# Signal Processing for Everyone

Gilbert Strang

In the past, signal processing was a topic that stayed almost exclusively in electrical engineering. It was only the specialists who applied lowpass filters to remove high frequencies from digital signals. The experts could cancel unwanted noise. They could compress the signal and then reconstruct. It took two-dimensional experts to do the same for images.

The truth is that everyone now deals with digital signals and images (involving large amounts of data). We all need to understand signal processing — *sampling, transforming, and filtering.* These pages are intended to explain these basic operations, using simple examples. We will reach as far as filter banks (in discrete time) and wavelet expansions (in continuous time).

*Most signals start their lives in analog form.* They become digital by sampling at equal time intervals. If $x_{\text{analog}}(t)$ is a *continuous time signal*, its samples give a discrete time signal:

$$x_{\text{digital}}(n) = x_{\text{analog}}(nT) \quad n = 0, \pm 1, \pm 2, \dots \tag{1}$$

The *sampling interval* is $T$. We often normalize to $T = 1$, by a simple rescaling of the time variable.

A device that actually does this sampling is called an *A-to-D converter.* The input is an analog (A) signal, probably from measurements. The output is a digital (D) signal, probably for computer processing. Usually an A-to-D converter loses high frequency information (or mixes it with low frequencies, which is aliasing). Shannon's Theorem will tell us that when there are no high frequencies in the signal, the analog signal can be recovered at all $t$ from its (digital) samples at the discrete times $nT$.

Notice that the signal is assumed to be infinitely long, with no start and no finish. The time line is $-\infty < t < \infty$. Then the discrete signal $x(n)$ is defined for all integers $(-\infty < n < \infty)$. Neither of these assumptions is exactly true for real signals. The realistic assumption (this is often well justified) is that the signal is so long that end effects are not significant. By working with the whole line $\mathbf{R}$ and all integers $\mathbf{Z}$, we can use Fourier methods to the utmost.

And those Fourier methods are very powerful. The chief tool in our analysis will be the Discrete Time Fourier Transform, which turns the samples $x(n) = x_{\text{digital}}(n)$ into the coefficients of a $2\pi$-periodic function $X(\omega)$:

$$X(\omega) = \sum_{n=-\infty}^{\infty} x(n) e^{-in\omega} . \tag{2}$$

All terms are unchanged when $\omega$ is increased by $2\pi$. We refer to $\omega$ as the *frequency*, and we graph the transform $X(\omega)$ between $\omega = -\pi$ and $\omega = \pi$. Then "low frequencies" refer to frequencies near zero, and "high frequencies" have $|\omega| \approx \pi$.

Two special signals have the lowest and highest frequencies, $\omega = 0$ and $\omega = \pi$. The pure DC signal $x = (\ldots, 1, 1, 1, 1, 1, \ldots)$ has exactly zero frequency. Its transform $X(\omega)$ has a Dirac delta function at $\omega = 0$. More precisely, $X(\omega)$ is a periodic train of delta functions of magnitude $2\pi$. The pure AC signal $x = (\ldots, 1, -1, 1, -1, \ldots)$ has the highest frequency $\omega = \pi$ (and $\omega = -\pi$). Its transform is a train of delta functions at $\omega = \pm\pi, \pm 3\pi, \ldots$.

This alternation between 1 and $-1$ gives the fastest oscillation of any discrete signal. Between $\omega = 0$ and $\omega = \pm\pi$ is the family of pure sinusoidal signals with frequency $-\pi < \omega < \pi$:

$$x_\omega(n) = e^{in\omega} \qquad \text{for each } n. \tag{3}$$

We are frequently working with systems that respond to these pure inputs with pure outputs. The output has *no change in frequency.* The only change is in amplitude and phase, from multiplication by $H(\omega)$. This is a Linear Time Invariant system:

**LTI systems:** ***The input*** $x_\omega(n)$ ***produces the output*** $H(\omega) x_\omega(n)$**.** (4)

The amplifying factor $H(\omega)$, also written $H(e^{i\omega})$, is the *frequency response.* It varies from one frequency to another, but separate frequencies stay separate. $H(\omega)$ is an "eigenvalue" of the system, when the eigenvector is the oscillating signal $x_\omega(n)$. A Linear Time Invariant system is often called a *filter.*

We will study filters in detail. First we look again at these special signals — complex exponentials and real sinusoids. Fourier (and Mozart too) assembled all signals out of these pure harmonics.

# 1 Sinusoidal Signals

The special signal that we call $x_\omega$ has the complex exponential $e^{in\omega}$ as its $n^{\text{th}}$ sample. Its pure frequency is $\omega$. A more general complex exponential has a positive real amplitude $A$ (not necessarily 1) and a real phase shift $\theta$ (not necessarily 0):

$$x(t) = Ae^{i(\omega t+\theta)}\,. \tag{5}$$

¿From the great formula $e^{i\theta} = \cos\theta + i\sin\theta$, the real part and imaginary part are a cosine signal and a sine signal:

$$\begin{aligned} \mathrm{Re}\{Ae^{i(\omega t+\theta)}\} &= A\cos(\omega t+\theta) \\ \mathrm{Im}\{Ae^{i(\omega t+\theta)}\} &= A\sin(\omega t+\theta) \end{aligned} \tag{6}$$

Notice how these signals have the same frequency $\omega$ and the same amplitude $A$ and a phase difference of $\frac{\pi}{2}$ (or 90°). A cosine function turns into a sine function if we shift by $\frac{\pi}{2}$ radians:

$$\sin(\theta) = \cos\left(\theta - \tfrac{\pi}{2}\right)\,.$$

The complex exponential is nice because it is a single function. The sinusoid representation is nice because the cosine and sine are real. Let us emphasize again that one complex function produces two real functions. But two complex functions can also produce *one* real function:

$$\begin{aligned} e^{i\omega t} + e^{-i\omega t} &= 2\cos\omega \\ e^{i\omega t} - e^{-i\omega t} &= 2\sin\omega\,. \end{aligned} \tag{7}$$

So a real sinusoid like $\cos\omega t$ or $\sin\omega t$, apparently with only one pure frequency $\omega$, actually has *two frequencies* $\omega$ and $-\omega$!

### Note on the relation of frequency to period

The pure sinusoid $x(t) = \cos\omega t$ repeats whenever the time $t$ is increased by $2\pi/\omega$. This is its period $T$:

$$\text{The } \textit{period} \text{ of } x(t) = \cos\omega t \text{ is } T = \frac{2\pi}{\omega}\,. \tag{8}$$

Suppose $T$ is measured in seconds. Then the number of cycles in one second is $f = 1/T$:

$$\text{The } \textit{frequency in Hertz} \text{ is } \quad f = \frac{1}{T} = \frac{\omega}{2\pi}\,. \tag{9}$$

Thus 60 cycles per second is the same as 60 Hertz. A radio wave (whose existence was demonstrated by Heinrich Hertz) might have a frequency of

$10^5$Hz. Notice the simple relation $f = \omega/2\pi$ between our two measures of frequency. We graph a function $H(\omega)$ between $\omega = -\pi$ and $\omega = \pi$. Using the normalized frequency the graph goes from $f = -0.5$ to $f = 0.5$. When the frequency doubles, the period of the signal is halved.

### Note on the relation of phase shift to time shift

The symbol $\theta$ represents the phase shift. The total phase is $\omega t+\theta$, the argument of the sine or cosine, and $\theta$ clearly produces a shift. But there is another way to obtain the same result. *We could shift the time variable $t$ by an amount $\theta/\omega$.* When $\omega$ multiplies the shifted time variable $t' = t + (\theta/\omega)$, the result $\omega t' = \omega t + \theta$ accounts for the phase shift.

There is one small point. By worldwide agreement, a "positive" time shift represents a "delay." A function $x(t)$ undergoing a unit delay becomes $x(t-1)$. The event that originally happened at $t = 5$ now waits until $t = 6$ (because then $t - 1 = 5$). The graph shifts one unit *to the right*. This is a fact of life, that replacing the argument by $t' = t - 1$ will delay the event by a unit time. (And changing the argument to $t'' = t + 1$ will *advance* the event by a unit time.)

The complication is that *a positive time shift* (a delay) *corresponds to a negative phase shift $\theta$*. We had one sign in $\omega t + \theta$ and we have the other sign in $\omega(t - 1)$. So this time shift of 1 corresponds to a phase shift $\theta = -\omega$. In terms of the period $T = 2\pi/\omega$ this is:

$$\Delta t \text{ corresponds to phase shift } \theta = -\tfrac{2\pi}{T}\Delta t. \tag{10}$$

The phase shift $\theta = \frac{\pi}{2}$ between the sine and cosine corresponds to a time delay (shift to the right) of a quarter-period $\Delta t = T/4$.

### Note on the sampling period and aliasing

To get a useful discrete-time signal from an analog signal, we must sample often enough. But too many samples will be expensive and possibly redundant (unless we are using the sample to draw a continuous graph, as below). Of course it is the ratio between the sampling period and the true oscillation period that is critical. The question is how many samples to take in each up-down oscillation of a sinusoid $\cos\omega_0 t$.

We start with a small number of samples:

**1**. *The critical sampling period $T_{\text{Nyquist}}$ has two samples per oscillation.* Figure 1 shows how this can produce the alternating signal $1, -1, 1, \ldots$ with $x(n) = (-1)^n$. This has the fastest oscillation and the highest frequency $\pi$ of any discrete signal:

$$\omega_0 T_{\text{Nyquist}} = \pi \quad \text{and} \quad \frac{1}{T_{\text{Nyquist}}} = \frac{\omega_0}{\pi} \quad \text{and} \quad f_{\text{sampling}} = 2 f_{\text{sinusoid}} \tag{11}$$

This Nyquist rate is the borderline between undersampling and oversampling. If the continuous signal is a combination of many sinusoids, the fastest $\omega_0$ in the combination sets its Nyquist rate.

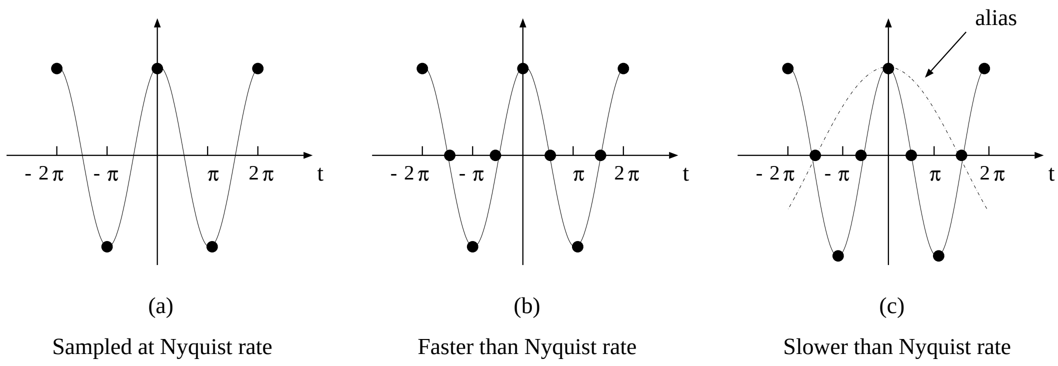



Figure 1: Sampling at 1 and 2 and $\frac{2}{3}$ times the Nyquist rate.

**2**. Sampling at twice the Nyquist rate is *oversampling.* Figure 1b shows the samples $1, 0, -1, 0, 1, 0, \ldots$ with $x(n) = \cos \frac{\pi n}{2}$. This discrete signal has frequency $\omega = \frac{\pi}{2}$.

**3**. Sampling at $\frac{2}{3}$ of the Nyquist rate is *undersampling.* Figure 1c shows how the samples can be the same values $1, 0, -1, 0, 1, 0, \ldots$ as before. We cannot tell from those samples which sinusoid $\cos \omega_0 t$ or $\cos \omega_0 t/3$ is the continuous time signal. *This is aliasing!* The slow frequency $\omega_0/3$ is an alias for the higher frequency $\omega_0$, because the discrete samples are the same. A familiar example is watching a wheel turn in the movies. Often the alias frequency is negative and the wheel appears to be rotating backwards.

A very clear case of undersampling is half the Nyquist rate (only one sample per oscillation). That sample will fall at the same point of every oscillation, so all samples are equal. The discrete signal is then DC (direct current with samples $s, s, s, s, \ldots$ of constant value). The alias of $\omega = 2\pi/T$ is $\omega = 0$.

4. Eight samples in a period will give a reasonably good representation. This linear interpolation is often adequate but it is certainly not perfect.

5. Eighty samples in a period will produce a very lifelike graph. A curved sinusoid is actually constructed in MATLAB by linear interpolation.

## Shannon Sampling Theorem

*Every analog signal $x(t)$ with frequencies not exceeding $\omega_{\max}$ can be perfectly reconstructed from its discrete samples $x(nT)$, provided the sampling rate $1/T$ exceeds $2f_{\max} = \omega_{\max}/\pi$.*

The conclusion of the Shannon Sampling Theorem is always amazing to me, that a band-limited analog signal (continuous time and low frequencies only) can be exactly recovered from a *countable number* of samples (discrete time). This fact is fundamental to communications and digital signal processing. Sinusoids can be recovered from samples that are taken faster than the Nyquist rate.

This is not just a statement of good approximation by linear interpolation. It is a case of perfect reconstruction through interpolation by "sinc functions." The interpolation formula (12) will be stated again (with proof). We are sampling the theorem twice in one book, as Shannon would have wished.

Notice that the signal $x(t) = \sin \omega t$ is not correctly reconstructed from its samples when $1/T$ is *exactly* the Nyquist rate $\omega/\pi$. Each sample $x(nT) = \sin \omega nT = \sin n\pi$ is zero! Shannon's formula (12) will give zero. Reconstruction requires a strict inequality $\omega_{\text{signal}} T_{\text{sampling}} > \pi$, and equality (the Nyquist rate) cuts it too close.

Here is Shannon's interpolation formula. The function $(\sin t)/t$ is known as the *sinc function*. It equals 1 at $t = 0$. By shifting to $t - nT$, we center a sinc function at the sampling point $nT$. By scaling with $\pi/T$, we make it vanish at all other sampling points:

$$x(t) = \sum_{n=-\infty}^{\infty} x(nT) \frac{\sin(t - nT)\pi/T}{(t - nT)\pi/T} \tag{12}$$

This is a case when the D-to-A converter, from samples back to the original function, is an exact inverse of the A-to-D converter. Normally high frequency information is lost in the A-to-D step. It is irretrievably mixed with low frequency information, because of aliasing. Shannon's assumption is that *the signal has no high-frequency information.* It is assumed to be band-limited.

No aliasing occurs, no information is lost, and the transformation from the A function $x(t)$ to the D signal $x(nT)$ can be reversed.

# 2 FIR Filters

A filter is the most important operation in signal processing. It acts on a signal to produce a modified signal. Usually some frequency components of the input signal are reduced; it is remarkable how simply this can be done. When the filter is FIR (*finite* impulse response), each output sample $y(n)$ is just a linear combination of a *finite* number of input samples.

The simplest example is a moving average

$$y(n) = \frac{1}{2}x(n) + \frac{1}{2}x(n-1)\,. \tag{13}$$

This filter combines each sample $x(n)$ with the previous sample $x(n-1)$. The weights in the linear combination are the filter coefficients $\frac{1}{2}$ and $\frac{1}{2}$. The filter is *time-invariant* because those coefficients are constant for all time. The filter is *causal* because it involves no future samples like $x(n+1)$. The effect $y(n)$ never comes earlier than its causes $x(n)$ and $x(n-1)$. Thus we have a causal linear time-invariant FIR system.

A noncausal system cannot operate on a real-time signal, because the input would not be available when the output is required. A pure delay $y(n) = x(n-1)$ is causal, and acceptable in real-time. A pure advance $y(n) = x(n+1)$ is anticausal and not acceptable. This certainly applies to audio signals. For an image the situation is different, because $n$ refers to position not time. The complete image may be available and the filters in image processing need not be causal.

What does this particular "running average" filter do to the input signal $x(n)$? Consider first three special inputs, an impulse and a constant (DC) signal and an alternating (AC) signal. Here are the outputs from those inputs:

I.
$$\begin{aligned} \text{Impulse } x(n) &= (\ldots,0,0,1,0,0,\ldots) \\ \text{Impulse Response } y(n) &= (\ldots,0,0,\tfrac{1}{2},\tfrac{1}{2},0,\ldots) \end{aligned}$$

The impulse response contains the filter coefficients!

