設若 c_k 是一組自格係數, 我們已知凌波分解的步驟相當於兩個矩陣的作用. 一個稱作低頻濾波器 (low pass filter), 寫作 $\frac{1}{2}H$; 另一個稱作高頻濾波器 (high pass filter), 寫作 $\frac{1}{2}G$. 其行列的長度由輸入訊號的長度而定, 但通常要求列數 (row) 是行數 (column) 的一半. 也就是, 將輸入訊號寫成行向量 (假設行數是個偶數), 被 H 和 G 乘後分別得到長度只有一半的向量. H 和 G 的前幾列如下:

$$H = \begin{pmatrix} c_0 & c_1 & c_2 & c_3 & \cdots \\ & & c_0 & c_1 & \cdots \\ \vdots & & & & \end{pmatrix}, \quad G = \begin{pmatrix} c_{2p-1} & -c_{2p-2} & c_{2p-3} & -c_{2p-4} & \cdots \\ & & c_{2p-1} & -c_{2p-2} & \cdots \\ \vdots & & & & \end{pmatrix}.$$

很明顯的, 這兩個矩陣的最後幾列並不能按照這個 "每列向右邊移兩項" 的方式造成. 原因是某些非零係數會落到矩陣的 "邊界" 的外面. 我們有幾種處理的方法. 最單純的處理方式就是, 不理會那些落到矩陣 "外面" 的係數. 這樣作所引起的麻煩是

$$\frac{1}{2}(H^T H + G^T G) = \begin{pmatrix} B & \\ & I \end{pmatrix} \neq I.$$

也就是說, 合成的步驟就不再是單純的轉置矩陣. 其中 B 會是一個維度 $2p-2$ 的方陣. 矯正的方法就是合成的步驟還要再乘上 B^{-1} (如果它存在).

另一個處理方式可以使得合成的步驟就是單純的轉置矩陣. 那叫作 "週期化." 當某一列的非零係數要落到 "邊界" 的外面時, 我們把它 "折回來" 放到此列向量的前面. 就好像這輸入向量的長度是無限長, 而每 n 個元素就週期性地重複. 亦即,

$$v_{n+1} = v_1, \quad v_{n+2} = v_2, \quad \cdots.$$

所以, H 的最後幾列就會像是

$$\begin{pmatrix} c_{2p-2} & c_{2p-1} & & & & c_0 & c_1 & \cdots & c_{2p-4} & c_{2p-3} \\ & & & \cdots & & & & & & \\ c_4 & \cdots & c_{2p-1} & & & c_0 & c_1 & & c_2 & c_3 \\ c_2 & c_3 & \cdots & c_{2p-1} & & & & & c_0 & c_1 \end{pmatrix}$$

G 的情形亦同. 這麼一來,

$$\frac{1}{2}(H^T H + G^T G) = I.$$

也就是說, 如果

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

($\mathbf{u}$ 和 $\mathbf{w}$ 的長度各是 $\mathbf{v}$ 的一半), 則

$$H^T\mathbf{u} + G^T\mathbf{w} = \mathbf{v}.$$

如果 **v** 的長度是個偶數, 則 **u** 和 **w** 恰可以存在 **v** 的位置上. 如此, 我們可以結省計憶空間. 但是, 當 **u** 算完了, 我們還不能將它寫到 **v** 的位置上. 因為計算 **w** 時還須要 **v**. 但是, 如果我們算完了 u_1, 把它放在一個暫存空間, 立刻就接著算 w_1, 這時我們將不再須要 v_1 和 v_2. 這時候, 可以把 u_1 存入 v_1 的位置, w_1 存入 v_2 的位置. 依此類推, u_i 可以存入 v_{2i-1} 的位置, w_i 可以存入 v_{2i} 的位置.

對應這種算法, 相當於把 H 和 G 的列向量交替穿插, 造成一個新矩陣 (偶數長度的方陣). 它的前面幾列是

$$L = \begin{pmatrix} c_0 & c_1 & c_2 & c_3 & \cdots \\ c_{2p-1} & -c_{2p-2} & c_{2p-3} & -c_{2p-4} & \cdots \\ & & c_0 & c_1 & \cdots \\ & & c_{2p-1} & -c_{2p-2} & \cdots \\ \vdots & & & & \end{pmatrix}.$$

這種新的資料結構方式, 使得我們只須要一個額外的暫存空間.

