AMAT 503: Wavelets and Signal Processing

Michael P. Lamoureux

Lecture 8 : February 1, 2018

Contents

0 Introduction

These notes follow the content of the course AMAT 503 at the University of Calgary. They focus on the theory of wavelets and their application to signal processing – in particular to 1D signals (such as sound) and 2D signal (such as photos). For a more in-depth analysis, please read the course textbook (which this year is Patrick van Fllet's "Discrete wavelet Transformations.")

1 Recap of Lecture 7: Filters and Decimation

There are two key steps in the discrete wavelet transform: filtering (both lowpass and high pass) and decimation.

Filtering is done by a convolution filter. In the simplest case, we can use a simply averaging filter as a lowpass filter, and differencing as the high pass filter. For instance, we filter a sequence of data points $\mathbf{x}$ to obtain sequences $\mathbf{y}, \mathbf{z}$ as follows:

$$y_k = \frac{x_k + x_{k+1}}{2}, \qquad z_k = \frac{x_k - x_{k+1}}{2}. \tag{1}$$

Notice we get twice the data out as we get in. For instance, is $\mathbf{x}$ is a sequence of length 1024 data points, both $\mathbf{y}$ and $\mathbf{z}$ have about 1024 data points each, totally 2048 data points. Neither $\mathbf{y}$ nor $\mathbf{z}$ has enough information, but together they do, for we can write

$$y_k + z_k = \frac{x_k + x_{k+1}}{2} + \frac{x_k - x_{k+1}}{2} = x_k, y_k - z_k = \frac{x_k + x_{k+1}}{2} - \frac{x_k - x_{k+1}}{2} = x_{k+1}. \tag{2}$$

That is, the $\mathbf{y}$ and $\mathbf{z}$ can be used to compute the original sequence $\mathbf{x}$, so we have an invertible transformation.

Looking a bit more closely, we notice that if k is even, from $y_k + z_k$ and $y_k - z_k$ we recover both x_k (an even index) and x_{k+1} (an odd index). So from the even data in $\mathbf{y}, \mathbf{z}$, we can recover all the data in $\mathbf{x}$. These leads to the idea of *decimation*, where we throw away every other data point in $\mathbf{y}, \mathbf{z}$. We can do this, and still get an invertible operation.

Thus, our simple wavelet transform (the Haar transform) is defined by the convolutions

$$y_k = \frac{x_k + x_{k+1}}{2}, \qquad z_k = \frac{x_k - x_{k+1}}{2}, \qquad \text{for } k \text{ even only}, \tag{3}$$

and we ignore the odd indices. The convolutions are the filters; the throwing away of odd indices is the decimation.

I need to say something about the filter response of those guys. Blah blah blah, will add this later, bu th e point is

$$|H(\omega)| = \cos(\pi\omega), \qquad |G(\omega)| = \sin(\pi\omega), \qquad 0 \le \omega \le 1/2. \tag{4}$$

So the first is lowpass, second is high pass.

Review how the book defines lowpass, high pass.

The two convolution operators with coefficients $\mathbf{h} = (1/2, 1/2)$ and $\mathbf{g} = (1/2, -1/2)$ represent lowpass and high pass filters, respectively. To see this, we compute the Fourier series and simplify, to see that the lowpass filter has frequency response

$$\begin{aligned} H(\omega) &= \sum h_k e^{2\pi i k\omega} && (5)\\ &= (1/2)e^{2\pi i 0\omega} + (1/2)e^{2\pi i 1\omega} && (6)\\ &= e^{\pi i\omega}\frac{e^{-\pi i\omega} + e^{\pi i\omega}}{2} && (7)\\ &= e^{\pi i\omega}\cos(\pi\omega), && (8) \end{aligned}$$

while the highpass filter has response

$$G(\omega) = \sum g_k e^{2\pi i k\omega} \tag{9}$$

$$= (1/2)e^{2\pi i 0\omega} - (1/2)e^{2\pi i 1\omega} \tag{10}$$

$$= e^{\pi i\omega}\frac{e^{-\pi i\omega} - e^{\pi i\omega}}{2} \tag{11}$$

$$= -e^{\pi i\omega} i \sin(\pi\omega). \tag{12}$$

Notice the little trick we used to factor out an exponential, to get a sum/difference of two exponentials that are complex conjugates of each other. Result is the simple trig functions, sin and cos. It's a good trick – remember it!