II.
$$\begin{aligned} \text{Constant } x(n) &= (\ldots,1,1,1,1,1,\ldots) \\ \text{Averaged Output } y(n) &= (\ldots,1,1,1,1,1,\ldots) \end{aligned}$$

The response exactly equals the input; $\omega = 0$ is in the *passband*.

$$\text{III.} \qquad \begin{aligned} \text{Alternating } x(n) &= (\ldots,1,-1,1,-1,1,\ldots) \\ \text{Averaged Output } y(n) &= (\ldots,0,0,0,0,0,\ldots) \end{aligned}$$

The response is zero; the frequency $\omega = \pi$ is in the *stopband.*

We conclude that the averaging filter is *lowpass.* Low frequencies are mostly passed, high frequencies are mostly stopped. To understand the specifics of that word "mostly," we have to choose input frequencies between $\omega = 0$ and $\omega = \pi$. So the input signal will now be $x(n) = e^{i\omega n}$, at the pure frequency $\omega$. The crucial point is that *the output signal* $y(n)$ *is also at frequency* $\omega$:

$$x(n) = e^{in\omega} \text{ produces } y(n) = \frac{1}{2}e^{in\omega} + \frac{1}{2}e^{i(n-1)\omega} = \left[\frac{1}{2} + \frac{1}{2}e^{-i\omega}\right] e^{in\omega}. \tag{14}$$

The output frequency is the same $\omega$, but the amplitude and phase are changed. The filter multiplies each frequency component of the input by the *frequency response* function

$$H(e^{i\omega}) = \frac{1}{2} + \frac{1}{2}e^{-i\omega}. \tag{15}$$

At $\omega = 0$, this frequency response is $H = 1$. Therefore the constant signal passes unchanged through the filter (as we know). At $\omega = \pi$, the frequency response is $H = 0$. Therefore the alternating signal is completely blacked out by the averaging filter. We now separate the function $H(e^{i\omega})$ into its amplitude and phase, to see what happens to each individual frequency:

$$H(e^{i\omega}) = \frac{1}{2} + \frac{1}{2}e^{-i\omega} = e^{-i\omega/2}\left(\frac{1}{2}e^{i\omega/2} + \frac{1}{2}e^{-i\omega/2}\right) = e^{-i\omega/2}\left(\cos\frac{\omega}{2}\right). \tag{16}$$

The amplitude is $|H| = \cos\frac{\omega}{2}$. It drops from one at $\omega = 0$ to zero at $\omega = \pi$. The graph of $|H|$ from $-\pi$ to $\pi$ is one arch of a cosine. It shows a passband and a stopband, where the multiplying factor $H(e^{i\omega})$ is near one and near zero. For this very short filter, the *transition band* in between is very wide. This is a somewhat crude lowpass filter, but extremely simple and inexpensive.

The phase of $H(e^{i\omega})$ in equation (16) is $\phi = -\omega/2$. For this filter, the phase depends linearly on $\omega$. This property of *linear phase* follows directly from the symmetry of the filter coefficients $h(0) = \frac{1}{2}$ and $h(1) = \frac{1}{2}$. Reversing the order produces no change. More precisely, the coefficients are symmetric around their middle point (at $\frac{1}{2}$):

Symmetry $h(n) = h(1-n)$ around $\frac{1}{2}$ produces the linear phase $-\frac{1}{2}\omega$.

This is one example of a general rule: *symmetry produces linear phase.* That is a highly important property in image processing, because the eye catches any failure of symmetry after an image is compressed.

## Moving Difference (Highpass Filter)

A second quick example will reinforce these points, by setting up a contrast. Instead of an averaging filter (coefficients $\frac{1}{2}$ and $\frac{1}{2}$) we take differences:

$$y(n) = \frac{1}{2}x(n) - \frac{1}{2}x(n-1)\,. \tag{17}$$

This is still FIR and causal. Suppose it acts on the three special input signals. The response to a unit impulse is the filter coefficients $\frac{1}{2}$ and $-\frac{1}{2}$. The all-ones input has zero differences, so the lowest frequency $\omega = 0$ is stopped. The highest frequency $\omega = \pi$ is passed without any change. This is a *highpass filter*:

| | | | | |
|---|---|---|---|---|
| **I.** | Impulse | $x(n) = \delta(n)$ | Impulse response | $y(0) = \frac{1}{2}\,,\ y(1) = -\frac{1}{2}$ |
| **II.** | Constant | $x(n) = 1$ | Zero output | $y(n) = 0$ |
| **III.** | Alternating | $x(n) = (-1)^n$ | Alternating output | $y(n) = (-1)^n$. |

As for every linear time-invariant system, the response to a pure frequency signal $x(n) = e^{in\omega}$ is a multiple $y(n) = H(e^{i\omega})e^{in\omega}$ of that signal:

$$x(n) = e^{in\omega} \text{ produces } y(n) = \frac{1}{2}e^{in\omega} - \frac{1}{2}e^{i(n-1)\omega} = \left[\frac{1}{2} - \frac{1}{2}e^{-i\omega}\right] e^{in\omega}\,. \tag{18}$$

The multiplying factor is the frequency response function $H(e^{i\omega})$. Again we separate the phase factor from the amplitude $|H|$:

$$\frac{1}{2} - \frac{1}{2}e^{-i\omega} = e^{-i\omega/2}\left(\frac{1}{2}e^{i\omega/2} - \frac{1}{2}e^{-i\omega/2}\right) = e^{-i\omega/2}\, i \sin\frac{\omega}{2}\,. \tag{19}$$

The amplitude is $|H| = \sin\frac{\omega}{2}$. This is zero at $\omega = 0$ and one at $\omega = \pi$. The filter is highpass (but again not very sharp). The amplitude response is now a sine instead of a cosine.

The phase factor is $e^{-i\omega/2}i$. The extra factor $i = \sqrt{-1}$ appears because this filter is *antisymmetric*. We still call this "linear phase," although strictly speaking the linearity ends at $\omega = \pi$ where $\sin\frac{\omega}{2}$ changes sign. (At that point the magnitude changes to $-\sin\frac{\omega}{2}$. Therefore the phase must jump by $\pi$ to produce a sign change $e^{i\pi} = -1$. So the true phase function $\phi(\omega)$ is linear with jumps.) In short: *linear phase filters are symmetric or antisymmetric around their centers.*

Let us emphasize the difference between lowpass and highpass. A lowpass filter preserves the *smooth* part of the signal. A highpass filter preserves the *rough and noisy* part. In wavelet language, lowpass gives averages and highpass gives details. In some applications those details are important to keep (like edges in an image). In other applications the high frequencies are mostly noise (from measurements). A good lowpass filter has many uses, so we look now at better filters with more coefficients.

# 3 Better FIR Filters

A causal FIR filter of order $N$ has coefficients $h(0), h(1), \ldots, h(N)$. Notice that there are $N+1$ coefficients; this is the *length* of the filter. These coefficients stay fixed for all time so the filter is time-invariant.

At each time step, the $N+1$ coefficients multiply $N+1$ samples from the input signal — the current sample $x(n)$, the previous sample $x(n-1)$, continuing back to the sample $x(n-N)$. This weighted combination of input values produces the output $y(n)$:

$$y(n) = h(0)x(n) + h(1)x(n-1) + \cdots + h(N)x(n-N)\,. \tag{20}$$

This is the action of the filter in the time domain. We may write it compactly as a sum from $k=0$ to $k=N$:

$$y(n) = \sum_{k=0}^{N} h(k)x(n-k)\,. \tag{21}$$

*A filter is a discrete convolution!* It is the fundamental operation for discrete time-invariant systems. To implement this convolution in hardware, we only need three building blocks: *unit delay, multiplier*, and *adder*. To account for the typical term $h(1)x(n-1)$ in equation (20), the three operations are represented in Figure 2:

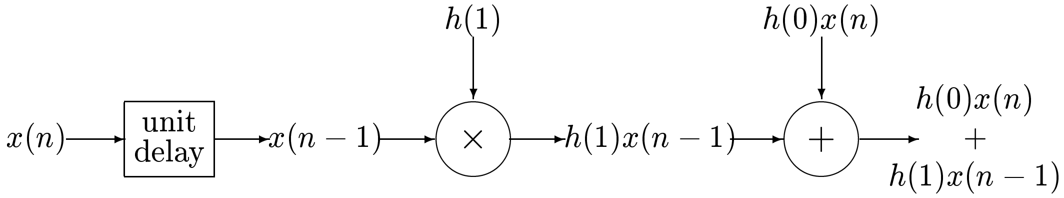



Figure 2: The three blocks that build every FIR filter: **delay, multiply, add**.

In a hardware implementation, the sample value $x(n-1)$ is stored in memory for one clock cycle and released. A double delay to $x(n-2)$ is a cascade of two unit delays. The $N$-unit delay in the convolution uses $N$ memory cells and a circular buffer. Basically we have a shift register.

Modern DSP microprocessors often combine "multiply-add" into one special unit whose speed is critical. We multiply and accumulate, exactly like a dot product of vectors, $h \cdot x = h_1 x_1 + h_2 x_2 + \cdots + h_n x_n$. In numerical linear algebra this is executed in double precision to minimize the damage from cancellation of large numbers.

We can implement a filter in MATLAB using the convolution command **conv**:

$$\begin{aligned} \boldsymbol{x} &= \mathbf{0:4+\sin(pi*(0:4)/2)\,;} \\ \boldsymbol{h} &= \mathbf{[0.25\ 0.5\ 0.25]\,;} \\ \boldsymbol{y} &= \mathbf{conv}(\boldsymbol{h},\boldsymbol{x}) \end{aligned}$$

The 5-point input vector is linear plus sinusoidal: $x = [0\ 1\ 2\ 3\ 4] + [0\ 1\ 0\ -1\ 0]$. The linear part is smooth. The sinusoid has discrete frequency $\omega = \frac{\pi}{2}$ because it takes two time intervals (not just one) to oscillate from its maximum to its minimum (from 1 to $-1$).

The filter $h = \left[\frac{1}{4}, \frac{1}{2}, \frac{1}{4}\right]$ is *lowpass* and *causal* and *linear phase* (it is symmetric around its center). The high frequency $\omega = \pi$ is eliminated because the alternating sum is $\frac{1}{4} - \frac{1}{2} + \frac{1}{4} = 0$. The all-ones signal is preserved because the filter coefficients sum to 1. Actually the linear signal $[0\ 1\ 2\ 3\ 4]$ is also preserved (except at the ends of the signal), but you will see how *the causal filter introduces a delay*. Let us display that part $y_{\text{linear}} = \mathbf{conv}(0:4,\ h)$ as an ordinary multiplication of the polynomials $(0 + x + 2x^2 + 3x^3 + 4x^4)$ and $(\frac{1}{4} + \frac{1}{2}x + \frac{1}{4}x^2)$:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| $x(n)$ | 0 | 1 | 2 | 3 | 4 | | |
| $h(n)$ | $\frac{1}{4}$ | $\frac{1}{2}$ | $\frac{1}{4}$ | | | | |
| $h(0)x(n)$ | 0 | $\frac{1}{4}$ | $\frac{2}{4}$ | $\frac{3}{4}$ | $\frac{4}{4}$ | | |
| $h(1)x(n-1)$ | | 0 | $\frac{1}{2}$ | $\frac{2}{2}$ | $\frac{3}{2}$ | $\frac{4}{2}$ | |
| $h(2)x(n-2)$ | | | 0 | $\frac{1}{4}$ | $\frac{2}{4}$ | $\frac{3}{4}$ | $\frac{4}{4}$ |
| $\mathbf{conv}(h, x_{\text{linear}})$ | 0 | $\frac{1}{4}$ | 1 | 2 | 3 | $2\frac{3}{4}$ | 1 |
| | end effect | | *delay to* $x(n-1)$ | | | end effect | |

Notice that there are *seven* outputs. An input signal of length $L$ convolved with an order $N$ filter produces an output of length $L + N$. A fourth degree polynomial times a second degree polynomial yields a sixth degree polynomial. Five terms convolved with three terms produce seven terms.

At the center of the output, the linear inputs $1, 2, 3$ are preserved (with a delay). The reason is that not only $\sum h(k) = 1$ (which preserves constant signals) but also $\sum kh(k) = 1$ (which preserves linear signals).

The end effects have length $N = 2$, the order of the filter. The effect appears at the left end because we don't have the samples $x(-2)$ and $x(-1)$ that should contribute to $y(0)$ and $y(1)$. (If those missing samples are both zero than the left end is correct as it stands.) The effect appears at the right end because we don't have the samples $x(5)$ and $x(6)$. (We have run out of samples to match with the filter window.) One of the possible techniques to produce an output of length $L = 5$ instead of $L + N = 7$ is *wrap-around* or *cycling* or *periodicity.* The right end $(2\frac{3}{4}, 1)$ is added to $(0, \frac{1}{4})$ at the left end. This is *circular convolution* and for periodic signals it is natural.

Now we input the sinusoid $[0\ 1\ 0\ -1\ 0]$. The output from a filter should be a sinusoid of the same frequency, but with a different (reduced) amplitude. Again we have to overlook the end effects:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| $x(n)$ | $0$ | $1$ | $0$ | $-1$ | $0$ | | |
| $h(n)$ | $\frac{1}{4}$ | $\frac{1}{2}$ | $\frac{1}{4}$ | | | | |
| $h(0)x(n)$ | $0$ | $\frac{1}{4}$ | $0$ | $-\frac{1}{4}$ | $0$ | | |
| $h(1)x(n-1)$ | | $0$ | $\frac{1}{2}$ | $0$ | $-\frac{1}{2}$ | $0$ | |
| $h(2)x(n-2)$ | | | $0$ | $\frac{1}{4}$ | $0$ | $-\frac{1}{4}$ | $0$ |
| $\mathbf{conv}(h, x_{\sin})$ | $0$ | $-\frac{1}{4}$ | $\frac{1}{2}$ | $0$ | $-\frac{1}{2}$ | $-\frac{1}{4}$ | $0$ |
| | left end | | sinusoid | | | right end | |

The output sinusoid has amplitude $\frac{1}{2}$. There is also a unit delay between input and output, because the filter is symmetric around $h(1)$. The center of the filter is at $n = 1$. The only causal filter centered at $n = 0$ is the identity filter $h(n) = \delta(n)$.

How should we have known that the amplitude is reduced by $\frac{1}{2}$ at the discrete frequency $\omega = \frac{\pi}{2}$? The answer for all sinusoids is contained in the frequency response function $H(\omega) = H(e^{i\omega})$. For the filter $h = (\frac{1}{4}, \frac{1}{2}, \frac{1}{4})$ the response is just a polynomial with these coefficients $h(0)$, $h(1)$, and $h(2)$:

$$H(\omega) = \frac{1}{4} + \frac{1}{2}e^{-i\omega} + \frac{1}{4}e^{-2i\omega}. \tag{22}$$

At $\omega = \frac{\pi}{2}$ this is $H = -\frac{1}{2}i$. *The magnitude at that frequency is* $|H| = \frac{1}{2}$, agreeing with the observed 50% reduction in amplitude. The factor $-i = e^{-i\pi/2}$ is responsible for the phase shift (which is the delay).

When a linear filter is applied to a *sum of inputs* (linear plus sinusoid), we get the *sum of outputs*. So when we know what the filter does to sinusoidal signals, we know everything. Those special signals take us into the frequency domain.

# 4 Filters in the Frequency Domain

In the time domain, a filter combines a signal with delays of that same signal. The signal $x(n-k)$ with $k$ delays is multiplied by the filter coefficient $h(k)$. The combination from $k=0$ to $k=N$ is the filtered output $y(n)$:

$$y(n) = \sum_{k=0}^{N} h(k)x(n-k)\,. \tag{23}$$

That is a discrete convolution $y = h * x$ of two vectors.

How does a filter look in the frequency domain? The basic rule is that ***a convolution becomes a multiplication***. We have seen that already for pure sinusoids; they are multiplied by $H(\omega)$. So the same must happen, by linearity, for any combination of sinusoids.

Algebraically, we multiply the Fourier series $H(\omega)$ with coefficients $h(k)$ and the Fourier series $X(\omega)$ with coefficients $x(m)$. The result is the series $Y(\omega)$ with coefficients $y(n)$. In the frequency domain, the output $Y$ is the input $X$ multiplied by the response function $H$:

$$\begin{aligned} \sum y(n)e^{-in\omega} &= \left(\sum h(k)e^{-ik\omega}\right)\left(\sum x(m)e^{-im\omega}\right) \\ Y(\omega) &= H(\omega)X(\omega) \end{aligned} \tag{24}$$

*To get* $e^{-in\omega}$ *in the product of* $e^{-ik\omega}$ *with* $e^{-im\omega}$*, we must have* $k+m=n$*.* That is exactly what we see in the convolution (23). The indices $k$ and $n-k$ add to $n$. The products $e^{-ik\omega}$ and $e^{-i(n-k)\omega}$ multiply to give $e^{-in\omega}$. Equations (23) and (24) match exactly.

This "convolution rule" is more than just algebra because equation (24) has a valuable scientific meaning:

> **Each component $X(\omega)$ of the input is amplified by the filter response $H(\omega)$ (or $H(e^{i\omega})$) at that frequency $\omega$.**

Now consider a combination of frequencies, integrating over $\omega$. We want to see how such a combination can produce every signal $x(n)$. From its formula,

$X(\omega) = \sum x(n)e^{-in\omega}$ is the (complex!) dot product of $\{x(n)\}$ with the pure exponential signal $\{e^{in\omega}\}$. The part of the signal in the direction of that harmonic is $X(\omega)e^{in\omega}$. When these harmonic parts are combined, by integrating from $\omega = -\pi$ to $\omega = \pi$ and dividing by $2\pi$, we recover the original vector $x(n)$:

$$x(n) = \frac{1}{2\pi}\int_{-\pi}^{\pi} X(\omega)e^{in\omega}\, d\omega\,. \tag{25}$$

So $X(\omega)$ is the input component, and $H(\omega)X(\omega)$ is the output component, and we are truly in the frequency domain.