作合成時就逆其道而行. 先把 **u** 和 **w** 交替穿插排列 (這個步驟的效率極為重要), 再用 L^T 來乘. 仍然只須要一個暫存空間, 用來存奇數列與輸入向量的內積. 輸入向量的初始資料是

$$\begin{pmatrix} u_1 \\ w_1 \\ u_2 \\ w_2 \\ \vdots \\ u_{n/2} \\ w_{n/2} \end{pmatrix}.$$

作完第一列和第二列的運算之後, 將不再須要 u_1 和 w_1, 它們的值就可以是 v_1 和 v_2. 依此類推.

但是, 真正寫程式的時候, 當然不須要在記憶體中生成矩陣 L. 我們只需要知道 L 和 L^T 的兩個列向量就夠了. 現在, 我們使用 `Matlab` 作為工具, 發展一個一維的離散凌波轉換 (discrete wavelet transform) 程式.

這是一個 `Matlab` 的 `function`. 假設輸入向量屬於某一個 V_J. 為了與數學符號作比較方便的對應, 我們假設輸入向量 x 的長度是 2^J. 我們要把這個訊號分解到

$$V_{j_0} \oplus W_{j_0} \oplus W_{j_0+1} \oplus \cdots \oplus W_{J-1}.$$

假設 x 在每個子空間 W_j 中的元素個數是 2^j, 則分解後的資料量仍是 2^J. 我們仍用 x 的儲存空間來存這分解後的訊號. 所以, 如果使用者還要保留原訊號, 他必須在把 x 送入這個 `function` 之前複製一份. 在程式中我們用 `Jend-1` 來代表 j_0. 也就是說, 我們對 `Jend` $\le j \le J$ 執行

$$V_j = V_{j-1} \oplus W_{j-1}$$

的步驟. j_0 由使用者輸入的參數 level 來控制, 但是如果這個 level 太小以至於沒有分解的意義, 則此程式會自動取一個有意義的最小值.

首先, 我們宣告這個 function.

```
function x = dwt(p,x,case,level)
```

其中 p 是指 Daubechies 的凌波函數的階數, 有意義的範圍是 $1 \le p \le 14$. case 是 1 或 -1; 1 代表分解, -1 代表合成.

若輸入 x 是個向量, 我們要它是個列向量 (column vector). 這樣, 以後的運算符號才會一致. 一般的情形, 若輸入是個圖形, 亦即矩陣資料, 則我們要它的行數 (row dimension) 不小於列數 (column dimension). 否則我們先將 x 作轉置.

```
[ROW,COL] = size(x);
if (ROW < COL)
    x = x';
end
```

接下來我們準備一些基本的參數或係數. 當 p 給定了, 我們從 getdaubc 得到自格係數 c_k, 存在向量 coef 中. 其中 $\texttt{coef}(i) = c_{i-1}$, $1 \le i \le 2p$. 我們已經將這些 c_k 存進一個檔案 daub_coef.mat. 它們是以雙精度浮點數的型式儲存, 這些值是用四倍精度在 VAX 6510 上計算的, 以 $\sum c_k - 2$ 作為誤差估計的標的, 這些值的誤差都在 10^{-24} 以下. 但是被存入雙精度型式時, 難免提高了誤差. getdaubc 是一個很簡單的程式.

```
function c = getdoubc(p)
    load daub_coef
    p = floor(p);
    if ((0<p) & (p<15))
        c = C(1:2*p, p);
    else
        disp('getdaubc: Not implemented.');
        c = zeros(2*p, 1);
    end
```

我們令 n 是 x 的行數, m 是 x 的列數 (一維向量的情形 $m = 1$). 假設 n 是 2 的整數次方. 一般而言, 取

$$J = \lceil \log_2 n \rceil.$$

然後假設輸入向量的長度是 $N = 2^J$. $\phi(x)$ 或 $\psi(x)$ 的函蓋是 $[0, R] := [0, 2p-1]$, 而非零自格係數的個數是 $2p = R + 1 = \texttt{NC}$.

```
coef = getdaubc(p);
```

```
[ndim, mdim] = size(x);
J = ceil ( log2(ndim) );
N = 2^J;
R = 2*p - 1;
NC = R+1;
```

