Contents

1. The Continuous Wavelet Transform

- Historically, the continuous wavelet transform came first.
- it is completely different from the discrete wavelet transform
- it is popular among physicists, whereas the DWT is more common in numerical analysis, signal- and image-processing.

1.1. The continuous wavelet transform (CWT)

Recall the CWT

$$\mathcal{W}_\psi f = S(a,b) = \frac{1}{\sqrt{a}} \int_{-\infty}^{\infty} f(t) \overline{\psi\left(\frac{t-b}{a}\right)} dt$$

This is an *overcomplete* function representation. (from one dimension, we transform into a 2D space!!). We want to be able to reconstruct $f(t)$ from this representation. This is possible if the *admissibility condition* is satisfied:

$$C_\psi = \int_{-\infty}^{\infty} \Psi(\omega) \frac{d\omega}{|\omega|} < \infty$$

A necessary (and normally also sufficient) condition is: $\Psi(0) = 0$ which means:

$$\int_{-\infty}^{\infty} \psi(t)dt = 0$$

All functions with zero integral are *wavelets*
There are:

- no multi-resolution analysis
- no father/scaling function

All wavelets in the DWT-theory remain wavelets here, but these functions are mostly too irregular for proper use in a CWT-analysis.

The CWT is often used to characterize singularities in functions, and from this it can distinguish between noise and signal (Jaffard, Mallat, Hwang, Zhong)

It can be used to study fractals, etc.

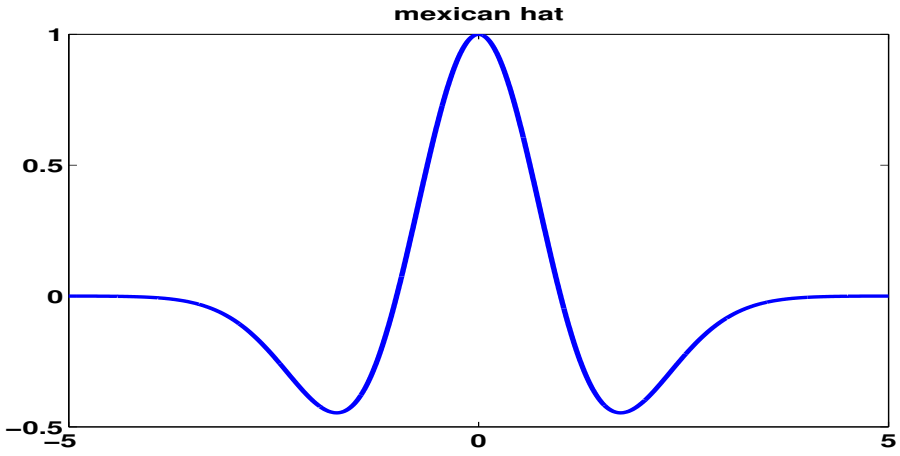
Two functions are popular for CWT-analysis. These functions do *not fit into a MRA*.

1. Mexican hat wavelet

$$\psi(x) = (1 - x^2)\mathrm{e}^{-x^2/2}$$

This is the second derivative of a Gaussian.

$$\Psi(\omega) = \omega^2 \mathrm{e}^{-\omega^2}$$



2. The Morlet wavelet

$$\psi(x) = \mathrm{e}^{i\omega_0 x}\mathrm{e}^{-x^2/2\sigma_0^2}$$

This function does not satisfy the admissibility condition exactly. It therefore has to be corrected a bit.

1.2. Discretisation of the CWT

Another difference between CWT and DWT:
DWT starts with a discrete set of data and considers a dyadic set of scales.
Discretisation comes first! (Hence Discrete WT!!)

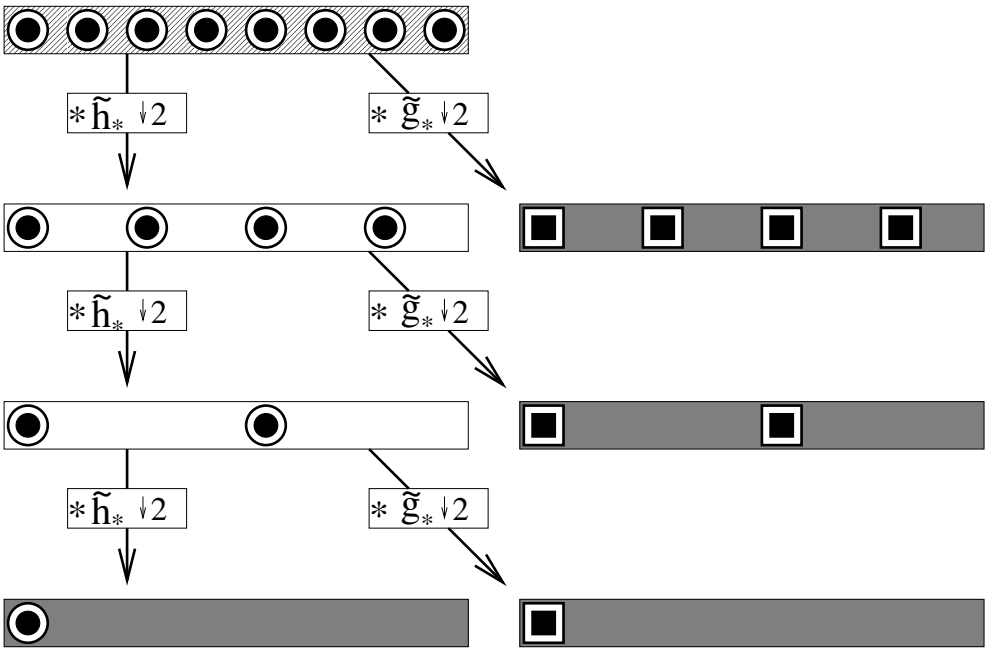
A CWT must be decretised to be implemented on a computer.
If $\psi(t)$ fits into a MRA, then a discretisation like in the discrete case is of course possible, but in general, the representation remains redundant.

Such overcomplete function expansions are called *frames*. Loosely spoken, a frame is a family of functions ψ_j, such that a function f can be reconstructed from