**Example.** Suppose the input $x$ is the unit impulse $\delta = (\ldots, 0, 0, 1, 0, 0, \ldots)$. This special signal combines *all frequencies in equal amounts.* Its transform $X(\omega)$ is a constant $\sum \delta(n)e^{-in\omega} = 1$, because the series has only one term $n = 0$. When we reassemble all parts — multiply the pure exponential vectors $\{e^{in\omega}\}$ by 1 and combine all frequencies by integration as in (25), we recover the impulse as expected:

$$\frac{1}{2\pi}\int_{-\pi}^{\pi} 1\cdot e^{in\omega}\, d\omega = \delta(n) = \left\{\begin{array}{ll} 1 & \text{if } n = 0 \\ 0 & \text{if } n \neq 0 \end{array}\right\}. \tag{26}$$

Now filter this input signal $\delta(n)$, the impulse. In the time domain, the convolution has only one term $k = n$:

$$y(n) = \sum h(k)\delta(n-k) = h(n)\,. \tag{27}$$

In the frequency domain, the rule $Y(\omega) = H(\omega)X(\omega)$ reduces to $Y(\omega) = H(\omega)$. As predicted, this matches $y(n) = h(n)$ in the time domain. ***The transform of the impulse response*** $\{h(n)\}$ ***is the frequency response*** $H(\omega)$***.*** This is the function that describes the filter in the frequency domain:

$$H(\omega) = \sum_{k=0}^{N} h(k)e^{-ik\omega} = h(0) + h(1)e^{-i\omega} + \cdots + h(N)e^{-iN\omega}\,. \tag{28}$$

Let us understand this function graphically. One particular frequency response $H(\omega)$ is shown by Figure 3. The filter is lowpass because low frequencies have $H(\omega) \approx 1$. They pass through the filter. The output $Y(\omega) \approx X(\omega)$ keeps those frequencies. High frequencies have $H(\omega) \approx 0$, so $H(\omega)X(\omega)$ is very small at these frequencies. We say that the output is a *smoothed* version of the input. High frequencies (fast oscillations) are associated with noise, and the filter removes them. If we want to keep them then we use a highpass filter.

You can see this smoothing in the time domain, but maybe not so well. The filter coefficients are the time domain outputs when the input is an impulse.

The vector $h(k)$ of filter coefficients is certainly smoother than $\delta(n)$; the spike at the center is much less prominent. But the details of smoothing are clearer from $H(\omega)$ than $h(k)$. The symmetry of the filter is shown by the fact that the amplitude $|H(\omega)|$ is an even function:

$$h(k) = h(N-k) \quad \text{means that} \quad H(\omega) = (\text{even function})e^{-i\omega N/2}.$$

# 5 Equiripple Lowpass Filters

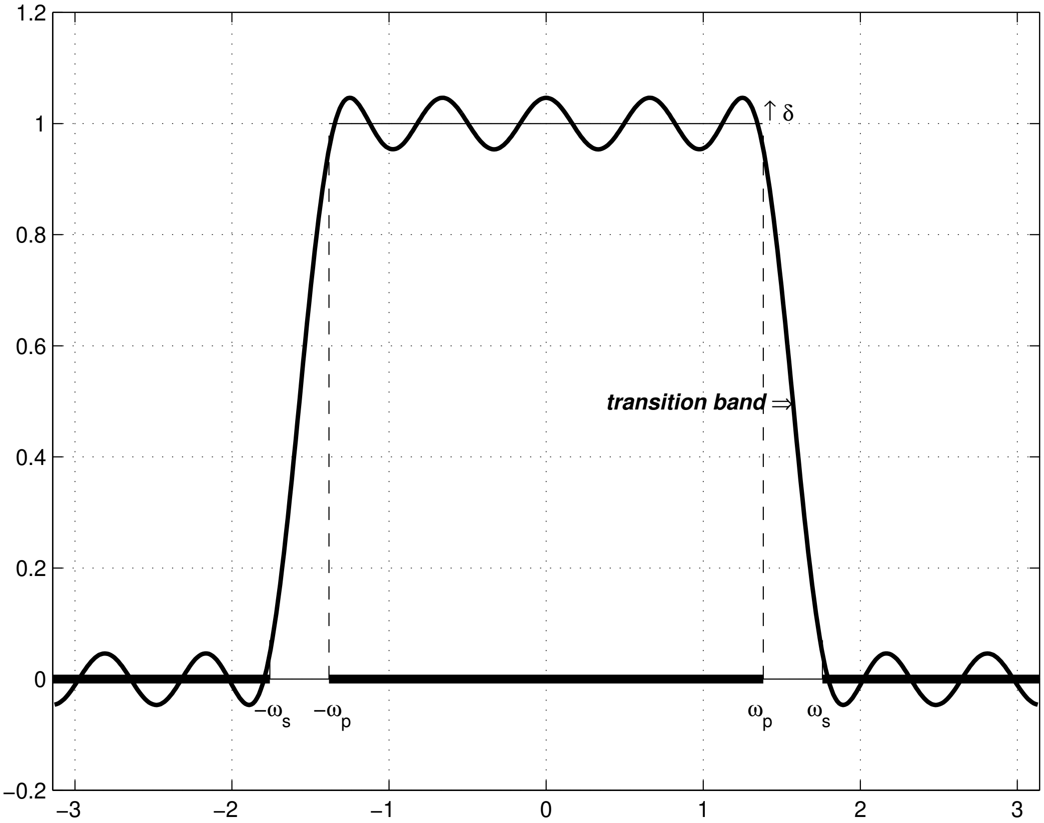



Figure 3: Frequency response for $N = 20$ and $\omega_{\text{pass}} = 0.44\pi$, $\omega_{\text{stop}} = 0.56\pi$.

Figure 3 is the graph of a good lowpass filter of order $N = 20$. Some designers might say it is the best. *The error in the passband* $|\omega| \leq \omega_{\text{pass}}$ *and in the stopband* $|\pi - \omega| \leq \pi - \omega_{\text{stop}}$ *is as small as possible.* That is *maximum error*, not mean-square error. We gave the inputs $N = 20$ and $\omega_{\text{pass}} = 0.44\pi$ and $\omega_{\text{stop}} = 0.56\pi$ to the MATLAB function `remez.m` and it designed the filter.

The output from `remez.m` is the set of filter coefficients $h(0), \ldots, h(20)$. We can multiply $H(e^{i\omega})$ by $e^{-i10\omega}$ to center it, with no change in amplitude. Then the cosine polynomial $|H(e^{i\omega})|$ has *degree ten*; it stops at $\cos 10\omega$.

The special feature of this minimax filter is its *equiripple* property. All "ripples" in the stopband plus passband have the same magnitude. This property tells us immediately that we cannot reduce all errors (the error is the distance from the ideal 1–0 filter in the graph). No correction could have the required plus or minus sign at all twelve frequencies, because those signs are alternating and the correction would cross zero too often (11 times between 0 and $\pi$) for a polynomial of degree 10. This little bit of mathematics, where the degree of the polynomial limits how often it can alternate sign, is the foundation of minimax approximation theory.

The design method used by MATLAB is called the Remez algorithm or the Parks-McClellan algorithm. The Remez idea of iteratively reducing the largest error until the graph is equiripple was adapted by Parks and McClellan for filter design. Users often assign a heavier weight $W > 1$ to the errors in the stopband, and then the stopband ripples have smaller heights $\delta/W$.

In a practical design problem, *we need to know what length of filter to ask for.* Therefore the relation of the error $\delta$ to the order $N$ and the cutoff frequencies ($\omega_{\text{pass}}$ and $\omega_{\text{stop}}$) is highly important in practice. Certainly $\delta$ decreases exponentially with $N$. The key exponent in $e^{-\beta N}$ was estimated by Jim Kaiser from numerical experiments, and his empirical formula has been adjusted by many authors. A recent paper [12] by Shen and the author gave the asymptotically correct formula

$$N \simeq \frac{20\log_{10}(\pi\delta)^{-1} \; - 10\log_{10}\log_{10}\delta^{-1}}{(10\log_{10}e)\;\ln\cot\frac{\pi-\Delta\omega}{4}}. \tag{29}$$

Here $\Delta\omega = \omega_{\text{stop}} - \omega_{\text{pass}}$ is the *transition bandwidth.* (We use the awkward form $20\log_{10}$ because this gives the quantity in *decibels.*) In the transition band between $\omega_{\text{pass}}$ and $\omega_{\text{stop}}$, the dropoff from $H(\omega) \approx 1$ to $H(\omega) \approx 0$ is described by an error function. Shen's formula

$$H(\omega) \simeq \operatorname{erf}\left(\sqrt{\frac{N\beta}{4}}\,\frac{\omega_m - \omega - S_N(W)}{\omega_{\text{stop}} - \omega_m}\right), \tag{30}$$

with

$$S_N(W) = \frac{\ln W + \frac{1}{2}\ln\ln W}{2N}.$$

gives a good fit in this "don't care region" around a critical frequency near $\omega_m = \frac{1}{2}(\omega_{\text{pass}} + \omega_{\text{stop}})$. As $N \to \infty$ the equiripple filters approach the ideal one-zero filter with cutoff at $\omega_{\text{critical}}$.

Of course the equiripple filter is not the only candidate! If we accept larger ripples near the endpoints $\omega_{\text{pass}}$ and $\omega_{\text{stop}}$, we can have smaller ripples inside the passband and stopband. A fairly extreme case (not usually recommended) is a *least squares filter design.* The filter minimizes the *energy* in the error, possibly weighted by $W$, instead of the maximum error:

$$\textit{Choose } H(\omega) \textit{ to minimize} \quad \int\limits_{\text{passband}} |H(\omega)-1|^2\, d\omega \; + \; W \int\limits_{\text{stopband}} |H(\omega)|^2\, d\omega\,. \tag{31}$$

We know the result when the passband meets the stopband ($\omega_{\text{pass}} = \omega_{\text{stop}}$). Then $H(\omega)$ is approximating an ideal 1–0 square wave or brick wall filter. *There is a Gibbs oscillation at the discontinuity.* The maximum error $E$, an overshoot right at the jump, quickly approaches the famous Gibbs constant. For large order $N$, the worst ripples have fixed height even though the integral of $(\text{ripples})^2$ is as small as possible. This Gibbs phenomenon is one of the great obstacles to accurate computations near a shock front.

# 6 Wavelet Transforms

A lowpass filter greatly reduces the high frequency components, which often represent noise in the signal. For some purposes that is exactly right. But suppose we want to *reconstruct* the signal. We may be storing it or transmitting it or operating on it, but we don't want to lose it. In this case we can use *two filters*, highpass as well as lowpass. That generates a "**filter bank**," which sends the signal through two or more filters.

The filter bank structure leads to a *Discrete Wavelet Transform.* This has become a guiding idea for so many problems in signal analysis and synthesis. In itself the transform is lossless! Its inverse (the synthesis step) is another transform of the same type — two filters that are fast to compute. Between the DWT and the inverse DWT we may compress and transmit the signal. This sequence of steps, *transform then compress then reconstruct*, is the key to more and more applications.

The word *wavelet* is properly associated with a *multiresolution into different scales.* The simplest change of scale comes from downsampling a signal — keeping only its even-numbered components $y(2n)$. This sampling operation is denoted by the symbol $\downarrow 2$:

$$(\downarrow 2) \begin{bmatrix} y(0) \\ y(1) \\ y(2) \\ y(3) \end{bmatrix} = \begin{bmatrix} y(0) \\ y(2) \end{bmatrix}.$$

Information is lost. But you will see how *doubling* the length by using two filters, then *halving* each output by $(\downarrow 2)$, can give a lossless transform. The input is at one time scale and the two half-length outputs are at another scale (an octave lower).

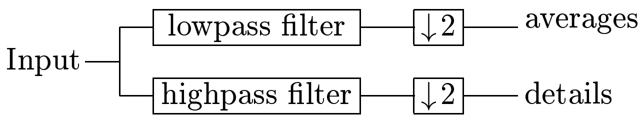



Figure 4: The discrete wavelet transform: averages and details.

Note that an input of even length $L$ produces two outputs of length $L/2$, after downsampling. The lowpass filter $H_0$ and the highpass filter $H_1$ originally maintain length $L$, when we deal suitably with the samples at the ends (possibly by extending the signal to produce the extra components that the filter needs). Figure 4 shows this combination of two filters, with each output downsampled. The redundancy from $2L$ outputs is removed by $(\downarrow 2)$. Then the overall filter bank is $L$ by $L$.

To simplify the theory we often pretend that $L = \infty$. This avoids any difficulty with the samples at the ends. But in reality signals have finite length.

*The wavelet idea is to repeat the filter bank.* The lowpass output in Figure 4 becomes the input to a second filter bank. The computation is cut in half because this input is half length. Typical applications of the wavelet transform go to four or five levels. We could interpret this multiscale transform as (quite long!) filters acting on the very first inputs. But that would miss the valuable information stored in the outputs (averages and details) along the way.

# 7 The Haar Transform

We now choose one specific example of a filter bank. At first there is no iteration (two scales only). Then we iterate to multiple scales. The example is associated with the name of Alfred Haar. It uses the *averaging filter* (moving average) and the *differencing filter* (moving difference). Those were our earliest and simplest examples of a lowpass filter and a highpass filter. So they combine naturally into the most basic example of a filter bank. It will be convenient (but not necessary) to reverse the sign of $H_1$.

The two filters are denoted by $H_0$ (lowpass) and $H_1$ (highpass):

$$\begin{array}{lll} y_0 = H_0 x & \text{is the averaging filter} & y_0(n) = \frac{1}{2}(x(n-1) + x(n)) \\ y_1 = H_1 x & \text{is the differencing filter} & y_1(n) = \frac{1}{2}(x(n-1) - x(n)) \,. \end{array}$$

Suppose the input signal is zero except for four samples $x(1) = 6$, $x(2) = 4$, $x(3) = 5$, $x(4) = 1$. This input vector is $x = (6, 4, 5, 1)$. *We are looking for its coefficients in the Haar wavelet basis.* Those will be four numbers, $y_0(2)$ and $y_0(4)$ from subsampling the lowpass output together with $y_1(2)$ and $y_1(4)$ from highpass.

In reality we would not compute the odd-numbered components $y(1)$ and $y(3)$ since they are immediately destroyed by $(\downarrow 2)$. But we do it here to see the complete picture. Take averages and differences of $x = (6, 4, 5, 1)$:

$$\text{Averages}\ \begin{array}{rcl} y_0(1) & = & 3 \\ y_0(2) & = & 5 \\ y_0(3) & = & 4.5 \\ y_0(4) & = & 3 \\ y_0(5) & = & 0.5 \end{array} \qquad \tfrac{1}{2}\,\text{(Differences)}\ \begin{array}{rcr} y_1(1) & = & -3 \\ y_1(2) & = & 1 \\ y_1(3) & = & -0.5 \\ y_1(4) & = & 2 \\ y_1(5) & = & 0.5 \end{array}$$

You might notice that the sum of the $y_0$ vector and the $y_1$ vector is the input $x = (6, 4, 5, 1)$ with a unit delay (to $x(n-1)$). This comes from a simple relation (average + difference) that is special to Haar:

$$\frac{1}{2}(x(n-1) + x(n)) + \frac{1}{2}(x(n-1) - x(n)) = x(n-1) \,.$$

It is more important to notice that the *differences tend to be smaller than the averages.* For a smooth input this would be even more true. So in a compression step, when we often lose information in the highpass coefficients, the loss is small using the wavelet basis. Here is a first look at the whole compression algorithm:

$$\textit{signal } x \xrightarrow[\text{[lossless]}]{} \begin{array}{c}\textit{wavelet}\\ \textit{coefficients}\end{array} \xrightarrow[\text{[lossy]}]{} \begin{array}{c}\textit{compressed}\\ \textit{coefficients}\end{array} \xrightarrow[\text{[lossless]}]{} \begin{array}{c}\textit{compressed}\\ \textit{signal}\end{array} \widehat{x}$$

At this point we have eight or even ten coefficients in $y_0$ and $y_1$. They are redundant! They came from only four samples in $x$. *Subsample* by $\downarrow 2$ to keep only the even-numbered components:

$$\begin{array}{rcl} y_0(2) & = & 5 \\ y_0(4) & = & 3 \end{array} \qquad \begin{array}{rcl} y_1(2) & = & 1 \\ y_1(4) & = & 2 \,. \end{array}$$

Those are the four "first-level wavelet coefficients" of the signal. The inverse transform (which is coming in the next section) will use those coefficients to reconstruct $x$. That will be the synthesis step. Computing the coefficients was the analysis step:

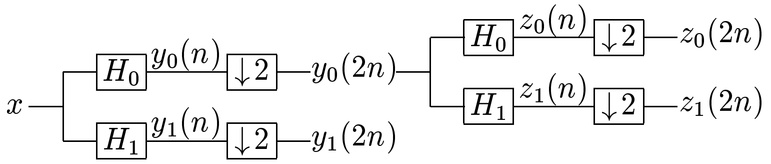
*Analysis:* Find the wavelet coefficients (separate the signal into wavelets)
*Synthesis:* Add up wavelets times coefficients (to reconstruct the signal).

