x), \psi_{j-1,k} \mid 0 \le k \le 2^{j-1} - 1\}. \end{aligned}$$

若 $\mathbf{v} \in V_j$, 存在 v_k, u_k 及 w_k 使得

$$\begin{aligned} \mathbf{v} &= \sum_{k=1}^{N} v_k \phi_{j,k-1}(x) \\ &= \sum_{k=1}^{N/2} u_k \phi_{j-1,k-1}(x) + \sum_{k=1}^{N/2} w_k \psi_{j-1,k-1}(x). \end{aligned}$$

當已知 v_k, 求 u_k 和 w_k 的計算法非常簡單:

$$\begin{aligned} u_k &= \frac{1}{\sqrt{2}}(v_{2k-1} + v_{2k}) \\ w_k &= \frac{1}{\sqrt{2}}(v_{2k-1} - v_{2k}) \end{aligned} \quad , \quad 1 \le k \le N/2.$$

當這個步驟結束了, 可以把 $\mathbf{u} = (u_k)$ 存在 $\mathbf{v}$ 的前半段, 把 $\mathbf{w} = (w_k)$ 存在 $\mathbf{v}$ 的後半段. 因為 $\sum w_k \phi_{j-1,k-1}(x) \in W_{j-1}$, 我們不再需要分解它, 所以 $\mathbf{v}$ 的後半段就不須再用. 如果 V_{j-1} 的解析度還不夠低, 就把 N 換成 $N/2$, $\mathbf{v}$ 換成 $\mathbf{v}$ 的的前半段, 再做一次分解步驟; 依此類推.

寫成矩陣形式, 令

$$H = \begin{pmatrix} 1 & 1 & & & & \\ & & 1 & 1 & & \\ & & & & \ddots & \\ & & & & 1 & 1 \end{pmatrix}_{N/2 \times N}, \quad G = \begin{pmatrix} 1 & -1 & & & & \\ & & 1 & -1 & & \\ & & & & \ddots & \\ & & & & 1 & -1 \end{pmatrix}_{N/2 \times N}.$$

則

$$\mathbf{u} = \frac{1}{\sqrt{2}} H\mathbf{v}, \quad \mathbf{w} = \frac{1}{\sqrt{2}} G\mathbf{v}.$$

我們稱 H 是一個低頻濾波器 (low-pass filter), G 是一個高頻濾波器 (high-pass filter). 所以, 一個 $V_j \to V_{j-1} \oplus W_{j-1}$ 的分解步驟, 相當於一個矩陣與向量乘法

$$\frac{1}{\sqrt{2}} \begin{pmatrix} H \\ G \end{pmatrix} \mathbf{v} = \begin{pmatrix} \mathbf{u} \\ \mathbf{w} \end{pmatrix}.$$

幾個計算方法的細節. 第一, 根本不必形成 H 與 G, 它們只提供理論敘述上的方便. 第二, 目前暫時不要理會 $1/\sqrt{2}$. 第三, 可以想像把 H 與 G 的列向量交錯放置:

$$L = \begin{pmatrix} 1 & 1 & & & & & & \\ 1 & -1 & & & & & & \\ & & 1 & 1 & & & & \\ & & 1 & -1 & & & & \\ & & & & \ddots & & & \\ & & & & & 1 & 1 & \\ & & & & & 1 & -1 \end{pmatrix}_{N \times N}$$

則

$$\frac{1}{\sqrt{2}} L\mathbf{v} = \begin{pmatrix} u_1 \\ w_1 \\ u_2 \\ w_2 \\ \vdots \\ u_{N/2} \\ w_{N/2} \end{pmatrix}.$$

這種算法, 只須要一個暫存空間就可以完成這個乘法, 而結果就直接覆寫在 $\mathbf{v}$ 的儲存空間上. 然後, 再重排足標, 把 $\mathbf{v}$ 的奇數項排在前半部, 偶數項排在後半部. 但是, 這種方式