Taking absolute values, we see that the lowpass filter has magnitude response

$$|H(\omega)| = |\cos(\pi\omega)|, \tag{13}$$

while the highpass filter has magnitude response

$$|G(\omega)| = |\sin(\pi\omega)|. \tag{14}$$

The frequency response (on the normalized frequency range $0 \le \omega \le 1/2$) is shown in Figure 1.

It is also interesting to note the sum of the squares is a constant, so

$$|H(\omega)|^2 + |G(\omega)|^2 = \cos^2 + \sin^2 = 1. \tag{15}$$

We will see this again with the Daubechies wavelet transformation.

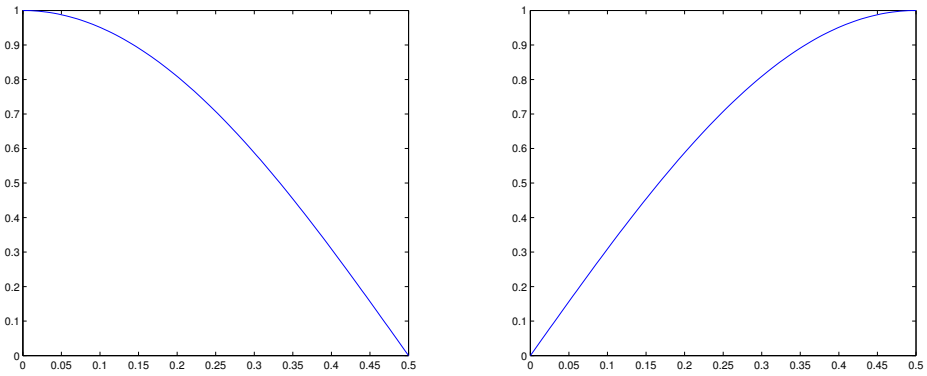

Figure 1: Magnitude response of the lowpass and highpass filters $H(\omega), G(\omega)$, respectively.

Now, we have a lot of freedom in selecting our low and high pass filters. The two given above, with the cos and sin frequency response, are okay, but not great. It might be useful to have filters with a sharper cutoff, as shown in the frequency response of the two filters in Figure 2. The point of the wavelet transform construction is that we can choose our lowpass, and high pass as we please, then decimate, to get (hopefully) an invertible transformation. And if we are really careful, we can get an orthogonal transformation.

The book is rather casual on what constitutes a lowpass or a highpass filter. For a lowpass, we would expect something like the magnitude response to be close to one for low frequencies, and

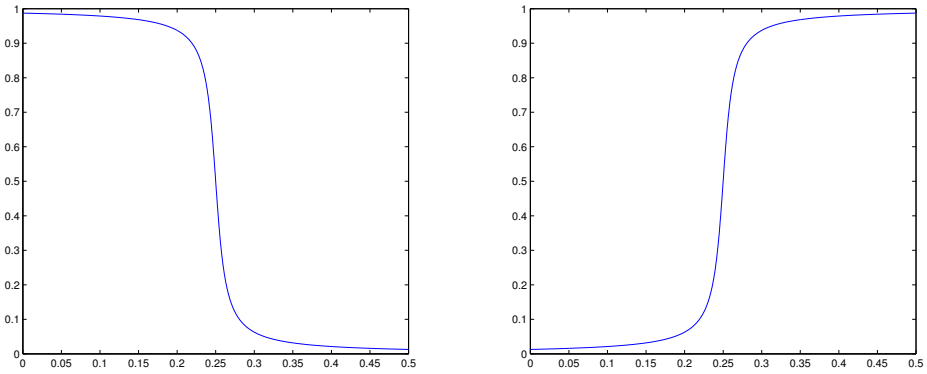

Figure 2: Magnitude response for better, sharper lowpass and highpass filters.

close to 0 for high frequencies. The book defines the following condition

$$\text{Condition 5.1 (Lowpass)} \quad |H(0)| = 1, \quad |H(1/2)| = 0. \tag{16}$$

For a high pass filter, we want the reverse:

$$\text{Condition 5.2 (Highpass)} \quad |G(0)| = 0, \quad |G(1/2)| = 1. \tag{17}$$

Since our magnitude response is always continuous (Why? Think about this!), these two conditions do give something like $H(\omega) \approx 1$ for ω near zero, and $H(\omega) \approx 0$ for ω near $1/2$.