It is like computing Fourier coefficients, and then summing the Fourier series. For wavelets, the *analysis filter bank* ($H_0$ and $H_1$ followed by $\downarrow 2$) computes the coefficients. The *synthesis filter bank* (this inverse wavelet transform will involve upsampling by $\uparrow 2$ and two filters) sums the wavelet series.

Now go from the lowpass $y_0(2) = 5$ and $y_0(4) = 3$ to the next scale by computing *averages of averages and differences of averages*:

$$z_0(2) = \frac{5+3}{2} = 4 \qquad z_1(2) = \frac{5-3}{2} = 1\,.$$

This completes the iteration of the Haar analysis bank. We can see the three scales (fine, medium, and coarse) in a block diagram that shows the tree of filters with subsampling:




Effectively, $z$ comes from downsampling by 4. The vector $z_0(2n)$ contains the *low-low* coefficients, averages of averages. The high-low vector $z_1(2n)$ is also $\frac{1}{4}$ the length of the original signal. The highpass vector $y_1(2n)$ is half the original length, and $\frac{1}{4} + \frac{1}{4} + \frac{1}{2} = 1$ (this is critical length sampling).

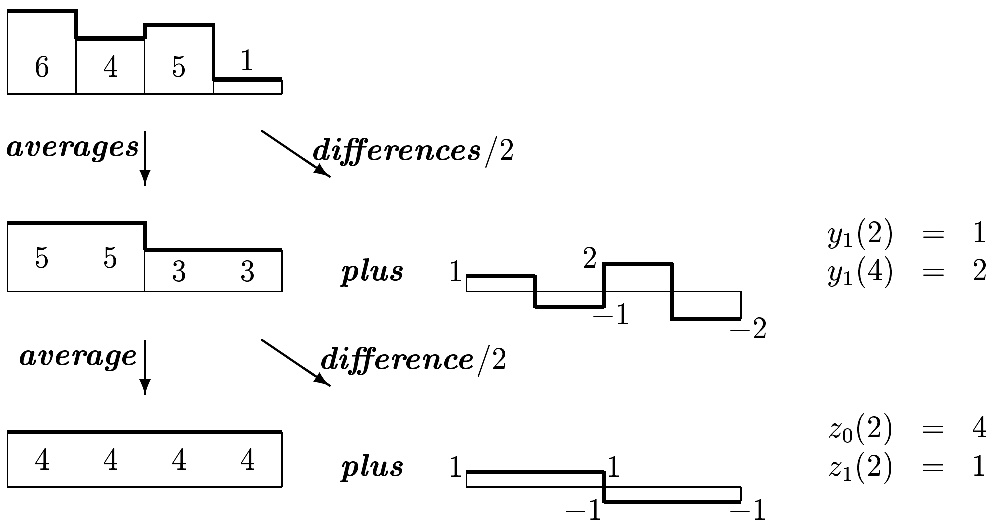
You will ask, why not take averages and differences also of this first highpass output $y_1(2n)$? That is certainly possible. A "wavelet packet" might choose to do it. But the basic wavelet tree assumes that the highpass coefficients are small and not worth the additional effort. They are candidates for compression. For a typical long smooth signal, iteration to four or five scale levels will further decorrelate the samples. Iterations beyond that are generally not worthwhile.

**Summary:** The input is $x = (6, 4, 5, 1)$. The wavelet coefficients are $(4, 1, 1, 2)$:

$$(\text{low-low } z_0,\ \text{high-low } z_1,\ \text{high } y_1) = (4, 1, 1, 2).$$

The special point of the wavelet basis is that you can pick off the highpass details (1 and 2 in $y_1$), before the coarse details in $z_1$ and the overall average in $z_0$. A picture will explain this *multiscale pyramid*:

***Split*** $x = (6, 4, 5, 1)$ ***into averages and waves at small scale and then large scale:***




## 8 Reconstruction by the Synthesis Bank

The averaging-differencing filter (named for Haar) produces two outputs from two successive inputs:

$$\begin{aligned} y_0(2n) &= \frac{1}{2}(x(2n-1) + x(2n)) \\ y_1(2n) &= \frac{1}{2}(x(2n-1) - x(2n)) \end{aligned} \tag{32}$$

It is easy to recover $x(2n-1)$ and $x(2n)$. In fact, the inputs are just sums and differences of these outputs:

$$\begin{aligned} x(2n-1) &= y_0(2n) + y_1(2n) \\ x(2n) &= y_0(2n) - y_1(2n) \end{aligned} \tag{33}$$

Thus the reconstruction step uses the same operations as the analysis step.

Other analysis banks don't have this simple "two at a time" block structure, but the basic principle still applies. We will show how filtering followed by downsampling is inverted by **upsampling followed by filtering**. This sequence of inverse operations (and the notation) is displayed in the following diagram. *This is the synthesis bank:*

$$\begin{array}{l} \text{Lowpass channel: } y_0(2n) \longrightarrow \boxed{\uparrow 2} \text{---} \boxed{F_0} \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad + \text{---} \widehat{x}(n) = x(n-\ell). \\ \text{Highpass channel: } y_1(2n) \longrightarrow \boxed{\uparrow 2} \text{---} \boxed{F_1} \end{array}$$

The goal is to recover the input exactly, when no compression has been applied to the wavelet transform $z(n)$. The synthesis bank does recover $x$ but with $\ell$ delays ($\ell$ depends on the filter bank). Long delays are not desirable, but causal filters cannot avoid some delay. The filter only looks backward.

First, we show how the combination of upsampling and filtering recovers the input to the Haar filter (with one delay!).

$$\begin{array}{l} (\uparrow 2)\, y_0(2n) = \begin{bmatrix} y_0(0) \\ 0 \\ y_0(2) \\ 0 \end{bmatrix} \longrightarrow \boxed{\begin{array}{c}\text{sum} \\ \text{filter } F_0\end{array}} \longrightarrow \begin{bmatrix} y_0(0) \\ y_0(0) \\ y_0(2) \\ y_0(2) \end{bmatrix} \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad + \text{---} \begin{bmatrix} x(-1) \\ x(0) \\ x(1) \\ x(2) \end{bmatrix}. \\ (\uparrow 2)\, y_1(2n) = \begin{bmatrix} y_1(0) \\ 0 \\ y_1(1) \\ 0 \end{bmatrix} \longrightarrow \boxed{\begin{array}{c}\text{difference} \\ \text{filter } F_1\end{array}} \longrightarrow \begin{bmatrix} y_1(0) \\ -y_1(0) \\ y_1(2) \\ -y_1(2) \end{bmatrix} \end{array} \tag{34}$$

At the end we used equations (33) for the sum and the difference, to produce $x$.

The choice of synthesis filters $F_0$ and $F_1$ is directly tied to the analysis filters $H_0$ and $H_1$ (since the two filter banks are inverses). We will write down the rule for $F_0$ and $F_1$, and show that it succeeds (**perfect reconstruction**). $F_0$ comes from *alternating the signs* in $H_1$. This takes the highpass $H_1$ into a lowpass filter $F_0$. Similarly, the highpass $F_1$ comes from alternating the signs of the coefficients in $H_0$. An example will make the point more clearly:

$$\begin{array}{lll} h_0 = \frac{1}{8}(-1, 2, 6, 2, -1) & & f_0 = \frac{1}{2}(1, 2, 1) \\ & & \\ h_1 = \frac{1}{2}(1, -2, 1) & & f_1 = \frac{1}{4}(1, 2, -6, 2, 1). \end{array}$$

The coefficients of $h_1$ and $f_1$ add to zero; a zero-frequency signal (a constant DC signal) is killed by these highpass filters. The coefficients of $h_0$ add to 1 and the coefficients of $f_0$ add to 2. The filters $h_0 = (\frac{1}{2}, \frac{1}{2})$ and $f_0 = (1, 1)$ in the Haar example followed these rules.

We can verify that this analysis-synthesis filter bank gives perfect reconstruction with $\ell = 3$ delays: The output is $x(n-3)$. Suppose the input is a sinusoid $x(n) = e^{in\omega}$ at frequency $\omega$. Then $H_0$ and $H_1$ become multiplications by their response functions $H_0(e^{i\omega})$ and $H_1(e^{i\omega})$. The combination of $(\downarrow 2)$

followed by $(\uparrow 2)$ introduces zeros in the odd-numbered components, which means that the **alias frequency** $\omega + \pi$ appears together with $\omega$:

$$(\uparrow 2)\,(\downarrow 2)\,\{e^{in\omega}\} = \begin{bmatrix} e^{in\omega} \\ 0(\text{odd}\,n) \\ e^{i(n+2)\omega} \\ 0(\text{odd}\,n) \end{bmatrix} = \frac{1}{2}\left[\{e^{in\omega}\} + \{e^{in(\omega+\pi)}\}\right]. \tag{35}$$

The factor $e^{in\pi}$ is $+1$ for even $n$ and $-1$ for odd $n$. This aliased frequency $\omega+\pi$ is appearing in both channels of the filter bank. It has to be cancelled at the last step by $F_0$ and $F_1$. This was the reason behind the alternating signs (between $h_0$ and $f_1$ and between $h_1$ and $f_0$). Those alternations introduce powers of $e^{i\pi}$ in the frequency response functions:

$$\begin{aligned} F_0(e^{i\omega}) &= \frac{1}{2}(1 + 2e^{-i\omega} + e^{-2i\omega}) \\ &= \frac{1}{2}(1 - 2e^{-i(\omega+\pi)} + e^{-2i(\omega+\pi)}) \\ &= 2H_1(e^{i(\omega+\pi)}) = 2H_1(e^{-i\omega}). \end{aligned} \tag{36}$$

Similarly,

$$F_1(e^{i\omega}) = -2H_0(e^{-i(\omega+\pi)}) = -2H_0(e^{-i\omega}) \tag{37}$$

Now follow the pure exponential signal through analysis and synthesis, substituting these expressions (36)– (37) for $F_0$ and $F_1$ when they multiply the signal (the last step in the filter bank):

$$\begin{aligned} H_0(e^{i\omega})e^{in\omega} &\longrightarrow \boxed{\downarrow 2}\!-\!\boxed{\uparrow 2} \longrightarrow \frac{1}{2}\left[H_0(e^{i\omega})e^{in\omega} + H_0(-e^{i\omega})e^{in(\omega+\pi)}\right] \\ &\longrightarrow \boxed{F_0} \longrightarrow H_1(-e^{i\omega})H_0(e^{i\omega})e^{in\omega} + H_1(-e^{i\omega})H_0(-e^{i\omega})e^{in(\omega+\pi)} \end{aligned}$$

$$\begin{aligned} H_1(e^{i\omega})e^{in\omega} &\longrightarrow \boxed{\downarrow 2}\!-\!\boxed{\uparrow 2} \longrightarrow \frac{1}{2}\left[H_1(e^{i\omega})e^{in\omega} + H_1(-e^{i\omega})e^{in(\omega+\pi)}\right] \\ &\longrightarrow \boxed{F_1} \longrightarrow H_0(-e^{i\omega})H_1(e^{i\omega})e^{in\omega} - H_0(-e^{i\omega})H_1(-e^{i\omega})e^{in(\omega+\pi)}. \end{aligned}$$

These are the outputs from the low and high channels. When we add, the alias terms with frequency $\omega + \pi$ cancel out. The choice of $F_0$ and $F_1$ gave “alias cancellation.”

The condition for perfect reconstruction comes from the terms involving $e^{in\omega}$. Those don't cancel! The sum of these terms should be $e^{i(n-\ell)\omega}$, which produces the

**PR Condition:** $$H_1(-e^{i\omega})H_0(e^{i\omega}) - H_0(-e^{i\omega})H_0(e^{i\omega}) = e^{-i\ell\omega}. \tag{38}$$

This must hold for all frequencies $\omega$. If we replace $\omega$ by $\omega + \pi$, the left side of the equation changes sign. Therefore, the right side must change sign, and *the system delay $\ell$ is always odd.*

Actually, the left side of (38) is exactly **the odd part** of the product

$$P(e^{i\omega}) = H_1(-e^{i\omega})H_0(e^{i\omega}) = \frac{1}{2}F_0(e^{i\omega})H_0(e^{i\omega})\,. \tag{39}$$

The PR condition states that *only one odd-numbered coefficient of this lowpass filter is nonzero*. That nonzero coefficient is the $\ell^{\text{th}}$. We now verify that the example satisfies this PR condition:

$$\begin{aligned} P(e^{i\omega}) &= \frac{1}{4}(1 + 2e^{-i\omega} + e^{-2i\omega})\frac{1}{8}(-1 + 2e^{-i\omega} + 6e^{-2i\omega} + 2e^{-3i\omega} - e^{-4i\omega}) \\ &= \frac{1}{32}(-1 + 9e^{-2i\omega} + 16e^{-3i\omega} + 9e^{-4i\omega} - e^{-6i\omega})\,. \end{aligned}$$

The coefficients of the product filter $P$ are $-1, 0, 9, 16, 9, 0, -1$ divided by 32. The middle coefficient is $\frac{16}{32} = \frac{1}{2}$. *The zeros in the other odd-numbered coefficients give perfect reconstruction.* And it is the particular coefficients in $-1, 0, 9, 16,\ 9, 0, -1$ that make $P$ a powerful lowpass filter, because it has a *fourth-order zero* at the top frequency $\omega = \pi$. Factoring this polynomial produces

$$P(e^{i\omega}) = \left(\frac{1 + e^{-i\omega}}{2}\right)^4 (-1 + 4e^{-i\omega} - e^{-2i\omega})\,. \tag{40}$$

Roughly speaking, the power $(1 + e^{-i\omega})^4$ makes $P$ a good lowpass filter. The final factor is needed to produce zeros in the first and fifth coefficients of $P$.

**Remark.** The PR condition applies to the product $P$ and not the separate filters $H_0$ and $H_1$. So any other factorization of $P(e^{i\omega})$ into $H_1(-e^{i\omega})H_0(e^{i\omega})$ gives perfect reconstruction. Here are two other analysis-synthesis pairs that share the same product $P(e^{i\omega})$:

**Biorthogonal** 6/2

$$h_0 = \frac{1}{16}(-1,1,8,8,1,-1) \quad f_0 = (1,1)$$
$$h_1 = \frac{1}{2}(1,-1) \quad f_1 = \frac{1}{8}(-1,1,-8,8,-1,1)$$

**Orthogonal** 4/4

$$\begin{aligned} h_0 &= \frac{1}{8}\left(1+\sqrt{3}, 3+\sqrt{3}, 3-\sqrt{3}, 1-\sqrt{3}\right) \\ f_0 &= \frac{1}{4}\left(1-\sqrt{3}, 3-\sqrt{3}, 3+\sqrt{3}, 1+\sqrt{3}\right) \\ h_1 &= \frac{1}{8}\left(1-\sqrt{3}, -3+\sqrt{3}, 3+\sqrt{3}, -1-\sqrt{3}\right) \\ f_1 &= \frac{1}{4}\left(-1-\sqrt{3}, 3+\sqrt{3}, -3+\sqrt{3}, 1-\sqrt{3}\right) \end{aligned}$$

The second one is *orthogonal* because the synthesis filters are just *transposes* (time-reversed flips) of the analysis filters. When the inverse is the same as the transpose, a matrix or a transform or a filter bank is called orthogonal. If we wrote out the infinite matrix $H_{\text{bank}}$ for this analysis pair, we would discover that $F_{\text{bank}}$ is essentially the transpose. (There will be a factor of 2 and also a shift by 3 diagonals to make the matrix lower triangular and causal again. These shifts are responsible for the $\ell = 3$ system delays.)

The algebra of PR filter banks was developed by Smith-Barnwell and Mintzer in the 1970s. The iteration of those filter banks led to wavelets in the 1980s. The great paper of Ingrid Daubechies [2] gave the theory of orthogonal wavelets, and the first wavelets she constructed came from the coefficients $(1+\sqrt{3}, 3+\sqrt{3}, 3-\sqrt{3}, 1-\sqrt{3})$ given above.

The other filter banks, not orthogonal, are called *biorthogonal.* Synthesis is biorthogonal to analysis. The rows of $H_{\text{bank}}$ are orthogonal to the columns of $F_{\text{bank}}$—except that $(\mathit{row}\, k) \cdot (\mathit{column}\, k) = 1$. This is always true for a matrix and its inverse:

$$HF = I \quad \text{means} \quad (\mathit{row}\, j \text{ of } H) \cdot (\mathit{column}\, k \text{ of } F) = \delta(j-k)\,.$$

The rows of $H$ and columns of $H^{-1}$ are biorthogonal. The analysis and synthesis wavelets that come below from infinite iteration will also be biorthogonal. For these basis functions we often use the shorter word *dual.*

# 9 Scaling Functions and Refinement Equation

May I write down immediately the most important equation in wavelet theory? It is an equation in continuous time, and its solution $\phi(t)$ is the "**scaling function**." This section will describe basic examples, which happen to be spline functions. In later sections the connection to a filter bank and to multiresolution will be developed. More precisely, the connection is to the lowpass operator $(\downarrow 2)H_0$ of filtering and downsampling.