必須用到疊代迴路 (for-loop). 用編譯語言 (C, FORTRAN) 寫來不成問題, 但是若用像 Matlab 的巨集指令來寫, 就可能會變慢了. 若用 Matlab 而空間不是問題, 最簡單 (可能也是最快) 的方法就是:

```
u = v(1:2:N) + v(2:2:N);
w = v(1:2:N) - v(2:2:N);
v(1:N/2) = u;
v(N/2+1:N) = w;
```

反過來, 用自格等式來作合成的工作. 若已知 $\mathbf{u}$ 和 $\mathbf{w}$, 要合成 $\mathbf{v}$; 也就是, 作 $V_{j-1} \oplus W_{j-1} \to V_j$. 這時有

$$\sum_{k=1}^{N/2} u_k \phi_{j-1,k-1}(x) + \sum_{k=1}^{N/2} w_k \psi_{j-1,k-1}(x).$$

因為

$$\phi(x) = \phi(2x) + \phi(2x-1), \quad \psi(x) = \phi(2x) - \phi(2x-1).$$

所以

$$\phi_{j-1,k} = \frac{1}{\sqrt{2}}(\phi_{j,2k} + \phi_{j,2k+1}), \quad \psi_{j-1,k} = \frac{1}{\sqrt{2}}(\phi_{j,2k} - \phi_{j,2k+1}).$$

我們得到

$$v_k = \begin{cases} \frac{1}{\sqrt{2}}(u_k + w_k) & \text{if } k \text{ is odd,} \\ \frac{1}{\sqrt{2}}(u_k - w_k) & \text{if } k \text{ is even.} \end{cases}$$

寫成矩陣形式, 就是

$$\mathbf{v} = \frac{1}{\sqrt{2}} \begin{pmatrix} u_1 \\ u_1 \\ u_2 \\ u_2 \\ \vdots \\ u_{N/2} \\ u_{N/2} \end{pmatrix} + \frac{1}{\sqrt{2}} \begin{pmatrix} w_1 \\ -w_1 \\ w_2 \\ -w_2 \\ \vdots \\ w_{N/2} \\ -w_{N/2} \end{pmatrix} = \frac{1}{\sqrt{2}} G^T \mathbf{u} + \frac{1}{\sqrt{2}} H^T \mathbf{w}.$$

換句話說,

$$(\frac{1}{\sqrt{2}}G)^T(\frac{1}{\sqrt{2}}G) + (\frac{1}{\sqrt{2}}H)^T(\frac{1}{\sqrt{2}}H) = I_{N\times N}.$$

用 $(1,1)$ 和 $(1,-1)$ 這兩個向量作成的一對高低頻濾波器 H 和 G, 既不產生多餘的資料 (no redundency): 濾前和濾後的資料量相等; 又不損失任何資料 (perfect reconstruction). 其實高頻的濾波係數 $(1,-1)$ 是由低頻的算出來的, 所以只有一組係數有待決定; 也就是, 一組特殊的自格等式的係數 c_k. 這樣的一對濾波器, 稱作 QMF (Quadrature Mirror Filter).

若是用 L 的格式, 則一個合成的步驟就寫成

$$\frac{1}{\sqrt{2}}L^T \begin{pmatrix} u_1 \\ w_1 \\ u_2 \\ w_2 \\ \vdots \\ u_{N/2} \\ w_{N/2} \end{pmatrix}.$$

也就是,

$$(\frac{1}{\sqrt{2}}L)^T(\frac{1}{\sqrt{2}}L) = I_{N\times N}.$$

但是, 又可以寫成

$$(\frac{1}{2}L)^T L = L^T(\frac{1}{2}L) = I.$$

哪一種比較好呢? 我們選擇在分解的時候除 2. 原因是, 這樣可以保持低頻數值的相對大小和原資料一致. 比如, 當 $\mathbf{v} = (1,1,\ldots,1,1)^T$, 則分解後 $\mathbf{u} = (1,\ldots,1)^T$, 而 $\mathbf{w} = (0,\ldots,0)^T$. 注意, 當然不要把每個資料除 2, 這會花掉 N 個計算量. 應該先把分解係數除 2, 這樣只要除四次. 除 2, 其實只是二進位數的一個平移.