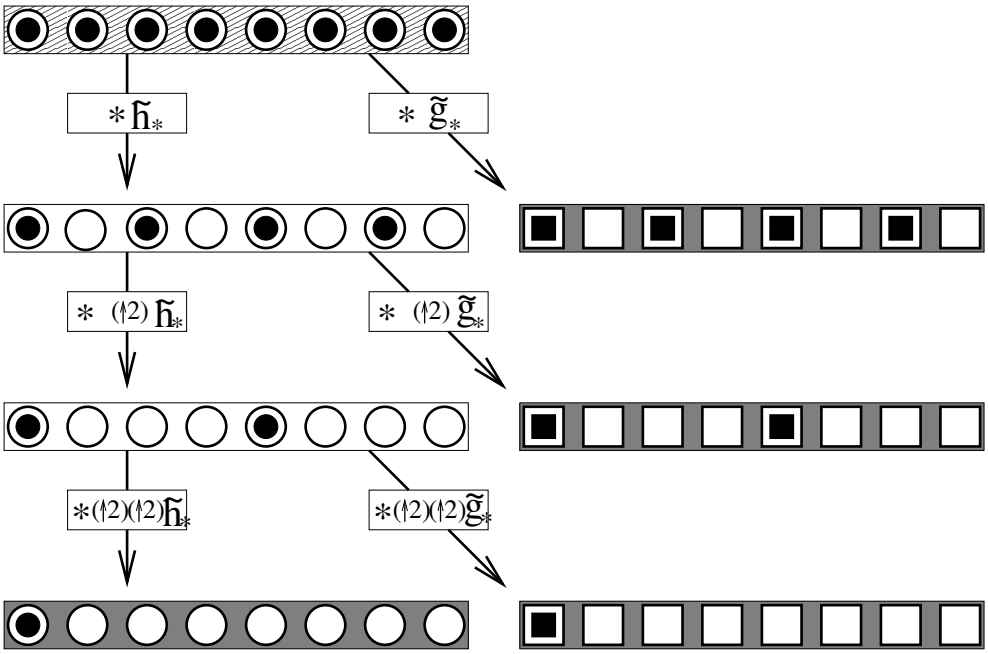
$$\langle f(t), \psi_j(t) \rangle$$

2. Stationary wavelet transform or redundant wavelet transform or a trous algorithm

Suppose we have a FWT:



Because of the subsampling step, we have less coefficients at coarse scales.
If we omit the subsampling step, we get a redundant wavelet transform. (RWT)



The subsampling is compensated by upsampling the filters.
$\rightarrow$ the transform is consistent with the FWT.

2.1. Reconstruction

In each step (i.e. scale) the number of data doubles. Hence at each step the input can be reconstructed from the output in 2 independent ways.

2.2. properties

1. the number of coefficients is equal at all scales. It equals the number of input samples.
2. the transform is translation-invariant ($\leftrightarrow$ FWT)
3. the transform is immediately extensible for non-dyadic inputs.
4. This is an overcomplete data representation.
   reconstruction is not unique: if the coefficients are manipulated, the result is probably not the RWT of any function. Averaging the results from different reconstructions has smoothing effect.
5. Computational complexity and storage is $\mathcal{O}(N \log N)$

2.3. RWT and frames

Suppose:

$$f(t) = \sum_{k \in \mathbb{Z}} s_{J,k} \varphi(2^J t - k)$$

then we know:

$$w_{j,k}^{\text{FWT}} = 2^{j/2} \int_{-\infty}^{\infty} f(t) \, \overline{\tilde{\psi}(2^j t - k)} \, dt$$

And similarly:

$$w_{j,k}^{\text{RWT}} = 2^{j/2} \int_{-\infty}^{\infty} f(t) \, \overline{\tilde{\psi}(2^j t - 2^{j-J} k)} \, dt$$

This can be interpreted as a dyadic discretisation of the CWT:

$$(\mathcal{W}_\psi f)(a, b) = \frac{1}{\sqrt{a}} \int_{-\infty}^{\infty} f(t) \, \overline{\tilde{\psi}\left(\frac{t-b}{a}\right)} \, dt$$

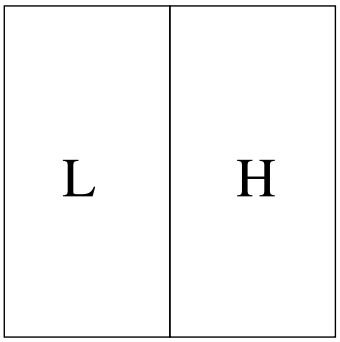
with:

$$a_j = 2^{-j} \text{ and } b_k = 2^{-J} k$$

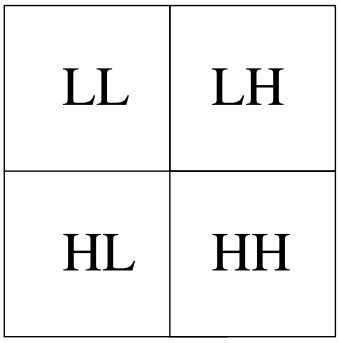
3. 2D Wavelet transform

1. the square wavelet transform

We first perform one step of the transform on all rows:
The left side of the matrix contains downsampled lowpass coefficients of each row, the right contains the highpass coefficients



Next, we apply one step to all columns:



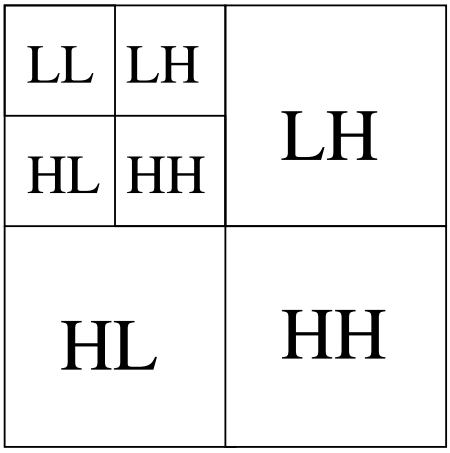
This results in four types of coefficients:

(a) coefficients that result from a convolution with $\boldsymbol{g}$ in both directions (HH) represent diagonal features of the image.

(b) coefficients that result from a convolution with $\boldsymbol{g}$ on the columns after a convolution with $\boldsymbol{h}$ on the rows (HL) correspond to horizontal structures.

(c) coefficients from highpass filtering on the rows, followed by lowpass filtering of the columns (LH) reflect vertical information.

(d) the coefficients from lowpass filtering in both directions are further processed in the next step.



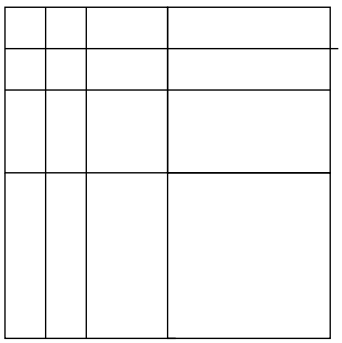
At each level, we have three *components* or *subbands*. If we start with:

$$f(t) = \sum_{k\in\mathbb{Z}} \sum_{l\in\mathbb{Z}} s_{J,k,l} \varphi_{J,k}(x) \varphi_{J,l}(y)$$

we end up with:

$$\begin{aligned} f(t) \quad = \quad & \sum_{k\in\mathbb{Z}} \sum_{l\in\mathbb{Z}} s_{L,k,l} \varphi_{L,k}(x) \varphi_{L,l}(y) \quad \in V_L \otimes V_L \\ + \quad & \sum_{j=L}^{J-1} \Bigg[\sum_{k\in\mathbb{Z}} \sum_{l\in\mathbb{Z}} w^{\mathrm{HH}}_{j,k,l} \psi_{j,k}(x) \psi_{j,l}(y) \qquad \bigoplus_{j=L}^{J-1} \quad [\quad W_j \otimes W_j \\ & \qquad + w^{\mathrm{LH}}_{j,k,l} \psi_{j,k}(x) \varphi_{j,l}(y) \qquad\qquad \oplus W_j \otimes V_j \\ & \qquad + w^{\mathrm{HL}}_{j,k,l} \varphi_{j,k}(x) \psi_{j,l}(y) \quad \Bigg] \qquad\qquad \oplus V_j \otimes W_j \quad] \end{aligned}$$

2. The rectangular wavelet transform
We could also transform in each step *all* rows and columns (and not only those that are in the LL-subband) This is the following scheme:



In this case, we can change order of operations: we could for instance, perform the complete transform on the columns, before transforming the rows.