現在讓我們先認定 $N = n$, $m = 1$. 我們的分解步驟將由 $j = J$ 依次遞減作到 $j =$ Jend. 根據 "週期化" 的想法, 把矩陣的最後幾列週期化其實等於將輸入向量的最前面 $2p-2$ 項抄到原向量的後面, 然後用完全一樣的矩陣乘法來作分解. 由此可見, 當 x 已經被分解到 V_j 而 $2^j < 2p-2$ 之後, 就不容易作這個週期化的動作. 其實 $2p-2$ 已經夠小了, 作到這裡停下來並沒有損失什麼. 注意當 $p = 1$ 時 $2p-2 = 0$, 因為 $p = 1$ 時我們並不須要週期化; 沒有係數會 "跨過" 矩陣的邊界. 為了方便起見, 我們就取最小可能的

$$\texttt{Jend} = \lceil \log_2 2p \rceil, \qquad j_0 = \texttt{Jend} - 1.$$

而使用者要求的最低層解析度是 $j_0 =$ level. 我們儘可能滿足這個要求.

```
Jend = ceil( log2(2*p) );
Jend = max(Jend, level+1);
```

如果要作分解, 我們將低頻濾波係數, 也就是 L 中奇數列的非零項, 存在長度 $2p$ 的行向量 DL 中; 而高頻濾波係數, 也就是 L 中偶數列的非零項, 存在長度 $2p$ 的行向量 DH 中. 為了達到 "低頻" 相當於 "取加權平均" 的想法, 我們要將 DL 和 DH 中的係數除 2.

```
if (case == 1)
    for i = 1:NC,
        DL(i) = coef(i);
        DH(i) = (-1)^(i-1) * coef(NC-i+1);
    end
    DL = DL/2;
    DH = DH/2;
    disp (['Decomposing from level ' int2str(J) ' to level ' int2str(Jend-1)])
```

接著我們就開始一層一層地作 $V_j = V_{j-1} \oplus W_{j-1}$, $j = J, J-1, \ldots,$ Jend. 我們令 x 向量中的 $x_1, \ldots, x_{2^j}$ 存的是在 V_j 中的投影係數, $x_{2^j+1}, \ldots, x_{2^{j+1}}$ 存的是在 W_j 中的投影係數, $x_{2^{j+1}+1}, \ldots, x_{2^{j+2}}$ 存的是在 W_{j+1} 中的投影係數, 依此類推到 W_{J-1}. 所以, 對每一個 j, 我們要取出 x 的前面 2^j 行作進一步的分解, 而保留其餘各行. 這個部份由 split 子程式作成. 然後就執行子程式 pwd, 它將 x 作適當的週期化處理, 然後作矩陣與向量的乘積. 執行了 pwd 之後, 奇數項是 V_{j-1} 中的低解析投影, 而偶數項是 W_{j-1} 的部份. 我們用 postd 來重整 x 的資料結構.

```
    for j = J:-1:Jend,
        [x xold] = split(2^j, x);
        x = pwd(p,x,DL,DH);
        x = postd(x, xold);
    end;
```

其中 split 子程式很單純地將一個矩陣分成上下兩部份, 列的部份不動.

```
function [xa, xb] = split(n,x)
    [N, M] = size(x);
    if (n > N)
        xa = x;
        xb = [];
        disp('WARNING. split: cut point larger than vector lenghth.')
    else
        xa = x(1:n,:);
        xb = x(n+1:N,:);
    end;
```

子程式 pwd 先作分解前的週期化, 然後叫 wf 作大量的實際計算工作, 作完之後再截去尾巴上為了週期化而加長的幾項.

```
function x = pwd(p,x,L,H)
    [n, m] = size(x);
    x(n+1:n+2*p-2, :) = x(1:2*p-2, :);
    x = wf(x,n,L,H);
    x = x(1:n, :);
```