This fundamentally new and fascinating equation is the **refinement equation** or **dilation equation**:

$$\phi(t) = 2\sum_{k=0}^{N} h(k)\phi(2t-k)\,. \tag{41}$$

The factor 2 gives the right side the same integral as the left side, because each $\phi(2t-k)$ has half the area (from compression to $2t$) and $\sum h(k) = 1$. It is the inner factor 2, the one that rescales $t$ to $2t$, that makes this equation so remarkable.

Equation (41) is linear. If $\phi(t)$ is a solution, so is any multiple of $\phi(t)$. The usual normalization is $\int \phi(t)\,dt = 1$. That integral extends over all time, $-\infty < t < \infty$, but we actually find that the solution $\phi(t)$ is zero outside the interval $0 \le t < N$. **The scaling function has compact support.** This localization of the function is one of its most useful properties.

**Example 1.** Suppose $h(0) = h(1) = \frac{1}{2}$ as in the averaging filter. Then the refinement equation for the scaling function $\phi(t)$ is

$$\phi(t) = \phi(2t) + \phi(2t-1)\,. \tag{42}$$

The solution is the unit *box function*, which equals 1 on the interval $0 \le t < 1$ (and elsewhere $\phi(t) = 0$). The function $\phi(2t)$ on the right side of the equation is a *half-box*, reaching only to $t = \frac{1}{2}$. The function $\phi(2t-1)$ is a *shifted half-box*, reaching from $t = \frac{1}{2}$ to $t = 1$. Together, the two half-boxes make up the whole box, and the refinement equation (42) is satisfied.

Notice in this example the two key operations of classical wavelet theory:

**Dilation** of the time scale by 2 or $2^j$:
The graph of $\phi(2t)$ is compressed by 2.

**Translation** of the time scale by 1 or $k$:
The graph of $\phi(t-k)$ is shifted by $k$.

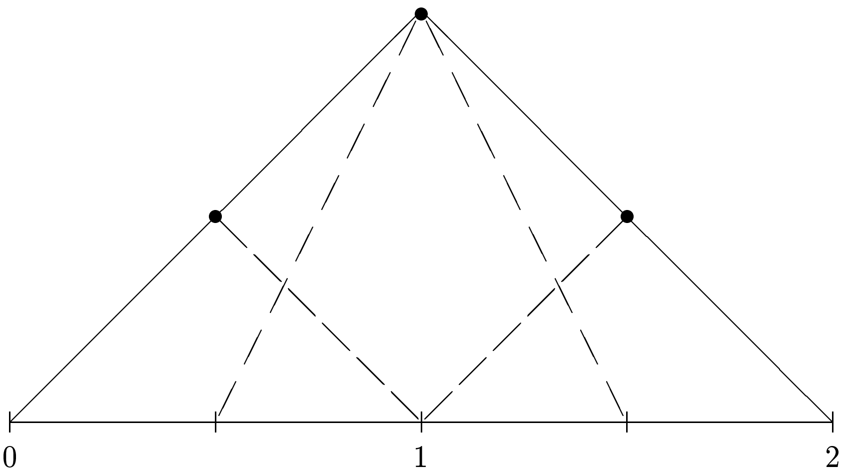



Figure 5: Three half-length hats combine into a full-length hat.

The combination of those operations, from $t$ to $2^j t$ to $2^j t-k$, will later produce a family of wavelets $w(2^j t-k)$ from a single wavelet $w(t)$.

**Example 2.** Suppose $h(0) = \frac{1}{4}$, $h(1) = \frac{1}{2}$, $h(2) = \frac{1}{4}$. The averaging filter has been repeated. Whenever we multiply filters (Toeplitz matrices), we are convolving their coefficients. Squaring a filter means convolving the vector of coefficients with itself, and this is verified by $(\frac{1}{2}, \frac{1}{2}) * (\frac{1}{2}, \frac{1}{2}) = (\frac{1}{4}, \frac{1}{2}, \frac{1}{4})$. This convolution is just a multiplication of a polynomial times itself:

$$\left(\frac{1}{2} + \frac{1}{2}z^{-1}\right)^2 = \frac{1}{4} + \frac{1}{2}z^{-1} + \frac{1}{4}z^{-2}.$$

The solution $\phi(t)$, which is the scaling function for $h = (\frac{1}{4}, \frac{1}{2}, \frac{1}{4})$, is the old solution convolved with itself:

$$(\text{box function}) * (\text{box function}) = \textbf{hat function}.$$

The box is piecewise constant. The hat is piecewise linear, equal to $t$ and $2-t$ on the intervals $[0,1]$ and $[1,2]$. Figure 5 shows how three compressed and shifted hat functions combine into the full hat function. The picture verifies the refinement equation $\phi(t) = \frac{1}{2}\phi(2t) + \phi(2t-1) + \frac{1}{2}\phi(2t-2)$.

**Example 3.** (*Splines of degree* $p-1$). Every time an extra averaging filter is introduced, we are convolving $\phi(t)$ with the box function. This makes the filter coefficients and the scaling function smoother; the function has one additional derivative. From a product of $p$ averaging filters, the frequency response is $(\frac{1+e^{-i\omega}}{2})^p$. Its $k^{\text{th}}$ coefficient will be the binomial number $\binom{p}{k}$ = "$p$ choose $k$" divided by $2^p$.

The scaling function for this filter is a **B-spline** on the interval $[0, p]$. This spline $\phi(t)$ is formed from polynomials of degree $p-1$ that are joined together smoothly ($p-2$ continuous derivatives) at the nodes $t = 0, 1, \ldots, p$. This is the convolution of $p$ box functions:

$$\text{Filter:} \qquad \left(\frac{1}{2}, \frac{1}{2}\right) * \left(\frac{1}{2}, \frac{1}{2}\right) * \cdots * \left(\frac{1}{2}, \frac{1}{2}\right) = \frac{1}{2^p}\left(1, p, \frac{p(p-1)}{2}, \ldots, p, 1\right)$$

$$\text{Function:} \quad (\text{box}) * (\text{box}) * \cdots * (\text{box}) = \text{B-spline of degree } p-1.$$

These are essentially all the examples in which the scaling function $\phi(t)$ has a simple form. If we take $h(0) = \frac{2}{3}$ and $h(1) = \frac{1}{3}$, we get a very singular and unbounded solution, with infinite energy. It blows up at $t = 0$, where (41) becomes $\phi(0) = \frac{4}{3}\phi(0)$, but it is *not* a standard delta function. After convolving with $(\frac{1}{2}, \frac{1}{2})$, the coefficients $h = (\frac{1}{3}, \frac{1}{2}, \frac{1}{6})$ are much better. The scaling function is smoother and in $L^2$. But still we have no elementary expression for $\phi(t)$ when the refinement equation has these coefficients. All we can do is to compute $\phi(t)$ at more and more binary points $t = m/2^j$, or find its Fourier transform (see below). We can graph the function as accurately as desired, but we don't know a simple formula.

*Now we begin the connection with a filter bank.* The output from a lowpass filter (convolution with $h$), followed by downsampling ($n$ changes to $2n$), is but it is *not* a standard delta function. $\sum h(k)x(2n-k)$. Suppose this output goes back in as input. (**This is iteration**.) The process could be studied in discrete time, but continuous time is more interesting and revealing. So the original input is a box function, and we get $\phi^{(i+1)}(t)$ from $\phi^{(i)}(t)$:

$$\textit{Filter and rescale:} \quad \phi^{(i+1)}(t) = 2\sum_{k=0}^{N} h(k)\phi^{(i)}(2t-k)\,. \tag{43}$$

This is the **cascade algorithm**. The lowpass filter is "cascaded." If this iteration converges, so that $\phi^{(i)}(t)$ approaches a limit $\phi(t)$ as $i \to \infty$, then that limit function satisfies the refinement equation (41).

In this sense the scaling function is a fixed point (or eigenfunction with $\lambda = 1$) of the filtering-downsampling operation $2(\downarrow 2)H_0$. The normalization by 2 is necessary because time is compressed by 2 at each iteration. Starting from the box function $\phi^{(0)}(t)$ on $[0, 1]$, the function $\phi^{(i)}(t)$ will be piecewise constant on intervals of length $2^{-i}$. Thus continuous time gives a natural way to account for the downsampling ($n \to 2n$ and $t \to 2t$). This time compression is clearer for a function than for a vector.

When we apply the iteration (43) for the averaging filter $h = \left(\frac{1}{2}, \frac{1}{2}\right)$, we find $\phi^{(1)} \equiv \phi^{(0)}$. The convergence is immediate because the box $\phi^{(0)}$ is the correct scaling function. When we use $h = \left(\frac{1}{4}, \frac{1}{2}, \frac{1}{4}\right)$, every iteration produces a "staircase up and down" with steps of $2^{-i}$. The function $\phi^{(i)}(t)$ is a piecewise constant hat, supported on the interval $[0, 2-2^{-i}]$. The limit as $i \to \infty$ is the correct piecewise linear hat function $\phi(t)$.

Similarly the B-spline comes from the cascade algorithm. But for other coefficients $h(k)$, always adding to 1, there may or may not be a limit function. The functions $\phi^{(i)}(t)$ can converge to a limit $\phi(t)$, or they can explode. If you try the bad filter $h = (\frac{2}{3}, \frac{1}{3})$, you find that $\phi^{(i)}(0) = (\frac{4}{3})^i$. And the iterations explode at infinitely many other points too (please try this example). A natural question is to find the requirement for the cascade algorithm to converge:

**1**. For which filters $h(k)$ does $\phi^{(i)}(t)$ approach a limit $\phi(t)$ in $L^2$?

A minimal condition is that $\sum(-1)^k h(k) = 0$. In the frequency domain, this is the response $H(e^{i\omega}) = \sum h(k)e^{-ik\omega}$ at the highest frequency $\omega = \pi$. The filter must have a "**zero at** $\pi$" for convergence to have a chance (and this eliminates many equiripple lowpass filters). The precise Condition E is on the eigenvalues of a matrix (as we expect in all iterations!):

**Condition E:** The matrix $T = 2(\downarrow 2)HH^T$ must have all eigenvalues $|\lambda| < 1$, except for a simple eigenvalue $\lambda = 1$.

This is a familiar condition for convergence of the powers $T^i$ of a matrix to a rank one matrix (as in the "power method" to compute eigenvalues).

You might like to see three rows of $T$ for the hat filter $h = \left(\frac{1}{4}, \frac{1}{2}, \frac{1}{4}\right)$:

$$2(\downarrow 2)HH^T = T = \frac{1}{8}\begin{bmatrix} 1 & 4 & 6 & \begin{pmatrix} 4 & 1 & \\ 4 & 6 & 4 \\ & 1 & 4 \end{pmatrix} & & & \\ & & 1 & & 1 & & \\ & & & & 6 & 4 & 1 \end{bmatrix}.$$

Note the double shift between rows which comes from $(\downarrow 2)$. The coefficients $1, 4, 6, 4, 1$ come from $HH^T$. In polynomial terms, we are multiplying $H(z) = \frac{1}{4}(1+2z^{-1}+z^{-2})$ by its time-reversed flip $H(z^{-1}) = \frac{1}{4}(1+2t+z^2)$. The matrix product $HH^T$ is an *autocorrelation* of the filter.

We have highlighted the crucial 3 by 3 matrix inside $T$. Its three eigenvalues are $\lambda = 1, \frac{1}{2}, \frac{1}{4}$. Because of the double shift, the rest of $T$ will contribute $\lambda = \frac{1}{8}$ (twice) and $\lambda = 0$ (infinitely often). No eigenvalue exceeds $\lambda = 1$ and Condition E is clearly satisfied.

A very short MATLAB program will compute $T$ from the coefficients $h(k)$ and find the eigenvalues. There will be $\lambda = 1, \frac{1}{2}, \frac{1}{4}, \ldots, \frac{1}{2^{2p-1}}$, when the filter response $H(e^{i\omega})$ has a $p^{\text{th}}$ order zero at $\omega = \pi$. It is the *largest other eigenvalue* $|\lambda_{\max}|$ of $T$ that decides the properties of $\phi(t)$. That eigenvalue $|\lambda_{\max}|$ must be smaller than 1. This is Condition E above for convergence of the cascade algorithm.

We mention one more important fact about $|\lambda_{\max}|$, without giving the proof here. This number determines the **smoothness** of the scaling function (and later also of the wavelet). A smaller number means a smoother function. We measure smoothness by the **number of derivatives** of $\phi(t)$ that have finite energy (belong to the space $L^2$). This number $s$ need not be an integer. Thus the box function is in $L^2$, and its first derivative (delta functions $\delta(t) - \delta(t-1)$) is certainly not in $L^2$. But at a jump, all derivatives of order $s < \frac{1}{2}$ do have finite energy:

$$\text{Energy} = \int_{-\infty}^{\infty} \left| \frac{d^s \phi}{dt^s} \right|^2 dt = \int_{-\infty}^{\infty} \left| (i\omega)^s \widehat{\phi}(\omega) \right|^2 d\omega\,.$$

The Fourier transform of the box function has $|\widehat{\phi}(\omega)| = |\operatorname{sinc} \omega| \le \mathit{const}\,/|\omega|$. Then for every $s < \frac{1}{2}$,

$$\text{Energy in } s^{\text{th}} \text{ derivative} < \mathit{const} + \mathit{const} \int_1^{\infty} |\omega|^{2s-2}\, d\omega < \infty\,.$$

The hat function is one order smoother than the box function (one extra Haar factor). So it has derivatives with finite energy up to order $s = \frac{3}{2}$. The general formula for the smoothness of $\phi(t)$ is remarkably neat: All derivatives have finite energy, up to the order

$$s_{\max} = -\log_4 |\lambda|_{\max}\,. \tag{44}$$

For the hat function and its filter $h = (\frac{1}{4}, \frac{1}{2}, \frac{1}{4})$, the double eigenvalue $\lambda = \frac{1}{8}$ was mentioned for the matrix $T$ above. This is $\lambda_{\max}$ and the formula (44) correctly finds the smoothness $s = -\log_4(\frac{1}{8}) = \frac{3}{2}$.

We can explicitly solve the refinement equation in the frequency domain, for each separate $\omega$. The solution $\widehat{\phi}(\omega)$ is an infinite product. The equation connects the scales $t$ and $2t$, so its Fourier transform connects the frequencies $\omega$ and $\omega/2$:

$$\text{The transform of } \phi(2t-k) \text{ is } \textstyle\int \phi(2t-k) e^{-i\omega t}\, dt = \frac{1}{2} e^{-i\omega k/2} \widehat{\phi}(\frac{\omega}{2}).$$

Then the refinement equation $\phi(t) = 2\sum h(k)\phi(2t-k)$ transforms to

$$\widehat{\phi}(\omega) = \left(\sum h(k)e^{-i\omega k/2}\right)\widehat{\phi}\left(\frac{\omega}{2}\right) = H\left(\frac{\omega}{2}\right)\widehat{\phi}\left(\frac{\omega}{2}\right). \tag{45}$$

By iteration (which is the cascade algorithm!), the next step gives $\widehat{\phi}(\omega) = H(\frac{\omega}{2})H(\frac{\omega}{4})\widehat{\phi}(\frac{\omega}{4})$. Infinite iteration gives the *infinite product formula*

$$\widehat{\phi}(\omega) = \prod_{j=1}^{\infty} H\left(\frac{\omega}{2^j}\right) \qquad (\text{since } \widehat{\phi}(0) = 1). \tag{46}$$

This product gives a finite answer for each frequency $\omega$, so we always have a formula for the Fourier transform of $\phi(t)$. But if $|\lambda_{\max}| > 1$ and Condition E fails to hold, the inverse transform from $\widehat{\phi}(\omega)$ to $\phi(t)$ will produce a wild function with smoothness $s < 0$ and infinite energy.

In the case of box functions and splines, this infinite product simplifies to sinc functions and their powers:

$$\widehat{\phi}_{\text{box}}(\omega) = \left(\frac{1-e^{-i\omega}}{i\omega}\right) \quad \text{and} \quad \widehat{\phi}_{\text{spline}}(\omega) = \left(\frac{1-e^{-i\omega}}{i\omega}\right)^p. \tag{47}$$

In an interesting recent paper, Blu and Unser [16] use this formula to define splines for *noninteger p*. These *fractional splines* are not polynomials, and their support is not compact, but they interpolate in a natural way between the piecewise polynomial splines (where $p$ is an integer).

The Daubechies 4-tap filter produces a scaling function with $s_{\max} = 1$. The "lazy filter" with only one coefficient $h(0) = 1$ produces $\phi(t) =$ Dirac delta function. You can verify directly that $\delta(t) = 2\delta(2t)$, so the refinement equation is satisfied. In the frequency domain, $H(\omega) \equiv 1$ so that $\widehat{\phi}(\omega) \equiv 1$. The smoothness of the delta function is one order lower ($s = -\frac{1}{2}$) than the step function: $H = I$ and $T = 2I$ and $|\lambda_{\max}| = 2$ and $s = -\log_4 2 = -\frac{1}{2}$.