If $\mathbf{w} = \mathbf{W}\mathbf{y}$ is the matrix representation of a 1D wavelet transform, then the rectangular transform, applied to an image $\mathbf{M}$ is:

$$\mathbf{W}\mathbf{M}\mathbf{W}^T$$

The basis corresponding to this decomposition contains functions that are tensor products of wavelets at *different* scales:

$$\psi_{j,k}(x)\psi_{i,l}(y)$$

Such functions do not appear in the basis of a square wavelet transform.

For image processing, we prefer the *square* transform:

- less computations
- more specific info on vertical, diagonal, horizontal features at *each scale*

Both rectangular and square wavelet transform are *separable* transforms.
example

$s_{L,k,l}$ and $255 - |w^c_{j,k,l}|$,
$c = \mathrm{HH}, \mathrm{HL}, \mathrm{LH}$

comparison of rectangular

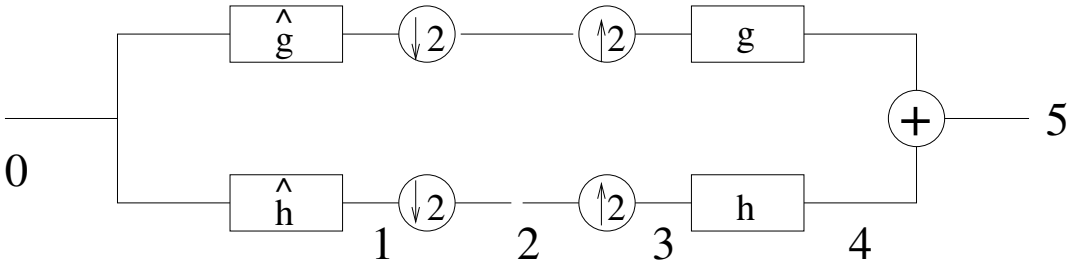
and square transform

4. Polyphase

If we want to describe the forward and inverse wavelet transform

$$
\begin{aligned}
s_{j,k} &= \sum_{l \in \mathbb{Z}} \bar{\bar{h}}_{l-2k} s_{j+1,l} \\
w_{j,k} &= \sum_{l \in \mathbb{Z}} \bar{\bar{g}}_{l-2k} s_{j+1,l} \\
s_{j+1,k} &= \sum_{l \in \mathbb{Z}} h_{k-2l} s_{j,l} + \sum_{l \in \mathbb{Z}} g_{k-2l} w_{j,l}
\end{aligned}
$$

as stationary filter-operations, then we need downsampling and upsampling operations.
In the classical scheme, downsampling comes after analysis filtering and upsampling comes before synthesis filtering.



If we implement the algorithm according to this scheme, we throw away half of the filtering work.
Therefore, we try to subsample on the input and output. Consider:

$$\begin{aligned} s^e_{j,l} &= s_{j,2l} \quad \text{or} \quad \boldsymbol{s}^e_j = (\downarrow 2)\boldsymbol{s}_j \\ s^o_{j,l} &= s_{j,2l+1} \quad \text{or} \quad \boldsymbol{s}^o_j = (\downarrow 2)\mathcal{S}\boldsymbol{s}_j \end{aligned}$$

$\mathcal{S} = \mathcal{D}^{-1}$ is the (forward) shift operator:

$$\boldsymbol{y} = \mathcal{S}\boldsymbol{x} \Leftrightarrow y_k = x_{k+1}$$

We can write:

$$\begin{aligned} s_{j,k} &= \sum_{l\in\mathbb{Z}} \bar{\tilde{h}}_{l-2k} s_{j+1,l} \\ &= \sum_{l\in\mathbb{Z}} \bar{\tilde{h}}_{2l-2k} s_{j+1,2l} + \sum_{l\in\mathbb{Z}} \bar{\tilde{h}}_{2l+1-2k} s_{j+1,2l+1} \\ &= \sum_{l\in\mathbb{Z}} \bar{\tilde{h}}^e_{l-k} s^e_{j+1,l} + \sum_{l\in\mathbb{Z}} \bar{\tilde{h}}^o_{l-k} s^o_{j+1,l} \end{aligned}$$

If we call $\hat{h}_k = \bar{\tilde{h}}_{-k}$ as before, we can write: $\bar{\tilde{h}}^e_{-m} = \hat{h}^e_m$.
BUT (!!!!) $\bar{\tilde{\boldsymbol{h}}}^o = \mathcal{D}\hat{\boldsymbol{h}}^o$ since

$$\bar{\tilde{h}}^o_{-m} = \bar{\tilde{h}}_{-2m+1} = \hat{h}_{2m-1} = \hat{h}_{2(m-1)+1} = \hat{h}^o_{m-1}$$

And so:

$$\sum_{l\in\mathbb{Z}} \bar{\tilde{h}}^o_{l-k} s^o_{j+1,l} = \sum_{l\in\mathbb{Z}} \hat{h}^o_{k-1-l} s^o_{j+1,l} = \left(\mathcal{D}\hat{\boldsymbol{h}}^o\right) * \boldsymbol{s}^o_{j+1}$$

$\mathcal{D}$ is a delay operator (the inverse shift).
$\mathcal{D}$ can be interchanged with convolution:

$$(\mathcal{D}\boldsymbol{x}) * \boldsymbol{y} = \mathcal{D}\left(\boldsymbol{x} * \boldsymbol{y}\right)$$

The equation

$$s_{j,k} = \sum_{l\in\mathbb{Z}} \bar{\tilde{h}}^e_{l-k} s^e_{j+1,l} + \sum_{l\in\mathbb{Z}} \bar{\tilde{h}}^o_{l-k} s^o_{j+1,l}$$

becomes:

$$\boldsymbol{s}_j = \hat{\boldsymbol{h}}^e * \boldsymbol{s}^e_{j+1} + \mathcal{D}\hat{\boldsymbol{h}}^o * \boldsymbol{s}^o_{j+1}$$

A similar argument leads to:

$$\boldsymbol{w}_j = \hat{\boldsymbol{g}}^e * \boldsymbol{s}^e_{j+1} + \mathcal{D}\hat{\boldsymbol{g}}^o * \boldsymbol{s}^o_{j+1}$$

We re-write the reconstruction:

$$\begin{aligned} s^e_{j+1,k} &= s_{j+1,2k} \\ &= \sum_{l\in\mathbb{Z}} h_{2k-2l} s_{j,l} + \sum_{l\in\mathbb{Z}} g_{2k-2l} w_{j,l} \\ &= \sum_{l\in\mathbb{Z}} h^e_{k-l} s_{j,l} + \sum_{l\in\mathbb{Z}} g^e_{k-l} w_{j,l} \end{aligned}$$

$$\begin{aligned} s^o_{j+1,k} &= s_{j+1,2k+1} \\ &= \sum_{l\in\mathbb{Z}} h_{2k+1-2l} s_{j,l} + \sum_{l\in\mathbb{Z}} g_{2k+1-2l} w_{j,l} \\ &= \sum_{l\in\mathbb{Z}} h^o_{k-l} s_{j,l} + \sum_{l\in\mathbb{Z}} g^o_{k-l} w_{j,l} \end{aligned}$$