(Keep in mind I am using normalized frequency, so $\omega = 1/2$ corresponds to the frequency at $1/2$ the sampling rate. The book would use $\omega = \pi$ which makes sense for a different normalization. I still think my way is better!)

2 Lecture 8: Haar Wavelet Transform

In the last section, we wrote out the simple Haar transformation using convolution and decimation, yielding the formulas:

$$y_k = \frac{x_k + x_{k+1}}{2}, \qquad z_k = \frac{x_k - x_{k+1}}{2}, \qquad \text{for } k \text{ even only.} \tag{18}$$

This formula can be applied to input sequences $\mathbf{x}$ of arbitrary length, but in practice we fix a length and think of this as a finite dimensional linear transformation.

It is convenient to organize this transform into a matrix, where the top half of the matrix gives the

y and the bottom half gives the **z**. In the 8x8 case, the matrix looks like this:

$$\frac{1}{2}\begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \end{bmatrix}. \tag{19}$$

It is an easy check to see that the rows are all orthogonal to each other. It is convenient to renormalize the matrix so it becomes orthonormal – that is, the rows will have length one if we write it as

$$W = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \end{bmatrix}. \tag{20}$$

This is called the 8x8 Haar wavelet transformation matrix. Obviously we can extend this to any nxn dimension (n even) and obtain and orthogonal matrix. Convince yourself that we have the identity

$$W^*W = I. \tag{21}$$

2.1 Sample code

Let's try out the Haar wavelet transform on some real data, to see what it does. We will take a one period of the sine function, with 1024 sample points, and apply the Haar wavelet transform. In MATLAB, the code is

```
t = linspace(0,1,1024);
x = sin(2*pi*t);
y = HWT1D1(x);
plot(x), plot(t)
```

The result is shown in Figure 3. The plot on the right shows the output of the computation. Note it is in two parts: the first half looks like the original sine wave, the second half looks like something close to zero. We can get a better picture by plotting these two halves separately, as in Figure 4. Notice the second half looks like a cosine wave (the derivative of the original sine, because the high pass filter is a difference operator). But it is very small – the scaling factor shown in the plot is 10^{-3}.

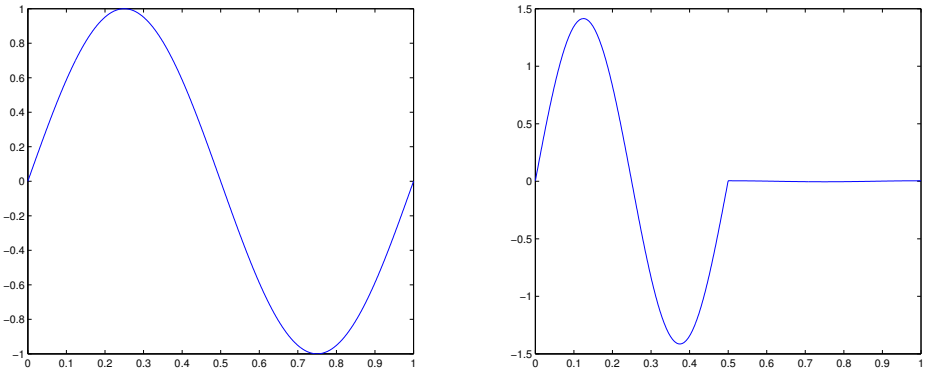

Figure 3: A sampled sine wave, and its Haar transform (notice low pass, high pass split).