# 10 Multiresolution and the Spaces $A$

Our first explanation of the refinement equation $\phi(t) = 2\sum h(k)\phi(2t-k)$ was by analogy. This is a continuous time version of filtering (by $h$) and downsampling ($t$ to $2t$). When we iterate the process, we are executing the cascade algorithm. When the initial $\phi^{(0)}(t)$ is the box function, all iterates $\phi^{(i)}(t)$ are piecewise constant and we are really copying a discrete filter bank. We are iterating forever! In this description, the refinement equation gives the limit of an infinite filter tree.

Now we give a different and deeper explanation of the same equation. We forget about filters and start with functions. Historically, this is how the refinement equation appeared. (Later it was realized that there was a close connection to the filter banks that had been developed in signal processing, and the two histories were joined. Wavelets and lowpass filters come from highpass and lowpass filters. An $M$-channel filter bank would lead to one scaling function, with $M$'s instead of 2's in the refinement equation, and $M-1$ wavelets.) But the first appearance of $\phi(t)$ was in the idea of **multiresolution**.

Multiresolution looks at different time scales, $t$ and $2t$ and every $2^j t$. These correspond to octaves in frequency. When a problem splits naturally into several time scales or frequency scales, this indicates that the wavelet approach (multiresolution) may be useful. Wavelets are another tool in time-frequency analysis comparable to the windowed "short-time" Fourier analysis that transforms $f(t)$ to a function of the window position $t$ and the frequency $\omega$. For wavelets, it is a *time-scale analysis*, with $j$ for the scale level and $k$ for the position. The wavelet coefficients produce a "scalogram" instead of a spectrogram.

Start with Mallat's basic rules for a multiresolution [10]. There is a space $V_0$, containing all combinations of the basis functions $\phi(t-k)$. This shift-invariant basis produces a shift-invariant subspace of functions:

Property 1: $f(t)$ is in $V_0$ if and only if $f(t-1)$ is in $V_0$.

Now compress each basis function, rescaling the time from $t$ to $2t$. The functions $\phi(2t-k)$ are a new basis for a new space $V_1$. This space is again shift-invariant (with shifts of $\frac{1}{2}$). And between $V_0$ and $V_1$ there is an automatic scale-invariance:

Property 2: $f(t)$ is in $V_0$ if and only if $f(2t)$ is in $V_1$.

Now we come to the crucial third requirement. **We insist that the subspace $V_1$ contains $V_0$.** This requires in particular that the subspace $V_1$ contains the all-important function $\phi(t)$ from $V_0$. In other words, $\phi(t)$ must be a combination of the functions $\phi(2t-k)$ that form a basis for $V_1$:

Property 3. (*Refinement Equation*) $\phi(t) = \sum c_k \phi(2t-k)$.

When this holds, all other basis functions $\phi(t-k)$ at the coarse scale will be combinations of the basis functions $\phi(2t-k)$ at the fine scale. In other words $V_0 \subset V_1$.

I hope that every reader recognizes the connection to filters. *Those coefficients $c_k$ in Property 3 will be exactly the filter coefficients $2h(k)$.* Previously we constructed $\phi(t)$ from the refinement equation. Now we are starting from $V_0$ and $V_1$ and their bases, and the requirement that $V_1$ contains $V_0$ gives the refinement equation.

The extension to a whole "scale of subspaces" is immediate. The space $V_j$ consists of all combinations of $\phi(2^j t)$ and its shifts $\phi(2^j t-k)$. When we replace $t$ by $2^j t$ in the refinement equation (Property 3), we discover that $V_j$ is contained in $V_{j+1}$. Thus the sequence of subspaces is "nested" and $V_0 \subset V_1 \subset V_2 \subset \cdots$.

There is one optional property, which was often included in the early papers but is not included now. That is the *orthogonality* of the basis functions $\phi(t-k)$. The Daubechies functions possess this property, but the spline functions do not. (The only exception is the box function = spline of degree zero = Haar scaling function = first Daubechies function.) In the orthogonal case, the coefficients $c_k$ in the refinement equation come from an *orthogonal filter bank*. Certainly orthogonality is a desirable property and these bases are very frequently used — but then $\phi(t)$ cannot have the symmetry that is desirable in image processing.

To complete the definition of a multiresolution, we should add a requirement on the basis functions. We don't insist on orthogonality, but we do require a stable basis (also known as a Riesz basis). We think of $V_0$ as a subspace of $L^2$ (the functions with finite energy $\|f\|^2 = \int |f(t)|^2\, dt$). The basis functions $\phi(t-k)$ must be "uniformly" independent:

Property 4. (*Stable Basis*) $\| \sum_{-\infty}^{\infty} a_k \phi(t-k)\|^2 \geq c \sum a_k^2$ with $c > 0$.

This turns out, beautifully and happily, to be equivalent to Condition E on the convergence of the cascade algorithm. An earlier equivalent condition on $\phi(t)$ itself was discovered by Cohen. We can also establish that the spaces $V_j$ approximate every finite energy function $f(t)$: the projection onto $V_j$ converges to $f(t)$ as $j \to \infty$. And one more equivalent condition makes this a very satisfactory theory: the scaling functions $\phi(t-k)$ together with the normalized wavelets $2^{j/2} w(2^j t - k)$ at all scales $j = 0, 1, 2, \ldots$ also form a stable basis.

We now have to describe these wavelets. For simplicity we do this in the orthogonal case. First a remark about "multiwavelets" and a word about the order of approximation to $f(t)$ by the subspaces $V_j$.

**Remark.** Instead of the translates $\phi(t-k)$ of one function, the basis for $V_0$ might consist of the translates $\phi_1(t-k), \ldots, \phi_r(t-k)$ of $r$ different functions.

All of the properties 1–4 are still desired. The refinement equation in Property 3 now takes a vector form, with $r$ by $r$ matrix coefficients $c_k$: $V_0$ is a subspace of $V_1$ when all the functions $\phi_1(t), \ldots, \phi_r(t)$ are combinations of the basis functions $\phi_1(2t-k), \ldots, \phi_r(2t-k)$:

$$\begin{bmatrix} \phi_1(t) \\ \vdots \\ \phi_r(t) \end{bmatrix} = \sum c_k \begin{bmatrix} \phi_1(2t-k) \\ \vdots \\ \phi_r(2t-k) \end{bmatrix}.$$

The corresponding lowpass filter becomes a "multifilter," with vectors of $r$ samples as inputs and $r$ by $r$ matrices in place of the numbers $h(0), \ldots, h(N)$. The processing is more delicate but multifilters have been successful in denoising [15].

The new feature of multiwavelets is that the basis functions can be both symmetric and orthogonal (with short support). A subclass of "balanced" multifilters needing no preprocessing of inputs has been identified by Vetterli-LeBrun [9] and by Selesnick [11].

# 11 Polynomial Reproduction and Accuracy of Approximation

The requirements for a good basis are easy to state, and important:

**1**. *Fast computation* (wavelets are quite fast)

**2**. *Accurate approximation* (by relatively few basis functions).

We now discuss this question of accuracy. When $f(t)$ is a smooth function, we ask how well it is approximated by combining the translates of $\phi(t)$. Why not give the answer first:

$$\left\| f(t) - \sum_{k=-\infty}^{\infty} a_{jk}\phi(2^j t - k) \right\| \leq C 2^{-jp} \left\| f^{(p)}(t) \right\|. \tag{48}$$

This says that with time steps of length $h = 2^{-j}$, the degree of approximation is $h^p$. This number $p$ is still the *multiplicity of the zero* of $H(e^{i\omega})$ at $\omega = \pi$.

This error estimate (48) is completely typical of numerical analysis. The error is *zero* when $f(t)$ is a polynomial of degree $p-1$. (We test Simpson's Rule and every quadrature rule on polynomials.) The Taylor series for $f(t)$ starts with such a polynomial (perfect approximation). Then the remainder term involves the $p^{\text{th}}$ derivative $f^{(p)}(t)$ and the power $h^p$. This leads to the error estimate (48). So the degree of the first polynomial $t^p$ that is not reproduced by the space $V_0$ determines the crucial exponent $p$.

The box function has $p = 1$. The error is $O(h)$ in piecewise constant approximation. The hat function has $p = 2$ because $H(e^{i\omega}) = (\frac{1+e^{-i\omega}}{2})^2$. The error is $O(h^2)$ in piecewise linear approximation. The 4-tap Daubechies filter also has $p = 2$, and gives an error that is $O(h^2)$. Daubechies created a whole family with $p = 2, 3, 4, \ldots$ of orthogonal filters and scaling functions and wavelets. They have $2p$ filter coefficients and approximation errors $O(h^p)$. They are zero outside the interval from $t = 0$ to $t = 2p - 1$.

We can briefly explain how the order $p$ of the zero at $\omega = \pi$ turns out to be decisive. **That multiple zero guarantees that all the polynomials $1, t, t^2, \ldots, t^{p-1}$ can be exactly produced as combinations of** $\phi(t-k)$. In the frequency domain, the requirement on $\widehat{\phi}(\omega)$ for this polynomial reproduction is known as the Strang-Fix condition:

$$\frac{d^j\widehat{\phi}}{d\omega^j}(2\pi n) = 0 \qquad \text{for } n \neq 0 \text{ and } j < p\,. \tag{49}$$

Thus $\widehat{\phi}(\omega)$ at the particular frequencies $\omega = 2\pi n$ is the key.

Formula (46) for $\widehat{\phi}(\omega)$ is an infinite product of $H(\omega/2^j)$. At frequency $\omega = 2\pi n$, we write $n = 2^\ell q$ with $q$ odd. Then the factor in the infinite product with $j = \ell + 1$ is $H(2\pi n/2^{\ell+1}) = H(q\pi)$. By periodicity this is $H(\pi)$. A $p^{\text{th}}$ order zero of $\widehat{\phi}$ lies at $\omega = 2\pi n$. Thus the Strang-Fix condition on $\widehat{\phi}$ is equivalent to Condition $\mathrm{A}_p$ on the filter:

**Condition $\mathbf{A}_p$:** $H(e^{i\omega})$ has a $p^{\text{th}}$ order zero at $\omega = \pi$.

For the filter coefficients, a zero at $\pi$ means that $\sum(-1)^n h(n) = 0$. A zero of order $p$ translates into $\sum(-1)^n n^k h(n) = 0$ for $k = 0, \ldots, p-1$.

It is remarkable to see in Figure 6 how combinations of $\phi(t-k)$ produce a straight line, when $\phi(t)$ is the highly irregular Daubechies scaling function coming from $h = \frac{1}{8}(1+\sqrt{3}, 3+\sqrt{3}, 3-\sqrt{3}, 1-\sqrt{3})$. The response $H(e^{i\omega})$ has a *double zero* at $\omega = \pi$. The eigenvalues of $T = 2(\downarrow 2)HH^T$ are $\lambda = 1, \frac{1}{2}, \frac{1}{4}, \frac{1}{4}, \frac{1}{8}$, which includes the $1, \frac{1}{2}, \frac{1}{4}, \frac{1}{8}$ that always come with $p = 2$. The smoothness $s = 1$ of the $D4$ scaling function is governed by that extra $\frac{1}{4} = \lambda_{\max}$.

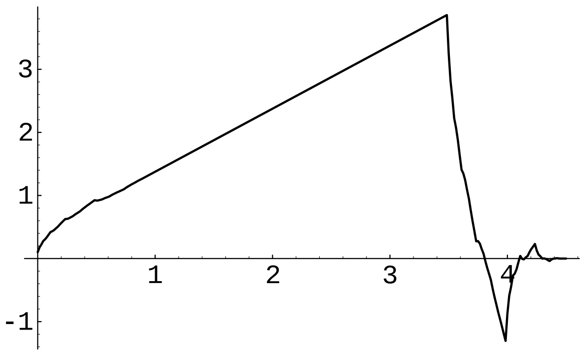



Figure 6: With $p = 2$, the D4 scaling functions $\sum k\phi(t-k)$ produces a straight line.

# 12 Multiresolution and Wavelets

When $j$ increases by 1, the mesh size $h = 2^{-j}$ is multiplied by $\frac{1}{2}$. The error of approximation by scaling functions is reduced by $(\frac{1}{2})^p$. There are twice as many functions available when approximation at scale $j$ uses the translates of a compressed $\phi(2t)$:

**The approximation space $V_j$ contains all combinations of the shifted and compressed scaling functions $\phi(2^jt - k)$, $-\infty < k < \infty$.**

The special feature of these spaces is that $V_1$ contains $V_0$ (and every $V_{j+1}$ contains $V_j$). This comes directly from $\phi(t) = 2\sum h(k)\phi(2t-k)$. *The right side of the refinement equation is a function in* $V_1$, at scale $j = 1$. Therefore the left side $\phi(t)$ lies in $V_1$. So do all its translates, and all their linear combinations. Thus the whole of $V_0$ lies in $V_1$.

Now we look at the functions that are in $V_1$ but *not in* $V_0$. This will lead us to the *wavelet space* $W_0$. The sum of the subspaces $V_0 + W_0$ will equal $V_1$.

The increasing sequence of spaces $V_0 \subset V_1 \subset V_2 \subset \cdots$ leads us to a **multiresolution** of any finite-energy function $f(t)$. Suppose first that the functions $\phi(t-k)$ are orthogonal. Let $f_j(t)$ be the perpendicular projection of $f(t)$ onto the space $V_j$. Thus $f_j(t)$ is the nearest function to $f(t)$ in $V_j$. We will write $f(t)$ as a telescoping sum

$$f(t) = f_0(t) + [f_1(t) - f_0(t)] + [f_2(t) - f_1(t)] + \cdots . \tag{50}$$

This is the sum of a lowest-level "coarse function" $f_0(t)$ and an infinite series of "fine details" $d_j(t) = f_{j+1}(t) - f_j(t)$. Those details will lie in the wavelet subspaces $W_j$. So the splitting of one function $f(t)$ indicates the splitting of the whole space of all $L^2$ functions:

$$L^2 = V_0 + W_0 + W_1 + \cdots \tag{51}$$

Notice that each detail $d_j(t)$ also lies in the next space $V_{j+1}$ (because $f_{j+1}$ and $f_j$ are both in $V_{j+1}$). Thus $W_j$ is in $V_{j+1}$. Furthermore $d_j(t)$ is orthogonal to $V_j$, by simple reasoning:

$f(t) - f_j(t)$ is orthogonal to $V_j$ because $f_j$ is the projection

$f(t) - f_{j+1}(t)$ is similarly orthogonal to $V_{j+1}$ and therefore to $V_j$.

Subtraction shows that the detail $d_j(t) = f_{j+1}(t) - f_j(t)$ is orthogonal to $V_j$.

We assign the name $W_j$ to the *wavelet subspace* containing all those "detail functions" in $V_{j+1}$ that are orthogonal to $V_j$. Then the finer space $V_{j+1}$ is a sum of coarse functions $f_j(t)$ in $V_j$ and details $d_j(t)$ in $W_j$:

$$\begin{array}{ll} \text{For functions:} & f_j(t) + d_j(t) = f_{j+1}(t) \\ \text{For subspaces:} & V_j + W_j = V_{j+1} \end{array}$$

The function $f(t)$ is a combination of the compressed translates $\phi(2^j t - k)$, which are a basis for $V_j$. We want the detail $d_j(t)$ to be a combination of compressed translates $w(2^j t - k)$, which are a basis for $W_j$. **The fundamental detail function $w(t)$ will be the wavelet.**

The wavelet has a neat construction using the highpass coefficients $h_1(k)$:

$$\textbf{Wavelet Equation:} \qquad w(t) = 2\sum h_1(k)\phi(2t-k)\,. \tag{52}$$

The right side is certainly in $V_1$. To prove that $w(t)$ is orthogonal to $\phi(t)$ and its translates, we use the relation $h_1(k) = (-1)^k h_0(N-k)$ between the lowpass and highpass coefficients. We omit this verification here, and turn instead to the most basic example.

The *Haar wavelet* is $w(t) = \phi(2t) - \phi(2t-1) =$ up-and-down box function. It is a *difference* of narrow boxes. Its integral is zero. (Always $\int w(t)\,dt = 0$ because $\sum h_1(k) = 0$.) Furthermore the Haar wavelets $w(2^j t - k)$, compressed and shifted versions of the square wave $w(t)$, are *all* orthogonal to each other and to the box function $\phi(t)$. This comes from the orthogonality of the Haar filter bank.