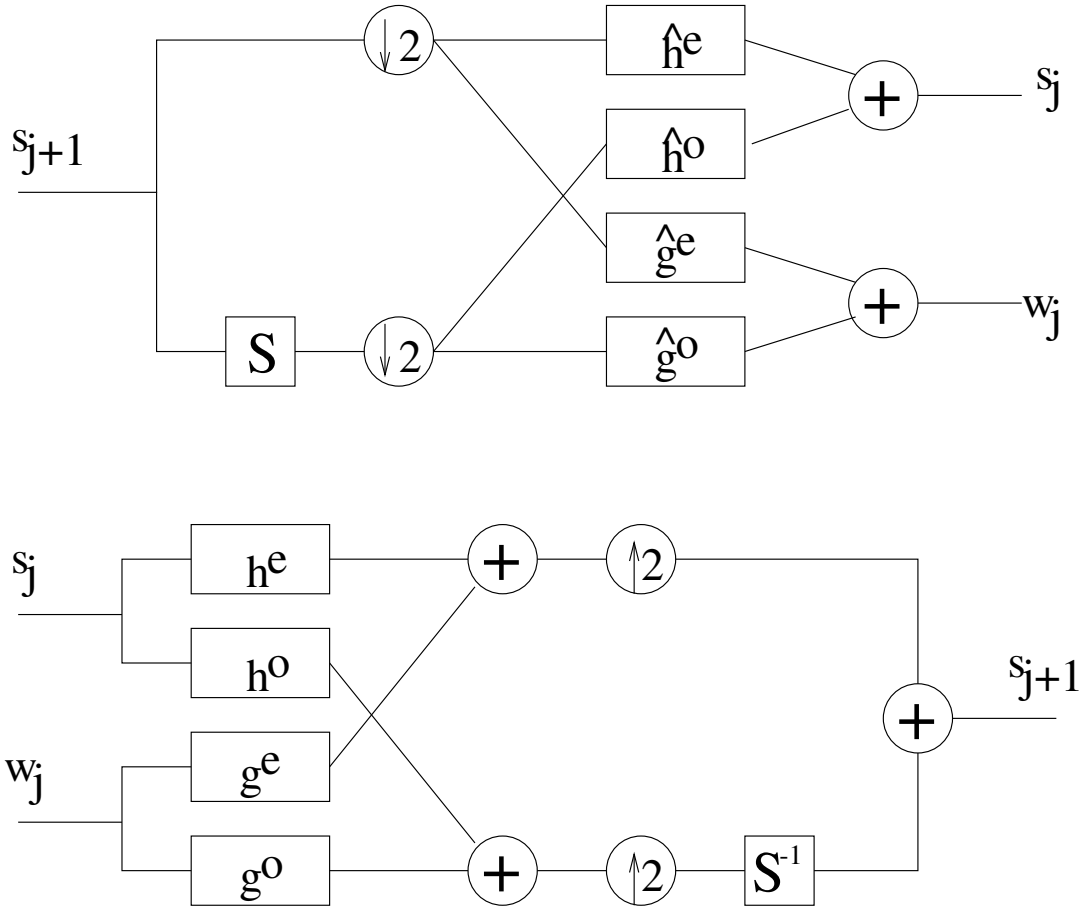
Or:

$$\boldsymbol{s}^e_{j+1} = \boldsymbol{h}^e * \boldsymbol{s}_j + \boldsymbol{g}^e * \boldsymbol{w}_j$$

and:

$$\boldsymbol{s}^o_{j+1} = \boldsymbol{h}^o * \boldsymbol{s}_j + \boldsymbol{g}^o * \boldsymbol{w}_j$$

This can be represented as:



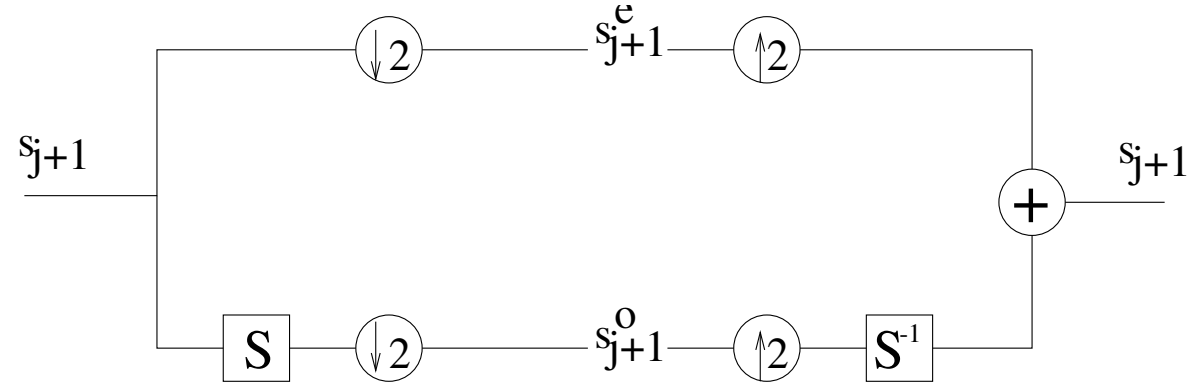
This is called polyphase: the filter bank is written as combinations of different phases of filters and signals. Implementing this scheme as such does not introduce useless computations.

4.1. The lazy wavelet transform

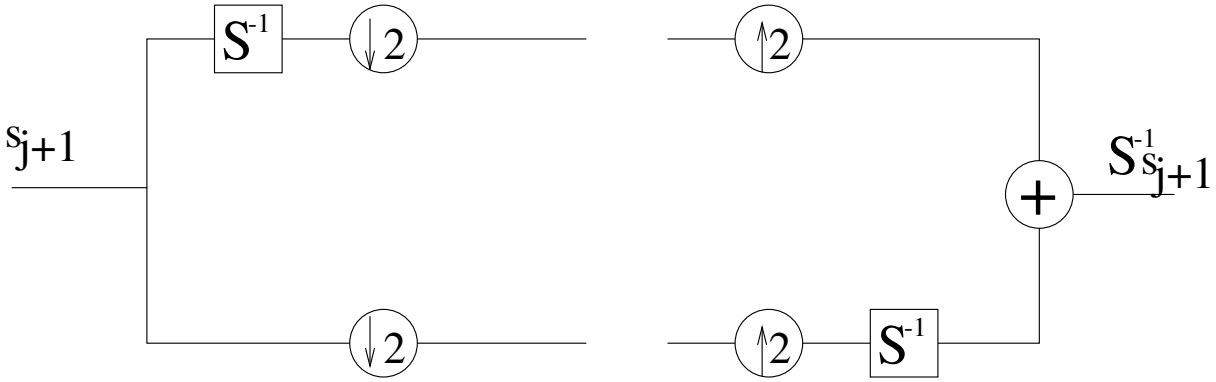
The lazy wavelet transform is the most simple filterbank: It has:

$$\begin{aligned}
\hat{h}^e_k &= \delta_k &&= \hat{g}^o_k \\
\hat{h}^o_k &= 0 &&= \hat{g}^e_k \\
h^e_k &= \delta_k &&= g^o_k \\
h^o_k &= 0 &&= g^e_k
\end{aligned}$$

It just splits the signal in even and odd:



Note 1: The shift-operator $\mathcal{S}$ is not causal, and therefore not practically realisable. Therefor, in practice we use the following scheme:



The output is a delay of the input. As a matter of fact, the same problem occurs for the classical filterbank representation of a wavelet transform. Since, for instance, $\tilde{g}_k = (-1)^k \bar{h}_{1-k}$, at least one of these filters is non causal (unless the filter length is less than 3, like in the Haar case)
From the mathematics point of view, we can ignore this problem.