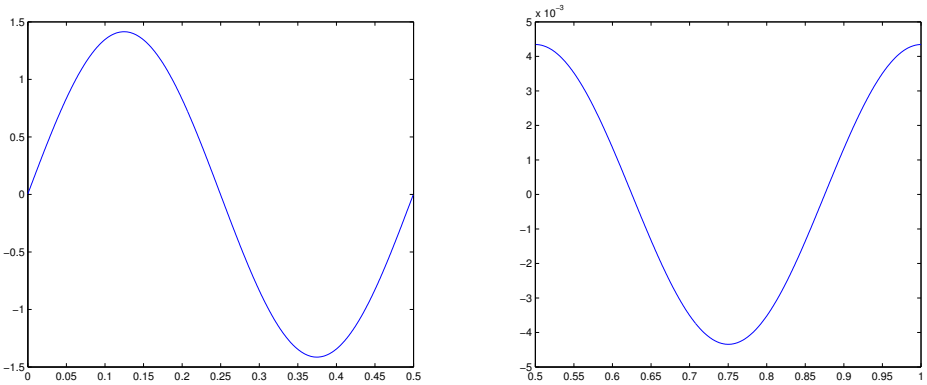

Figure 4: The two parts of the Haar transform, low pass on left, high pass on right.

So why do we do this? The point is to compress most of the energy in the signal into few coefficients. We can see this by plotting the cumulative energy of the original signal and of the transforms signal, as shown in Figure 5. What we notice is that the cumulative energy for the Haar transform version rises to one much faster. Which is to say, more energy is concentrated in fewer coefficients.

2.2 Iterating the transform

Let's be careful about what it means to iterate the Haar transform. You don't apply it over and over ageing to the whole output. What you do is apply the transform to the 1024 samples in the initial signal. Then you apply it to the 512 samples of the first half of the output – this is where the signal is large and we hope to compress it. The result is the output is 512 samples, the first 256 are the result of a low pass, the second 256 are the result of the high pass. The remaining 512 sample are left untouched.

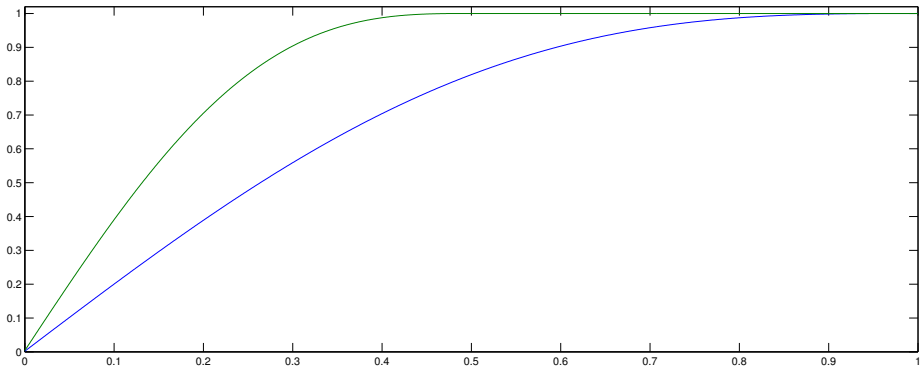

Figure 5: Cumulative energy of the original signal (sine) and its Haar transform.

On the next iteration, we apply the transform to the first 256 samples, and leave the rest alone. We can plot the results of the three iterations in Figure 6. What you should notice is that the main bulk of the signal shows up in the first 1/2 of the 1st iteration, in the first 1/4 of the 2nd iteration, and in the first 1/8 of the 3rd iteration. Overall, this means that with more iterations, there are fewer "big coefficients."

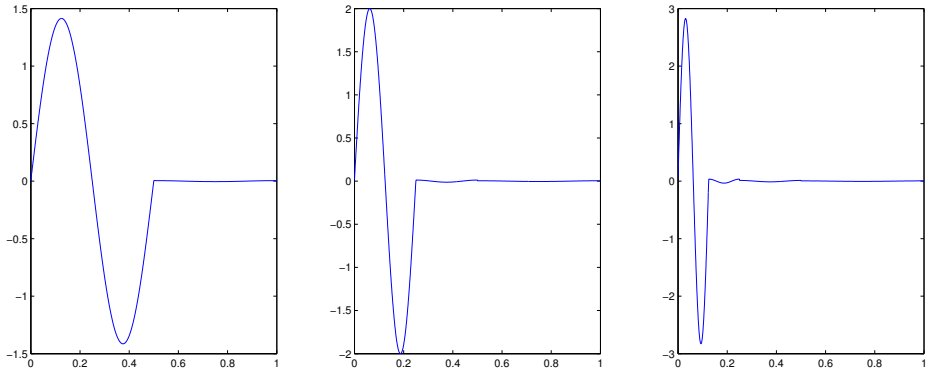

Figure 6: 1st, 2nd and 3rd iterations of the Haar transform, applied to the sine wave.