A biorthogonal filter bank will have four filters $H_0, F_0, H_1, F_1$. Then $H_0$ and $F_0$ yield two different scaling functions $\tilde{\phi}(t)$ and $\phi(t)$, in analysis and synthesis. Similarly $H_1$ and $F_1$ yield analysis wavelets $\tilde{w}(t)$ and synthesis wavelets $w(t)$. The orthogonality property becomes *biorthogonality* between analysis and synthesis:

$$\int \phi(t)\tilde{w}(2^j t - k)\, dt = \int \tilde{\phi}(t) w(2^j t - k)\, dt = 0\,.$$

# 13 Good Basis Functions

This section returns to one of the fundamental ideas of linear algebra — a basis. It often happens that one basis is more suitable than another for a specific application like compression. The whole idea of a *transform* (this paper concentrates on the Fourier transform and wavelet transform) is exactly a change of basis. Each term in a Fourier series or a wavelet series is a basis vector, a sinusoid or a wavelet, times its coefficient. A change of basis gives a *new representation of the same function.*

Remember what it means for the vectors $w_1, w_2, \ldots, w_n$ to be a basis for an $n$-dimensional space $\mathbf{R}^n$:

**1**. The $w$'s are linearly independent.

**2**. The $n \times n$ matrix $W$ with these column vectors is invertible.

**3**. Every vector $x$ in $\mathbf{R}^n$ can be written in exactly one way as a combination of the $w$'s:

$$\mathbf{x} = c_1 w_1 + c_2 w_2 + \cdots + c_n w_n\,.$$

Here is the key point: those coefficients $c_1, \cdots, c_n$ completely describe the vector. In the original basis (standard basis), the coefficients are just the samples $x(1), \ldots, x(n)$. In the new basis of $w$'s, the same $x$ is described by $c_1, \ldots, c_n$. It takes $n$ numbers to describe each vector and it *also* takes a choice of basis:

$$x = \begin{bmatrix} x(1) \\ x(2) \\ x(3) \\ x(4) \end{bmatrix}_{\text{standard basis}} \qquad \text{and} \qquad x = \begin{bmatrix} z_0(2) \\ z_1(2) \\ y_1(2) \\ y_1(4) \end{bmatrix}_{\text{Haar basis}}\,.$$

A basis is a set of axes for $\mathbf{R}^n$. The coordinates $c_1, \ldots, c_n$ tell how far to go along each axis. The axes are at right angles when the basis vectors $w_1, \ldots, w_n$ are orthogonal.

*The Haar basis is orthogonal.* This is a valuable property, shared by the Fourier basis. For the vector $x = (6, 4, 5, 1)$ with four samples, we need four Haar basis vectors. Notice their orthogonality (inner products are zero).

$$w_1 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \qquad w_2 = \begin{bmatrix} 1 \\ 1 \\ -1 \\ -1 \end{bmatrix} \qquad w_3 = \begin{bmatrix} 1 \\ -1 \\ 0 \\ 0 \end{bmatrix} \qquad w_4 = \begin{bmatrix} 0 \\ 0 \\ 1 \\ -1 \end{bmatrix}.$$

That first basis vector is not actually a wavelet, it is the very useful flat vector of all ones. It represents a constant signal (the DC component). Its coefficient $c_1$ is the overall average of the samples this low-low coefficient $c_1$ was $\frac{1}{4}(6 + 4 + 5 + 1) = 4$ for our example vector.

*The Haar wavelet representation of the signal is simply $x = Wc$.* The input $x$ is given in the standard basis of samples. The basis vectors $w_1, \dots, w_4$ go into the columns of $W$. They are multiplied by the coefficients $c_1, \dots, c_4$ (this is how matrix multiplication works!). The matrix-vector product $Wc$ is exactly $c_1 w_1 + c_2 w_2 + c_3 w_3 + c_4 w_4$. Here are the numbers in $x = Wc$:

$$\begin{bmatrix} 6 \\ 4 \\ 5 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & -1 & 0 \\ 1 & -1 & 0 & 1 \\ 1 & -1 & 0 & -1 \end{bmatrix} \begin{bmatrix} 4 \\ 1 \\ 1 \\ 2 \end{bmatrix}. \tag{53}$$

*Those coefficients $c = (4, 1, 1, 2)$ are $W^{-1}x$.* This is what the analysis filter bank must do; a change of basis involves the inverse of the basis matrix. The analysis step computes coefficients $c = W^{-1}x$, and the synthesis step multiplies coefficients times basis vectors to reconstruct $x = Wc$.

The point of wavelets is that both steps, analysis and synthesis, are executed quickly by filtering and subsampling. Let me extend this point to repeat a more general comment on good bases. Orthogonality gives a simple relation between the transform and the inverse transform (they are "transposes" of each other). But I regard two other properties of a basis as more important in practice than orthogonality:

**Speed:** The coefficients $c_1, \dots, c_n$ are fast to compute.

**Accuracy:** A small number of basis vectors and their coefficients can represent the signal very accurately.

For the Fourier and wavelet bases, the speed comes from the FFT and FWT: Fast Fourier Transform and Fast Wavelet Transform. The FWT is exactly the filter bank tree that we just illustrated for the Haar wavelets.

The accuracy depends on the signal! We want to choose a basis appropriate for the class of signals we expect. (It is usually too expensive to choose a basis adapted to each individual signal.) Here is a rough guideline:

For *smooth* signals with no jumps, the *Fourier basis* is hard to beat.

For *piecewise smooth* signals with jumps, a *wavelet basis* can be better.

The clearest example is a step function. The Fourier basis suffers from the *Gibbs phenomenon*: oscillations (ringing) near the jump, and very slow $\frac{1}{n}$ decay of the Fourier coefficients $c_n$. The wavelet basis is much more localized, and only the wavelets crossing the jump will have large coefficients. Of course the Haar basis could be perfect for a step function, but it has slow decay for a simple linear ramp. Other wavelets of higher order can successfully capture a jump discontinuity in the function or its derivatives.

To emphasize the importance of a good basis (an efficient representation of the data), we will list some important choices. Each example is associated with a major algorithm in applied mathematics. We will mention bases of functions (continuous variable $x$ or $t$) although the algorithm involve vectors (discrete variables $n$ and $k$).

**1**. Fourier series (sinusoidal basis)

**2**. Finite Element Method (piecewise polynomial basis)

**3**. Spline Interpolation (smooth piecewise polynomials)

**4**. Radial Basis Functions (functions of the distance $\|x-x_i\|$ to interpolation points $x_i$ in $d$ dimensions)

**5**. Legendre polynomials, Bessel functions, Hermite functions, ... (these are orthogonal solutions of differential equations)

**6**. Wavelet series (functions $w(2^j x - k)$ from dilation and translation of a wavelet).

There is a similar list of discrete bases. Those are vectors, transformed by matrices. It is time to see the matrix that represents a lowpass filter (convolution matrix), and also the matrix that represents a filter bank (wavelet transform matrix).

# 14 Filters and Filter Banks by Matrices

We begin with the matrix for the averaging filter $y(n) = \frac{1}{2}(x(n-1) + x(n))$. The matrix is lower triangular, because this filter is causal. The output at time $n$ does not involve inputs at times later than $n$. The averaging matrix has the entries $\frac{1}{2}$ on its main diagonal (to produce $\frac{1}{2}x(n)$), and it has $\frac{1}{2}$ on its first subdiagonal (to produce $\frac{1}{2}x(n-1)$). We regard the vectors $x$ and $y$ as doubly infinite, so the index $n$ goes from $-\infty$ to $\infty$. Then the matrix that executes this filter is $H = H_{\text{averaging}}$:

$$y = Hx = \begin{bmatrix} \cdot & & & & \\ \frac{1}{2} & \frac{1}{2} & & & \\ & \frac{1}{2} & \frac{1}{2} & & \\ & & \frac{1}{2} & \frac{1}{2} & \\ & & & \cdot & \cdot \end{bmatrix} \begin{bmatrix} \cdot \\ x(n-1) \\ x(n) \\ x(n+1) \\ \cdot \end{bmatrix}$$

Multiplying along the middle row of $Hx$ gives the desired output $y(n) = \frac{1}{2}(x(n-1)+x(n))$. Multiplying the pure sinusoid with components $x(n) = e^{i\omega n}$ gives the desired output $y(n) = (\frac{1}{2} + \frac{1}{2}e^{-i\omega})e^{i\omega n}$. Thus the frequency response function $H(e^{i\omega}) = \frac{1}{2} + \frac{1}{2}e^{-i\omega}$ is the crucial multiplying factor. In linear algebra terms, the sinusoid $x = e^{i\omega n}$ is an eigenvector of every filter and the frequency response $H(e^{i\omega})$ is the eigenvalue $\lambda$.

We might ask if this averaging matrix is invertible. The answer is no, because the response $\frac{1}{2} + \frac{1}{2}e^{-i\omega}$ is *zero* at $\omega = \pi$. The alternating sinusoid $x(n) = e^{i\pi n} = (-1)^n$ averages to zero (each sample is the opposite of the previous sample). Therefore $Hx = 0$ for this oscillating input.

When the input is the impulse $x = (\ldots, 0, 1, 0, \ldots)$, the response $Hx$ is exactly *a column of the matrix.* This is the impulse response $h = (\frac{1}{2}, \frac{1}{2}, 0, \ldots)$.

Now look at the matrix for any causal FIR filter. It is still lower triangular (causal) and it has a finite number of nonzero diagonals (FIR). Most important, the matrix is "shift-invariant." Each row is a shift of the previous row. If $(h(0), h(1), h(2), h(3))$ are the coefficients $(a, b, c, d)$ of a four-tap filter, then those coefficients go down the four nonzero diagonals of the matrix. The coefficient $h(k)$ is the entry along the $k^{\text{th}}$ diagonal of $H$:

$$H = \begin{bmatrix} \cdot & \cdot & \cdot & & & & \\ d & c & b & a & & & \\ & d & c & b & a & & \\ & & d & c & b & a & \\ & & & \cdot & \cdot & \cdot & \cdot \end{bmatrix}. \tag{54}$$

This is the *Toeplitz matrix* or *convolution matrix.*

Otto Toeplitz actually studied finite constant-diagonal matrices. Those are more difficult because the matrix is cut off at the top row and bottom row. The pure sinusoids $x = e^{i\omega n}$ are no longer eigenvectors; Fourier analysis always has trouble at boundaries. The problem for Toeplitz was to understand the properties of his matrix as $N \to \infty$. The conclusion (see Szegö and Grenander [7]) was that the limit is a *singly* infinite matrix, with one boundary, rather than our doubly infinite matrix (no boundaries).

The multiplication $y = Hx$ executes a discrete convolution of $x$ with $h$:

$$y(n) = \sum_{k=0}^{n} h(k)x(n-k)\,. \tag{55}$$

This impulse response $h = (a, b, c, d)$ is in each column of $H$.

The eigenvectors are still the special exponentials $x(n) = e^{i\omega n}$. The eigenvalue is the frequency response $H(e^{i\omega})$. Just multiply $x$ by a typical row of $H$, and factor out $e^{i\omega n}$:

$$\begin{aligned} y(n) &= de^{i\omega(n-3)} + ce^{i\omega(n-2)} + be^{i\omega(n-1)} + ae^{i\omega n} \\ &= \left(de^{-i3\omega} + ce^{-i2\omega} + be^{-i\omega} + a\right) e^{i\omega n} \\ &= H(e^{i\omega})e^{i\omega n}\,. \end{aligned}$$

To say this in linear algebra language, the Fourier basis (pure sinusoids) will diagonalize any constant-diagonal matrix. *The convolution $Hx = h*x$ becomes a multiplication by $H(e^{i\omega})$.* This is the *convolution rule* on which filter design is based.

A single filter is not intended to be inverted (and is probably not invertible).

To move toward filter banks we introduce *downsampling*. This is the length-reducing operation $(\downarrow 2)y(n) = y(2n)$. Since it is linear, it can also be represented by a matrix. But that matrix does not have constant diagonals! It is not a Toeplitz matrix. Downsampling is not shift-invariant, because a unit delay of the signal produces a completely different set of even-numbered samples (they previously had odd numbers). In some way the matrix for $\downarrow 2$ is short and wide:

$$(\downarrow 2)y = \begin{bmatrix} \cdot & 1 & & & & \\ \cdot & 0 & 0 & 1 & & \\ \cdot & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cdot \\ y(0) \\ y(1) \\ y(2) \\ y(3) \\ y(4) \end{bmatrix} = \begin{bmatrix} y(0) \\ y(2) \\ y(4) \end{bmatrix}. \tag{56}$$

One interesting point about this matrix is its transpose (tall and thin). Multiplying by $(\downarrow 2)^T$ will put zeros back into odd-numbered samples. This is exactly the process of *upsampling* (denoted by $\uparrow 2$):

$$(\uparrow 2)z = (\downarrow 2)^T z = \begin{bmatrix} \cdot & \cdot & \cdot \\ 1 & 0 & 0 \\ & 0 & 0 \\ & 1 & 0 \\ & & 0 \\ & & 1 \end{bmatrix} \begin{bmatrix} z(0) \\ z(1) \\ z(2) \end{bmatrix} = \begin{bmatrix} \cdot \\ z(0) \\ 0 \\ z(1) \\ 0 \\ z(2) \end{bmatrix}. \tag{57}$$

The product $(\uparrow 2)(\downarrow 2)$, with downsampling first, is different from $(\downarrow 2)(\uparrow 2)$:

$$(\uparrow 2)(\downarrow 2)y = \begin{bmatrix} y(0) \\ 0 \\ y(2) \\ 0 \\ y(4) \end{bmatrix} \qquad \text{but} \qquad (\downarrow 2)(\uparrow 2)z = z\,.$$

The lossy equation $(\downarrow 2)$ has a right inverse $(\uparrow 2)$ but not a left inverse.

Now combine filtering with downsampling. The "2" enters into the convolution:

$$(\downarrow 2)Hx(n) = y(2n) = \sum_{k=0}^{N} h(k)x(2n-k)\,. \tag{58}$$

This is the crucial combination for filter banks and wavelets. It is linear but not shift-invariant. To execute $(\downarrow 2)H$ by a matrix, *we remove the odd-numbered rows of $H$.* This leaves a short wide matrix whose rows have a *double shift*:

$$(\downarrow 2)H = \begin{bmatrix} \cdot & \cdot & & & & & & \\ d & c & b & a & & & & \\ & & d & c & b & a & & \\ & & & & d & c & b & a \\ & & & & & & \cdot & \cdot \end{bmatrix}. \tag{59}$$

The matrix multiplication $(\downarrow 2)H_0x$ produces the subsampled lowpass output $y_0(2n)$, which is half of the wavelet transform of $x$.

The other half of the transform is from the highpass channel. It begins with the second filter $H_1$. The output $y_1 = H_1x$ is subsampled to produce $y_1(2n) = (\downarrow 2)H_1x$. The two half-length vectors $z_0$ and $z_1$ give the "averages" and "details" in the input signal $x$. There is overlap between them at all frequencies because the filters are not ideal (they don't have brick wall perfect cutoffs). With properly chosen filters we can still reconstruct $x$ perfectly from $y_0(2n)$ and $y_1(2n)$.

The analysis filter bank consists of $(\downarrow 2)H_0$ and $(\downarrow 2)H_1$. To express the wavelet transform in one square matrix, we combine those two matrices (each containing a double shift):

$$H_{\text{bank}} = \begin{bmatrix} (\downarrow 2)H_0 \\ (\downarrow 2)H_1 \end{bmatrix} = \begin{bmatrix} \cdot & \cdot & & & & \\ h_0(3) & h_0(2) & h_0(1) & h_0(0) & & \\ & & h_0(3) & h_0(2) & h_0(1) & h_0(0) \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ h_1(3) & h_1(2) & h_1(1) & h_1(0) & & \\ & & h_1(3) & h_1(2) & h_1(1) & h_1(0) \end{bmatrix}. \tag{60}$$

The diagonal of $H_{\text{bank}}$ is not constant and the filter bank is not shift-invariant. But it is *double-shift invariant!* If the input $x$ is delayed by two time steps, then the output is also delayed by two time steps — and otherwise unchanged. We have a *block* FIR time-invariant system.

Notice that the columns containing even-numbered coefficients are separated from the columns containing odd-numbered coefficients. The filter bank is dealing with two phases, $x_{\text{even}}$ and $x_{\text{odd}}$, which are producing two outputs, $y_0(2n)$ and $y_1(2n)$.