Note 2: The lazy wavelet transform corresponds to

$$h_k = \delta_k$$

The cascade or iteration algorithm with this filter does not converge in L_2, i.e. there is no L_2 function for which:

$$\varphi(x) = \sqrt{2}\varphi(2x)$$

5. Polyphase in Z-domain

$$\begin{aligned} \mathcal{Z}(\mathcal{S}\boldsymbol{s}_{j+1}) &= z\mathcal{Z}(\boldsymbol{s}_{j+1}) = zS_{j+1}(z) \\ \mathcal{Z}(\mathcal{S}^{-1}\boldsymbol{s}_{j+1}) &= z^{-1}S_{j+1}(z) \end{aligned}$$

$$\begin{aligned} S_j(z) &= \hat{H}^e(z)S^e_{j+1}(z) + z^{-1}\hat{H}^o(z)S^o_{j+1}(z) \\ W_j(z) &= \hat{G}^e(z)S^e_{j+1}(z) + z^{-1}\hat{G}^o(z)S^o_{j+1}(z) \end{aligned}$$

We know: $\tilde{h}_k = \bar{\hat{h}}_{-k}$ $\tilde{g}_k = \bar{\hat{g}}_{-k}$ and so:

$$\hat{H}^e(z) = \sum_{k\in\mathbb{Z}} \hat{h}_{2k} z^{-k} = \sum_{k\in\mathbb{Z}} \bar{\hat{h}}_{-2k} z^{-k} = \sum_{l\in\mathbb{Z}} \bar{\hat{h}}_{2l} z^{l} = \tilde{H}^e_*(z)$$

$$\begin{aligned}\hat{H}^o(z) &= \sum_{k\in\mathbb{Z}} \hat{h}_{2k+1} z^{-k} = \sum_{k\in\mathbb{Z}} \bar{\hat{h}}_{-2k-1} z^{-k} = \sum_{l\in\mathbb{Z}} \bar{\hat{h}}_{2l-1} z^{l} \\ &= \sum_{m\in\mathbb{Z}} \bar{\hat{h}}_{2m+1} z^{m+1} = z\tilde{H}^o_*(z)\end{aligned}$$

We can write a matrix-equation:

$$\begin{bmatrix} S_j(z) \\ W_j(z) \end{bmatrix} = \begin{bmatrix} \tilde{H}^e_*(z) & \tilde{G}^e_*(z) \\ \tilde{H}^o_*(z) & \tilde{G}^o_*(z) \end{bmatrix}^T \cdot \begin{bmatrix} S^e_{j+1}(z) \\ S^o_{j+1}(z) \end{bmatrix}$$

And for the synthesis:

$$\begin{bmatrix} S^e_{j+1}(z) \\ S^o_{j+1}(z) \end{bmatrix} = \begin{bmatrix} H^e(z) & G^e(z) \\ H^o(z) & G^o(z) \end{bmatrix} \cdot \begin{bmatrix} S_j(z) \\ W_j(z) \end{bmatrix}$$

Call the polyphase matrix:

$$P(z) = \begin{bmatrix} H^e(z) & G^e(z) \\ H^o(z) & G^o(z) \end{bmatrix}$$

Its para-hermitian conjugate equals:

$$P_*(z) = \overline{P(1/\overline{z})}^T = \begin{bmatrix} H^e_*(z) & G^e_*(z) \\ H^o_*(z) & G^o_*(z) \end{bmatrix}^T$$

Thus we have PR iff $P(z)\tilde{P}_*(z) = I$

For orthogonal filterbanks, we have

$$\begin{aligned}\tilde{h}_k = h_k \quad \text{and} \quad \tilde{g}_k = g_k \\ \tilde{H}(z) = H(z) \quad \text{and} \quad \tilde{G}(z) = G(z)\end{aligned}$$

So: $\tilde{P}(z) = P(z)$ and PR is equivalent to a para-unitary polyphase matrix: $P(z)P_*(z) = I$.

6. Lifting

6.1. In place calculation of the Haar transform

Fast Haar transform:

$$\begin{cases} s_{j,k} = \frac{s_{j+1,2k+1}+s_{j+1,2k}}{2} \\ w_{j,k} = s_{j+1,2k+1} - s_{j+1,2k} \end{cases}$$

If we implement the algorithm as such, we need the values of $s_{j+1,2k+1}$ and $s_{j+1,2k}$ until the computation of $w_{j,k}$ is done. Therefore we need new memory

locations for $s_{j,k}$. This is not an in-place calculation: we cannot overwrite the existing locations of $s_{j+1,2k+1}$ and $s_{j+1,2k}$.

However:

$$s_{j+1,2k+1} = w_{j,k} + s_{j+1,2k}$$

and so:

$$\begin{aligned} s_{j,k} &= \frac{s_{j+1,2k+1} + s_{j+1,2k}}{2} \\ &= \frac{w_{j,k} + s_{j+1,2k} + s_{j+1,2k}}{2} \\ &= s_{j+1,2k} + \frac{w_{j,k}}{2} \end{aligned}$$

After the computation of $w_{j,k}$, we don't need $s_{j+1,2k+1}$ anymore, so we could overwrite $s_{j+1,2k+1}$ with the value of $w_{j,k}$ and $s_{j+1,2k}$ with the value of $s_{j,k}$. We drop the level index j:

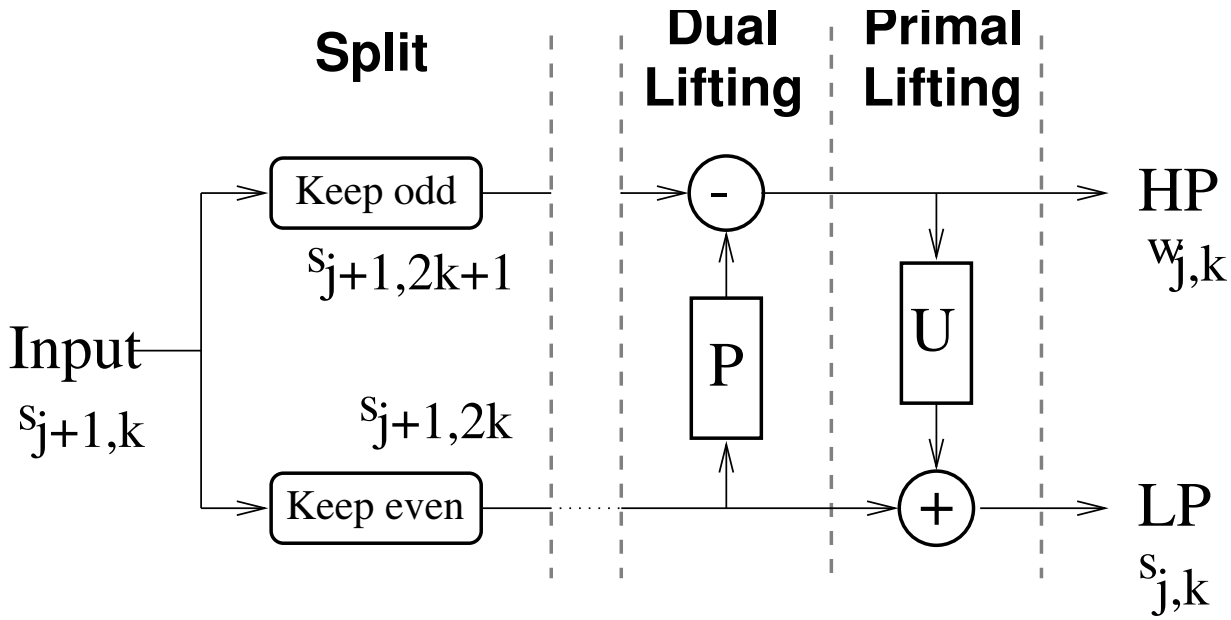
$$\begin{aligned} s_{2k+1} &\leftarrow s_{2k+1} - s_{2k} \\ s_{2k} &\leftarrow s_{2k} + s_{2k+1}/2 \end{aligned}$$