真正的大量計算工作是由 wf 完成的. 它接受 x 為輸入向量, 以 L 和 H 為低頻與高頻濾波係數 (行向量格式), 作 n 次濾波. 注意, 若 x 是一個矩陣, 則濾波係數乘上了 x 的每一個行向量.

```
function x = wf(x,n,L,H)
    R = length(L) - 1;
    for i = 1:2:n,
        t1 = L*x(i:i+R, :);
        t2 = H*x(i:i+R, :);
        x(i, :) = t1;
        x(i+1, :) = t2;
    end;
```

當 case $= -1$, 也就是, 作合成時, 我們將 L^T 的奇數列存在 RL, 偶數列存在 RH. 不須除 2, 否則就不會是 L 逆矩陣了.

```
elseif (case == −1)
    i = 1;
    for j = 2∗p−2:−2:0,
        RL(i) = coef(j+1);
        RH(i) = coef(j+2);
        RL(i+1) = coef(R−j+1);
        RH(i+1) = (−1) ∗ coef(R−j);
        i = i+2;
    end;
    disp (['Reconstruct from level ' int2str(Jend−1) ' to level ' int2str(J)])
```

接著, 我們還是要先從 x 中取得其在 $V_{j-1} \oplus W_{j-1}$ 的部份, 這還是用 split 來作. 然後我們要把 V_{j-1} 和 W_{j-1} 中的元素交替穿插重排, 這由子程式 interlace 來完成. 最後, 我們把這一部份的 x 送入 pwc, 再將結果與 x 的另一部份聯結在一起.

```
    for j = Jend:J,
        [x xold] = split(2^j, x);
        x = interlace(x);
        x = [pwc(p,x,RL,RH)
             xold];
    end;
```

子程式 interlace 先檢查輸入向量是否有偶數行. 如果是, 則把此向量的前面一半放到奇數項, 後面一半放到偶數項. 否則給一個錯誤訊息.

```
function w = interlace(v)
    [n, m] = size(v);
    ifodd = 'n - 2*floor(n/2)';
    if eval(ifodd) == 0
        w(1:2:n−1, :) = v(1:n/2, :);
        w(2:2:n, :) = v(n/2+1:n, :);
    else
        disp ('ERROR: interlace. Cannot interlace odd number of data.')
        w = v;
    end
```

而 pwc 和 pwd 所作的工作很像, 就是把週期化的結構準備好, 再把實際的計算工作交給 wf.

注意, 只要資料給得對, 基本的計算步驟都是由 wf 完成的. 在 wf 之後, 要把最後的幾個因為週期化加長的項移到前面; 它們才是正確的值.

```
function x = pwc(p,x,L,H)
    [n, m] = size(x);
    x(n+1:n+2*p-2, :) = x(1:2*p-2, :);
    x = wf(x,n,L,H);
    z=x(n-2*p+3:n, :);
    x = [z
         x(1:n-2*p+2, :)];
```

主程式的最後, 我們提醒使用者, case 只有 1 或 -1, 所以這個程式不必因此而犯錯. 如果一開始的時候我們曾轉置過 x, 現在該把它轉回來.

```
else
   disp ('ERROR: case must be either 1 or -1.')
end
if (ROW < COL)
    x = x';
end
```

我們用 Matlab 中的 flops 功能來實際的估計計算複雜度. Matlab 把一個實浮點數的相加或相乘各算一個 flop. 當輸入向量的長度分別是

$$2048, \quad 1024, \quad 512, \quad 256, \quad 128,$$

而 $p = 4$ 時, Matlab 回答的 flops 分別是

$$71703, \quad 35852, \quad 17921, \quad 8950, \quad 4459.$$

所以這程式的計算複雜度大約是

$$8.75pN.$$

因為濾波係數的長度是 $2p$, 依照 Matlab 對 flop 的定義, 理論上的複雜度是

$$8pN.$$

很明顯的, 計算量全部出於 wf 這個子程式. 如果去掉這個部份, 則 Matlab 回答的 flops 分別是

$$145, \quad 136, \quad 127, \quad 118, \quad 109.$$

所以 wf 是計算時間的瓶頸. 這也是我們要把它獨立出來的原因.