For a single filter, we selected the special input $x(n) = e^{i\omega n}$. The filter multiplied it by a scalar $H(e^{i\omega})$. The pure frequency $\omega$ was preserved. For a filter bank, the corresponding idea is to select an even phase $x_{\text{even}}(n) = x(2n) = \alpha e^{i\omega n}$ and an odd phase $x_{\text{odd}}(n) = x(2n-1) = \beta e^{i\omega n}$. Then the output in each channel again has this frequency $\omega$:

$$H_{\text{bank}} \begin{bmatrix} \cdot \\ \alpha e^{i\omega(n-1)} \\ \beta e^{i\omega n} \\ \alpha e^{i\omega n} \\ \beta e^{i\omega(n+1)} \\ \cdot \end{bmatrix} = \begin{bmatrix} H_{0,\text{even}} & H_{0,\text{odd}} \\ H_{1,\text{even}} & H_{1,\text{odd}} \end{bmatrix} \begin{bmatrix} X_{\text{even}} \\ X_{\text{odd}} \end{bmatrix}. \tag{61}$$

The crucial point is to identify that 2 by 2 *polyphase matrix*, which is the block analog of the frequency response function $H(e^{i\omega})$ for a single filter. The polyphase matrix is

$$H_{\text{poly}}(e^{i\omega}) = \begin{bmatrix} H_{0,\text{even}} & H_{0,\text{odd}} \\ H_{1,\text{even}} & H_{1,\text{odd}} \end{bmatrix} = \begin{bmatrix} \sum h_0(2k)e^{ik\omega} & \sum h_0(2k+1)e^{ik\omega} \\ \sum h_1(2k)e^{ik\omega} & \sum h_1(2k+1)e^{ik\omega} \end{bmatrix}. \tag{62}$$

In the analysis of filter banks (see [14] and other textbooks), the polyphase matrix plays the leading part. Let us mention the most important point:

*The polyphase matrix for the synthesis filter bank is the inverse of* $H_{\text{poly}}$:

$$H_{\text{bank}}^{-1} \qquad \text{corresponds to} \qquad H_{\text{poly}}^{-1}(e^{i\omega})$$

Therefore the synthesis filter bank is FIR only if $H_{\text{poly}}^{-1}$ is a finite polynomial (not an infinite series in $e^{i\omega}$). Since the inverse of a matrix involves division by the *determinant*, the condition for an FIR analysis bank (wavelet transform) to have an FIR synthesis bank (inverse wavelet transform) is this:

*The determinant of* $H_{\text{poly}}(e^{i\omega})$ *must be a monomial* $Ce^{iL\omega}$ *(one term only).*

Then we can divide by the determinant, and the inverse matrix is a finite polynomial. This is the requirement for a perfect reconstruction FIR filter bank. From $H^{-1}(e^{i\omega})$ we construct the inverse transform, which is the synthesis bank.

**Remark 1.** You can see the blocks more clearly by interlacing the rows of $(\downarrow 2)H_0$ and $(\downarrow 2)H_1$. This will interlace the output vectors $y_0(2n)$ and $y_1(2n)$. We are doing this not in the actual implementation, but in order to recognize the *block Toeplitz matrix* that appears:

$$H_{\text{block}} = \begin{bmatrix} \cdot & \cdot & & & & \\ h_0(3) & h_0(2) & h_0(1) & h_0(0) & & \\ h_1(3) & h_1(2) & h_1(1) & h_1(0) & & \\ & & h_0(3) & h_0(2) & h_0(1) & h_0(0) \\ & & h_1(3) & h_1(2) & h_1(1) & h_1(0) \\ & & & & \cdot & \cdot \end{bmatrix}. \tag{63}$$

The same 2 by 2 matrix is repeated down each *block diagonal*.

**Remark 2.** Matrix multiplication is optional for time-invariant filters and doubly infinite signals. Many non-specialists in signal processing are comfortable with matrices as a way to understand the action of the transform (and block matrices correspond perfectly to block transforms). Specialists may prefer the description in the frequency domain, where a filter multiplies by the simple scalar response function $H(e^{i\omega})$. In popular textbooks like Oppenheim and Schafer's *Discrete-Time Signal Processing*, there is scarcely a single matrix. I meet those students in my wavelet class, and they thoroughly understand filters.

For the finite-length signals that enter in image processing, and for time-varying filter banks of all kinds, I believe that matrix notation is valuable. This approach is especially helpful for non-specialists (who think first of the signal in time or space, instead of in the frequency domain). So we want to comment briefly on the matrices that describe *filtering for a signal of finite length $L$.*

The "interior" of the matrix will have constant diagonals as before. The entry in the $k^{\text{th}}$ diagonal is the filter coefficient $h(k)$. At the top rows and/or bottom rows (the ends of the signal) some change in this pattern is necessary. We must introduce boundary conditions to tell us what to do when the convolution $y(n) = \sum h(k)x(n-k)$ is asking for samples of $x$ that are outside the image boundary (and therefore not defined). We may change $h(k)$ near the boundary or we may extend $x(n)$ beyond the boundary — but we have to do something! And it is often clearest to see $y = H_L x$ as an $L$ by $L$ matrix multiplication.

A crude approach is *zero-padding*, which assumes that those unknown samples are all zero:

$$\text{Define } x(n) = 0 \quad \text{for} \quad n < 0 \quad \text{and} \quad n \geq L\,.$$

In this case the infinite Toeplitz matrix is chopped off to a finite Toeplitz matrix. We have an $L$ by $L$ section of the original matrix. The result is a discontinuity in the signal and a visible artifact (sometimes a dark band at the boundary) in the reconstruction. Most software does better than this.

Cyclic convolution, which assumes a *periodic signal* with $x(L) = x(0)$, is more popular. It preserves the Fourier structure, the DFT of length $L$. There is perfect agreement between *circulant matrices* (Toeplitz with wrap-around of the diagonals) and diagonalization by the DFT. The eigenvectors of every circulant matrix are the DFT basis vectors $v_k = (1, w^k, w^{2k}, \ldots)$ with $w = \exp(2\pi i/L)$. Note that $w^L = 1$, which assures the periodic wrap-around. The corresponding eigenvalues of the circulant matrix are the numbers $H(w^k) = H(e^{2\pi ik/L})$ for $k = 0, \ldots, L-1$. We are seeing $L$ discrete frequencies with equal spacing $2\pi/L$, instead of the whole continuum of frequencies $|\omega| \leq \pi$.

An example will put this idea into matrix notation. The cyclic averaging filter for length $L = 4$ is represented by a 4 by 4 circulant matrix:

$$H_{\mathbf{cyclic}} = \frac{1}{2}\begin{bmatrix} 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}.$$

The entry $\frac{1}{2}$ is on the main diagonal and the first subdiagonal — which cycles around into the top right corner of the matrix. If we apply $H_{\mathbf{cyclic}}$ to an input $x$, the output $y$ is the moving average (with the cyclic assumption that $x(-1) = x(3)$):

$$y = H_{\mathbf{cyclic}} x = \frac{1}{2} \begin{bmatrix} x(0) & + & x(3) \\ x(1) & + & x(0) \\ x(2) & + & x(1) \\ x(3) & + & x(2) \end{bmatrix}.$$

You can see that the zero-frequency input $x = (1,1,1,1)$ is an eigenvector with eigenvalue $H(e^{i0}) = 1$. The next eigenvector is $x = (1, i, -1, -i)$ containing powers of $e^{2\pi i/L} = i$. The eigenvalue is $\frac{1}{2}(1-i)$, which agrees with the response $H(e^{i\omega}) = \frac{1}{2} + \frac{1}{2}e^{-i\omega}$ at the frequency $\omega = \pi/2$. Thus circulant matrices are a perfect match with cyclic convolution and the DFT.

A third approach is preferred for symmetric (or antisymmetric) filters. *We extend the signal symmetrically.* At each boundary, there are two possible *symmetric reflections*:

"Whole-sample" = Reflection with no repeat: $x(2), x(1), x(0), x(1), x(2)$

"Half-sample" = Reflection with a repeat: $x(2), x(1), x(0), x(0), x(1), x(2)$

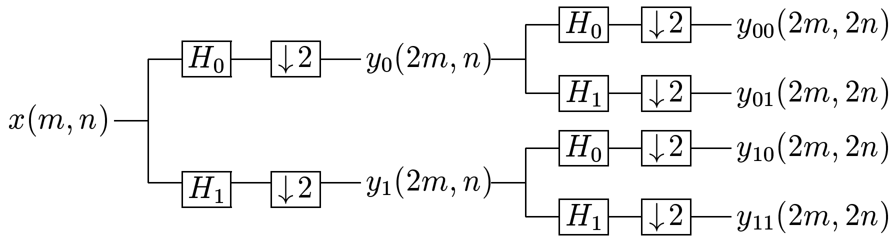
We choose the reflection that agrees with the filter. Symmetric even-length filters have repeats as in $a, b, b, a$; the symmetric odd-length filter $a, b, c, b, a$ has no repeat. This extension process is easy to implement because the filter can act normally on the extended signal, and then the output $y$ is reduced back to length $L$ (and subsampled, when the filter bank includes $\downarrow 2$). Chapter 8 of our joint textbook *Wavelets and Filter Banks* (Wellesley-Cambridge Press 1996) gives the details of symmetric extension [14].

The top and bottom rows of the filter matrix $H$ are modified by this "fold-across" coming from symmetric extension of the input signal. The matrix remains banded but the diagonal entries change in a systematic way near the boundaries. In an FIR filter bank with FIR inverse, the key point is that both transforms (analysis and synthesis, deconstruction and reconstruction) remain banded. This bandedness means a fast transform with a fast inverse. So wavelets have the two essential properties, speed of execution and accuracy of approximation, that make them a good transform.

## 15 Filters and Wavelets for 2D Images

It is easy to describe the most popular and convenient 2-dimensional filters. They come directly from 1-dimensional filters. Those are applied to each separate row of the image. Then the output is reorganized into columns, and the 1-dimensional filters are applied to each column.

This produces four outputs instead of the usual two. The low-low output (lowpass filter on rows and then columns) normally contains most of the energy and the information. This quarter-size image is the input to the next iteration. Effectively, one step of the analysis bank has downsampled the signal by four.




It is useful to follow through the Haar example. The low-low output will be an average over four samples — two neighbors from each row and then averaged over two neighboring rows:

$$y_{00}(2m,2n) = \frac{1}{4}\left[x(2m,2n) + x(2m-1,2n) + x(2m,2n-1) + x(2m-1,2n-1)\right]$$

The other three outputs $y_{10}$ and $y_{01}$ and $y_{11}$ involve the same four input samples. There are minus signs from taking differences instead of averages. The four sign patterns give the same four outputs as a 2-dimensional Discrete Fourier Transform on $2 \times 2$ points:

$$\begin{array}{cc} + & + \\ + & + \end{array} \qquad \begin{array}{cc} + & - \\ + & - \end{array} \qquad \begin{array}{cc} + & + \\ - & - \end{array} \qquad \begin{array}{cc} + & - \\ - & + \end{array}$$

The middle two outputs pick up vertical and horizontal edges. The last output includes (but not so well) any diagonal edges. The 2D version of longer filters with more zeros at $\omega = \pi$ will be much better, but they still have this bias to the vertical and horizontal directions. These are "separable" filters or "tensor product" filters — they separate into 1D times 1D.

When we reconstruct from these four pieces of the wavelet transform, we get four output images. With perfect reconstruction and no compression, they add up to the input image. Each piece uses a quarter of the transform coefficients. It is usual to display the four quarter-size reconstructions in one original-size image (and the low-low part in one corner is always recognizable as a blurred version of the input).

| | |
|---|---|
| low-low (close to original) | high-low (vertical edges) |
| low-high (horizontal edges) | high-high (little energy) |

The next iteration (which effectively has downsampling by 16) splits the low-low quarter of this display into four smaller pieces.

It is also possible to develop genuinely two-dimensional filters. We think of the samples as positioned on a lattice in the two-dimensional plane. We could use all gridpoints $(m, n)$, as above, or we could use the "quincunx lattice" that keeps only half of those points ($m + n$ is even) on a staggered mesh. The nonseparable filters can be more isotropic (less dependent on $xy$ orientation of the image). But these filters quickly become complicated if we want higher accuracy. The separable filters are much simpler to implement, and more popular.

In continuous time, for functions instead of vectors, we have **one scaling function and three wavelets**. The scaling function comes from a two-dimensional refinement equation. The coefficients are $h_0(k, \ell)$ for a nonseparable 2D lowpass filter, and they are $h_0(k)h_0(\ell)$ when the filter is separable. In that case the refinement equation is

$$\phi(x, y) = 4 \sum \sum h_0(k)h_0(\ell)\phi(2x - k, 2y - \ell) \tag{64}$$

The solution will be the product $\phi(x)\phi(y)$ of two 1-dimensional scaling functions.

Similarly the three wavelet equations will have coefficients $h_0(k)h_1(\ell)$ and $h_1(k)h_0(\ell)$ and $h_1(k)h_1(\ell)$. The resulting wavelets are $\phi(x)w(y)$ and $w(x)\phi(y)$ and $w(x)w(y)$. Their translates form a basis for the wavelet subspace $W_0$.

# 16 Applications: Compression and Denoising

We have concentrated so far on the wavelet transform and its inverse, which are the linear (and lossless) steps in the action diagram. This final section will discuss the nonlinear (and lossy) step, which replaces the wavelet coefficients $y(2n)$ by approximations $\widehat{y}(2n)$. Those are the coefficients that we use in reconstruction. So the output $\widehat{x}$ is different from the input $x$.

We hope that the difference $x-\widehat{x}$ is too small to be heard (in audio) and too small to be seen (in video). We begin by recalling the big picture of transform plus processing plus inverse transform:

$$\textit{signal } x \xrightarrow[\text{[lossless]}]{} \begin{array}{c}\textit{wavelet}\\ \textit{coefficients}\end{array} \xrightarrow[\text{[lossy]}]{} \begin{array}{c}\textit{compressed}\\ \textit{coefficients}\end{array} \xrightarrow[\text{[lossless]}]{} \begin{array}{c}\textit{compressed}\\ \textit{signal}\end{array}\ \widehat{x}$$

A basic processing step is **thresholding**. We simply set to zero all coefficients that are smaller than a specified threshold $\alpha$:

$$\text{Hard threshholding: } \widehat{y}(2n) = 0 \text{ if } |y(2n)| \le \alpha\,. \tag{65}$$

This normally removes all, or almost all, the high frequency components of the signal. Those components often contain "noise" from random errors in the inputs. This process of **denoising** is highly important in statistics and Donoho [5] has proposed a suitable threshold $\alpha = \sigma\sqrt{2N}$, with standard deviation $\sigma$ and $N$ components.

Notice that thresholding is not a linear operation. We are keeping the largest components and not the first $N$ components. This is simple to execute, but the mathematical analysis by DeVore and others requires the right function spaces (they are Besor spaces). We refer to [4] for an excellent summary of this theory.

A more subtle processing step is **quantization**. The inputs $y_0(2n)$ are real numbers. The outputs $\widehat{y_0}(2n)$ are strings of zeros and ones, ready for transmission or storage. Roughly speaking, the numbers $y_0(2n)$ are sorted into a finite set of bins. The compression is greater when more numbers go into the "zero bin." Then we transmit only the bin numbers of the coefficients, and use a value $\widehat{y}$ at the center of the bin for the reconstruction step. A vector quantization has $M$ dimensional bins for packets of $M$ coefficients at a time.

This transform coding is of critical importance to the whole compression process, It is a highly developed form of roundoff, and we mention two basic references [6, 8]. I believe that quantization should be studied and applied in a wide range of numerical analysis and scientific computing.

The combination of linear transform and nonlinear compression is fundamental. The transform is a change to a better basis — a more efficient representation of the signal. The compression step takes advantage of that improved representation, to reduce the work. New transforms and new bases will be needed in processing new types of signals. The needs of modern technology impose the fundamental message that has been the starting point of this paper: **Signal processing is for everyone**.

# References

[1] A. Cohen and R.D. Ryan, *Wavelets and Multiscale Signal Processing*, Chapman and Hall, 1995.

[2] I. Daubechies, Orthonormal bases of compactly supported wavelets, *Comm. Pure Appl. Math.* **41** (1988) 909–996.

[3] I. Daubechies, *Ten Lectures on Wavelets*, SIAM, 1993.

[4] R. DeVore and B. Lucier, Wavelets, *Acta Numerica* **1** (1991) 1–56.

[5] D. Donoho, De-noising by soft thresholding, *IEEE Trans. Inf. Theory* **41** (1995) 613–627.

[6] R.M. Gray, *Source Coding Theory*, Kluwer, 1990.

[7] U. Grenander and G. Szegö, *Toeplitz Forms and Their Applications*, University of California Press, Berkeley, 1958.

[8] N.J. Jayant and P. Noll, *Digital Coding of Waveforms*, Prentice-Hall, 1984.

[9] J. LeBrun and Vetterli, Balanced wavelets: Theory and design, *IEEE Trans. Sig. Proc.* **46** (1998) 1119–1124; Higher order balanced multiwavelets, *ICASSP Proceedings* (1998).

[10] S. Mallat, Multiresolution approximation and wavelet orthonormal bases of $L^2$ (R), *Trans. Amer. Math. Soc.* **315** (1989) 69–87.

[11] I. Selesnick, Multiwavelets with extra approximation properties, *IEEE Trans. Sig. Proc.* **46** (1998) 2898–2909; Balanced multiwavelet bases based on symmetric FIR filters, *IEEE Trans. Sig. Proc.*, to appear.

[12] J. Shen and G. Strang, The asymptotics of optimal (equiripple) filters, *IEEE Trans. Sig. Proc.* **47** (1999) 1087–1098.

[13] G. Strang, The Discrete Cosine Transform, *SIAM Review* **41** (1999) 135–147.

[14] G. Strang and T. Nguyen, *Wavelets and Filter Banks*, Wellesley-Cambridge Press, 1996.

[15] V. Strela and A. Walden, Signal and Image Denoising via Wavelet Thresholding: Orthogonal and Biorthogonal, Scalar and Multiple Wavelet Transforms, *Imperial College, Statistics Section, Technical Report TR-98-01* (1998).

[16] M. Unser and T. Blu, Fractional splines and wavelets, *SIAM Review* **41** (1999), to appear.