Recovering from this is immediate:

$$\begin{aligned} s_{2k} &\leftarrow s_{2k} - s_{2k+1}/2 \\ s_{2k+1} &\leftarrow s_{2k+1} + s_{2k} \end{aligned}$$

6.2. Split, predict, update

One step of this in-place Haar transform fits into the following scheme:



This is the *lifting scheme*. It consists of a subsequence of three fundamental operations:

1. Splitting
   splits the signal in an even and odd part.
   also appears in the polyphase-representation
   in fact, it is the forward lazy wavelet transform.

After splitting, we have an alternating subsequence of two *lifting* operations:

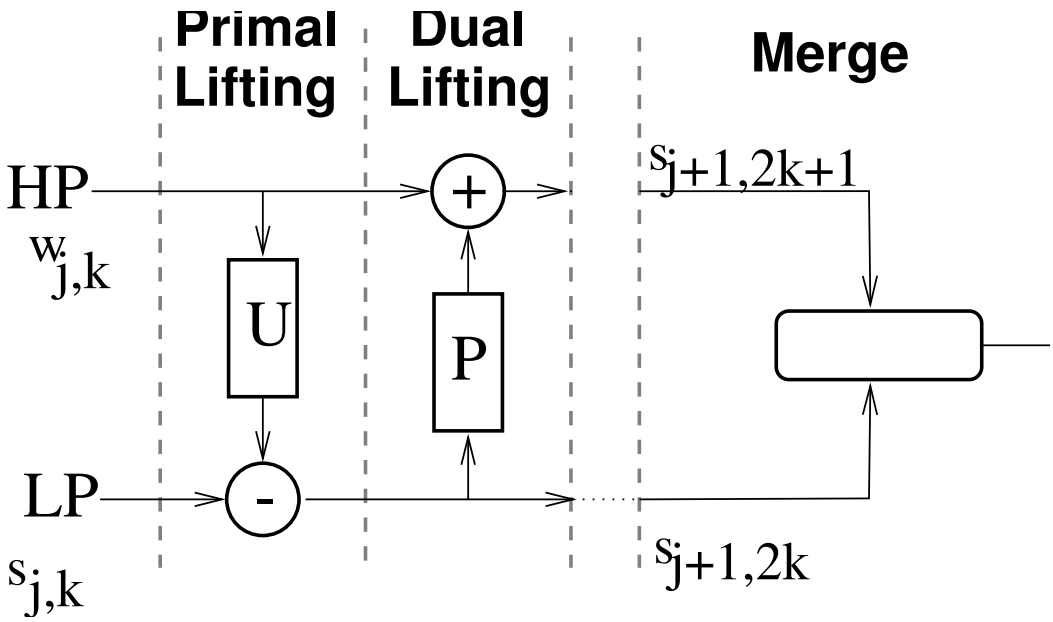
2. Dual lifting (prediction P)
   substracts a filtered version of the evens from the odd. For the Haar transform, the operator P is simply the identity: $P(s_{j+1,2l}) = s_{j+1,2l}$. One can iterpret this step as a *prediction* of the odd set by the evens: the prediction error is the essential/detail information: these are highpass coefficients

3. Primal lifting (update U)
   It is necessary to update the lowpass coefficients before using them in the next step. For the Haar transform, this update is simply:
   $U(w_{j,k}) = w_{j,k}/2$.

After these first two lifting operations, more steps may follow. (this is not the case for the Haar-transform)

6.3. Subdivision, cascade

Inverting this operation is very easy: just go through the scheme in opposite direction (and replace all plus with minus and vice versa):



If we start this algorithm at a coarse scale $j = L$ with

$$s_{L,k} = \delta_k$$

and all wavelet coefficients equal to zero, the samples $s_{j,k}$ converge to the primal scaling function $\varphi(x)$. After one step in the iteration procedure, we have coefficients $s_{L+1,k}$.
Since all detail coefficients are zero, we can write:

$$\sum_{k\in\mathbb{Z}} s_{L,k}\varphi(2^L x - k) = \sum_{k\in\mathbb{Z}} s_{L+1,k}\varphi(2^{L+1}x - k)$$

And, because $s_{L,k} = \delta_k$, we have for $u = 2^L x$:

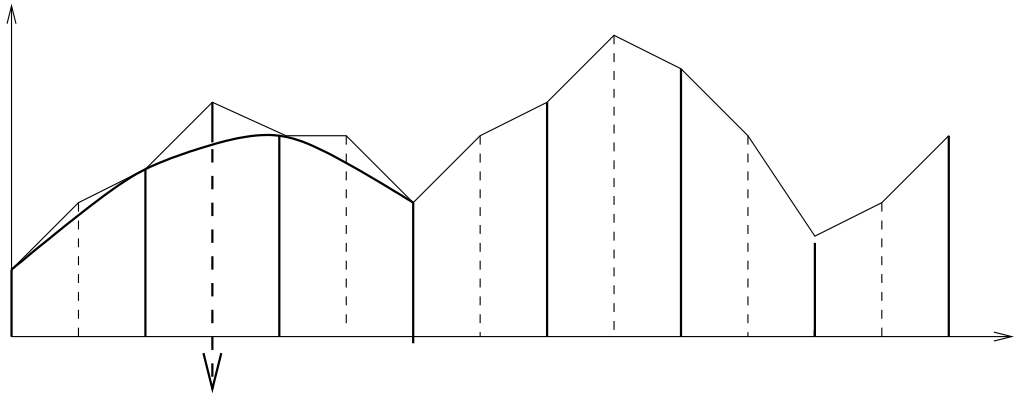
$$\varphi(u) = \sum_{k\in\mathbb{Z}} s_{L+1,k}\varphi(2u - k)$$

Hence, the values after the first step are the primal refinement coefficients.

6.4. Interpolating subdivision

This is the most simple example of subdivision: the lifting scheme has only one prediction-operator and one update.