Let's iterate the transform, and see what happens with the cumulative energy.

```
y1 = HWT1D(x,1);
y2 = HWT1D(x,2);
y3 = HWT1D(x,3);
plot(t,CE(x),t,CE(y1),t,CE(y2),t,CE(y3))
```

A plot of the various cumulative energies shows the more we iterate the Haar transform, the more energy gets concentrated into fewer coefficients. As shown in Figure 7.

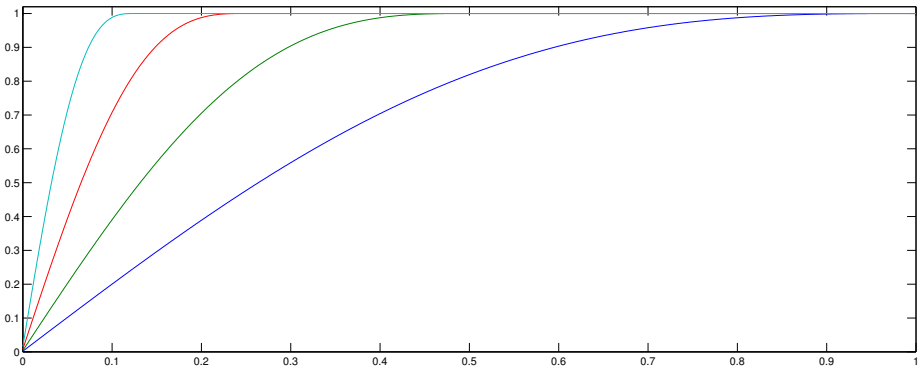

Figure 7: Cumulative energy of the original signal, and 3 iterations of the Haar transform.

2.3 Operations count

It is worth noting that the above 8x8 matrix, when applied to vector $\mathbf{x}$, will compute the output in about 16 floating point operations. There are 4 additions, 4 subtractions, and 8 multiplications by $1/\sqrt{2}$. In the general nxn case, it will take 2n operations. This is considerably faster than the usual matrix-vector multiplication, which takes $O(n^2)$ operations.

When you iterate the Haar wavelet transform, it is not much worse. The reason is we decrease the size of the input vector by a factor of 2 at each stage. So, for instance, if we start with a vector of length 1024, we need 2*1024 operations. In next iteration, we have an input of length 512, operations needed are 2*512. Next step, input is length 256, operations is 2*256. This continues until we get down to an input of length 1. The total operations is

$$2 \cdot 1024 + 2 \cdot 512 + 2 \cdot 256 + 2 \cdot 128 + \cdots 2 \cdot 1 = 2 \cdot 2047, \tag{22}$$

or about $4n$ operations, with $n = 1024$ the length of the original input.

With other wavelet transforms we will consider in later lecture, there are typically k operations per output, where k is a small number like 2,4,6, 8, or so. The same calculation as above shows the total number of operations for an input vector of length n is about $2kn$.

2.4 Ordering the iterations

In practice, it has been found useful to iterate always on the lowpass output. However, this is not a requirement of the wavelet transform. One could iterate on the high pass output only. Or one could alternate between the lowpass output, and the high pass output. At each stage, it is important to choose only one of the lowpass or high pass output, so that the total number of operations to compute is kept low.

The choice can be made adaptively – you check the total energy in the low and high pass output,

and iterate on the one that has more energy (on the theory that this is where you get the most gains).

Or, you could choose to iterate on both the lowpass and high pass outputs. This process is known as using "wavelet packets" or optimal sub band tree structuring.

But for now, let's just assume we iterating on the low pass outputs only.