The predictor is the interpolating polynomial in the even samples



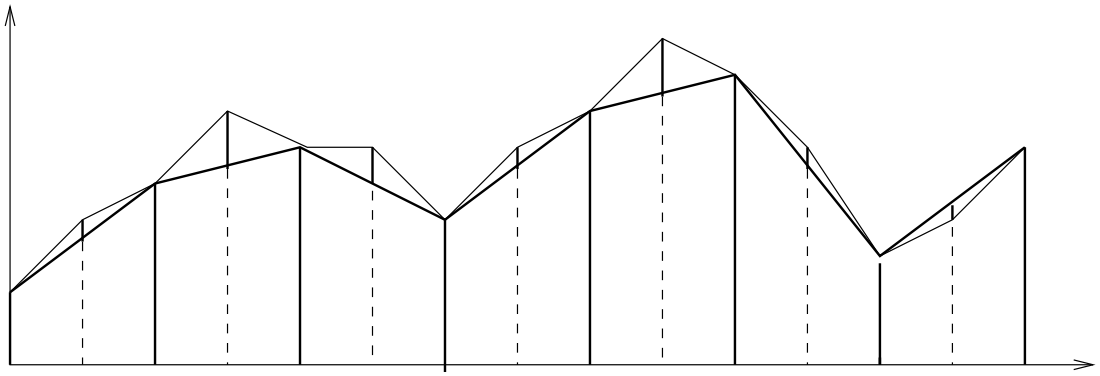
The resulting scaling function has the interpolating property:

$$\varphi(m) = \delta_m$$

6.4.1. The linear interpolating subdivision

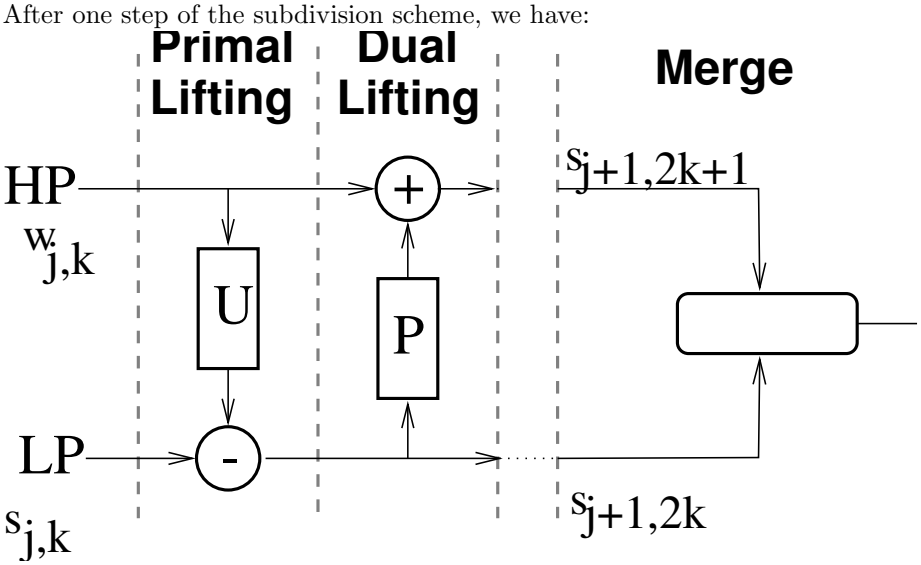
has:

$$P(\boldsymbol{s_{j+1}}) = (1/2)(s_{j+1,2l} + s_{j+1,2l+2})$$



Linear interpolation in odd samples

After one step of the subdivision scheme, we have:

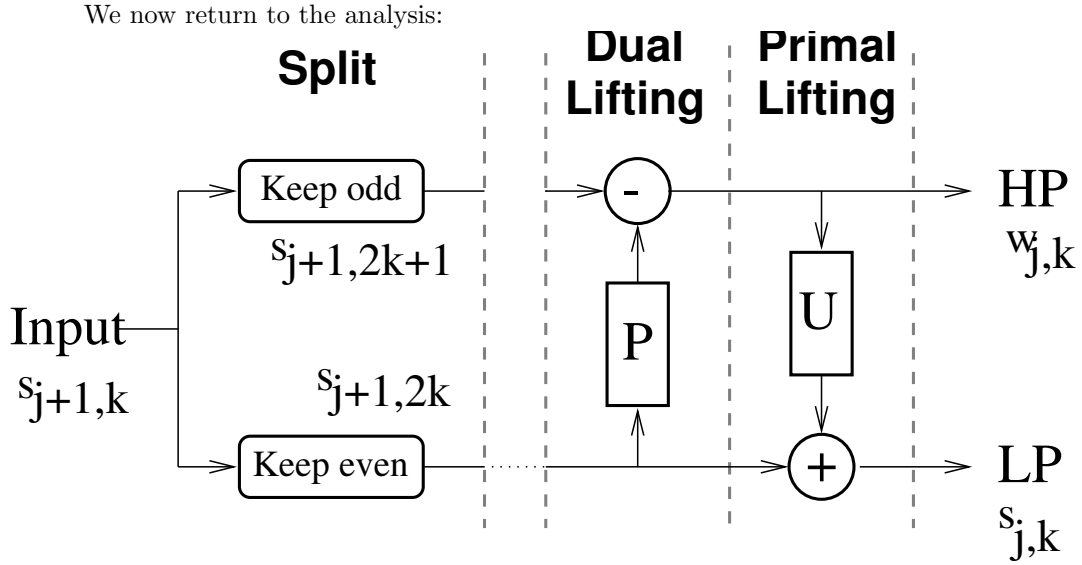


$$
\begin{aligned}
s_{L+1,0} &= s_{L,0} = 1 = c_0 \\
s_{L+1,1} &= 0 + (1/2)(s_{L+1,0} + s_{L+1,2}) = 1/2 = c_1 \\
s_{L+1,-1} &= 0 + (1/2)(s_{L+1,-2} + s_{L+1,0}) = 1/2 = c_{-1}
\end{aligned}
$$

All other refinement coefficients are zero. The algorithm converges to the hat function.

6.4.2. Update

We now return to the analysis:



We see that the same predictor now defines the dual wavelet coefficients, but with alternating sign, this corresponds to

$$\tilde{g}_k = (-1)^k \bar{h}_{1-k}$$

If we would not introduce an update operator, we see that:

$$s_{j,k} = s_{j+1,2k}$$

This would mean that $\tilde{h}_l = \delta_l$ which is too lazy to converge!
Note $w_{j,k} = s_{j+1,2k+1} - \frac{1}{2}(s_{j+1,2k} + s_{j+1,2k+2})$
To determine how to update, we consider the interpretation of one step in wavelet transform as a decomposition of:

$$\sum_{k\in\mathbb{Z}} s_{j+1,k}\varphi(2u-k) = \sum_{k\in\mathbb{Z}} s_{j,k}\varphi(u-k) + \sum_{k\in\mathbb{Z}} w_{j,k}\psi(u-k)$$

(with $u = 2^j x$)
We have seen that $\tilde{\psi}$ must have at least one vanishing moment. This is not strictly necessary for ψ, but wavelets without this property are very irregular. So, we impose:

$$\int_{-\infty}^{\infty} \psi(u)du = 0$$

and integrate the decomposition equation:

$$\sum_{k\in\mathbb{Z}} s_{j+1,k} \int_{-\infty}^{\infty} \varphi(2u-k)du = \sum_{k\in\mathbb{Z}} s_{j,k} \int_{-\infty}^{\infty} \varphi(u-k)du$$

which leads to:

$$\sum_{k\in\mathbb{Z}} s_{j+1,k} = 2\sum_{k\in\mathbb{Z}} s_{j,k}$$

Imposing a second vanishing moment leads to:

$$\sum_{k\in\mathbb{Z}} k s_{j+1,k} = 2 \sum_{k\in\mathbb{Z}} k s_{j,k}$$

This gives two conditions, therefore we need at least two free parameters in the update filter U. We propose:

$$U(\boldsymbol{w_j}) = A w_{j,k-1} + B w_{j,k}$$

and find that:

$$A = 1/4 = B$$

So the forward wavelet transform is:

$$\begin{aligned} w_{j,k} &= -1/2 s_{j+1,2k} + s_{j+1,2k+1} - 1/2 s_{j+1,2k+2} \\ s_{j,k} &= s_{j,2k} + (1/4) w_{j,k} + (1/4) w_{j,k-1} \end{aligned}$$

Substituting the predict in the update, the second equation becomes:

$$\begin{aligned} s_{j,k} = \; & (-1/8) s_{j+1,2k-2} + (1/4) s_{j+1,2k-1} \\ & + (3/4) s_{j+1,2k} \\ & + (1/4) s_{j+1,2k+1} + (-1/8) s_{j+1,2k+2} \end{aligned}$$

From this, one sees that

$$\tilde{\boldsymbol{h}} = ((-1/8), (1/4), (3/4), (1/4), (-1/8))$$

This is the dual filter of the CDF 2,2. The primal filter indeed corresponds to the hat function !!

6.5. Application of the lifting scheme

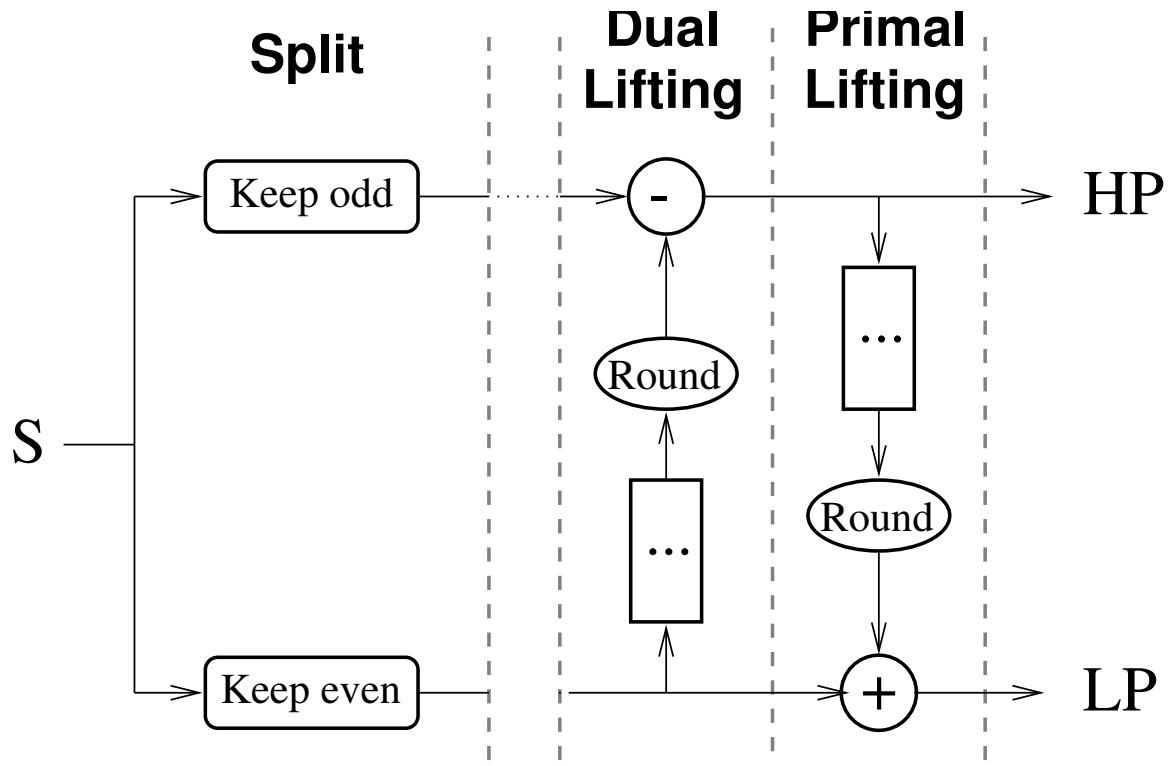
The lifting scheme has following properties:

1. Inverse transform is obvious
2. In place calculation
3. Faster (up to factor 2)
   lifting is a decomposition of polyphase
4. Integer transforms
5. Second generation transforms
6. Arbitrary size (not dyadic)
7. All existing wavelet transforms can be decomposed into lifting steps

6.6. Integer transforms

Digital images are matrices of integer values. A wavelet transform maps these numbers onto reals. Reals need more computer memory. Rounding would cause an invertible transform.

But the following scheme is invertible:



This is the *integer wavelet transform*.
It illustrates the fact that we have freedom in choosing the filters in the lifting steps: the operations even do not have to be linear.

6.7. 2nd generation wavelets

If data are given on an irregular grid (non-equidistant samples), the lifting scheme allows to take into account this grid structure: e.g. interpolating as prediction can be done on an irregular grid.

For finite signals, we can also adapt the filters in the neighbourhood of the begin or end point. In this way, we can define wavelets on an interval.

Imposing a second vanishing moment leads to:

$$\sum_{k\in\mathbb{Z}} k s_{j+1,k} = 2 \sum_{k\in\mathbb{Z}} k s_{j,k}$$

which leads to:

$$\sum_{k\in\mathbb{Z}} s_{j+1,k} = 2\sum_{k\in\mathbb{Z}} s_{j,k}$$

this is:

$$\sum_{k\in\mathbb{Z}} s_{j+1,k} = 2\sum_{k\in\mathbb{Z}} (s_{j+1,2k} + U(\boldsymbol{w_j})_k)$$

From this, there will follow a condition for even and odd k. This gives two conditions, therefore we need at least two free parameters in the update filter U. We propose:

$$U(\boldsymbol{w_j}) = Aw_{j,k-1} + Bw_{j,k}$$

and find that:

$$\begin{aligned}
\sum_{k\in\mathbb{Z}} s_{j+1,k} &= 2\sum_{k\in\mathbb{Z}} (s_{j+1,2k} + Aw_{j,k-1} + Bw_{j,k}) \\
&= 2\sum_{k\in\mathbb{Z}} s_{j+1,2k} + 2(A+B)\sum_{k\in\mathbb{Z}} w_{j,k} \\
&= 2\sum_{k\in\mathbb{Z}} s_{j+1,2k} + 2(A+B) \\
&\quad \cdot \sum_{k\in\mathbb{Z}} \left(s_{j+1,2k+1} - \frac{s_{j+1,2k}}{2} - \frac{s_{j+1,2k+2}}{2} \right) \\
&= 2\sum_{k\in\mathbb{Z}} s_{j+1,2k} \\
&\quad +2(A+B)\sum_{k\in\mathbb{Z}} s_{j+1,2k+1} \\
&\quad -2(A+B)\sum_{k\in\mathbb{Z}} s_{j+1,2k}
\end{aligned}$$

This means for $2k$:

$$1 = 2 - 2(A+B)$$

and for $2k+1$:

$$1 = 2(A+B)$$

Les 4