Wavelets

E. Rosolowsky

1. Basic Signal Processing

A large section of astrophysics is dedicated to rigorous analysis of astrophysical signals. Signal processing represents the full array of techniques that are used to extract meaningful results from telescope data that are often of highly dubious quality. This section of the course will focus on signal extraction, filtering and compression using *wavelets*. Wavelets are an increasingly popular set of functions that are used in a fashion quite similar to the Fourier transform. However, because of the differences between the functions used, wavelets have strengths in areas that the Fourier transform does not.

The Fourier transform is an excellent and well-used signal processing technique. As an example, it is ideal for finding periodic signals in very noisy data (like pulsar data). Effectively, the signal is moved from the time domain, where the power in the signal is scattered throughout the time sample, to the frequency domain where all the power in the signal is concentrated at one or a few frequencies. This makes it relatively easy to identify the signal. This represents signal extraction. To filter the data, we'd make the assumption that all the frequencies that don't have obvious signal are just the result of noise and set those to zero. With an inverse Fourier transform, the signal, with the zeroed out frequencies can be moved back into the time domain where the periodic signal will magically appear and be noise free. Finally, signal compression: if the signal is represented by only a few frequencies, then we could adequately represent the signal with just a few Fourier components, drastically reducing the amount of data required to represent the signal. In contrast, Fourier transforms are less adept at finding isolated signals, and wavelets can be used to better effect in this situation.

1.1. The Need for Significance Tests

All this requires some caution. As a noteworthy example, consider a time signal of pure noise, normally distributed. The Fourier transform of such a signal is plotted in Figure 1. We impose a filter around the frequencies ± 20 (δt^{-1}) of a Gaussian shape which is indicated by the thick gray line. The resulting inverse Fourier transform shows a "signal" with a frequency corresponding to the filter. This illustrates that reckless filtering can create the impression of a signal in the data. White light interferometry relies upon this principal, but in this case, we have extracted a signal that is meaningless. Note, however, the amplitude of the signal, which is significantly smaller than original signal. This is because we have filtered out most of the power in the original signal, which occurred at other frequencies.

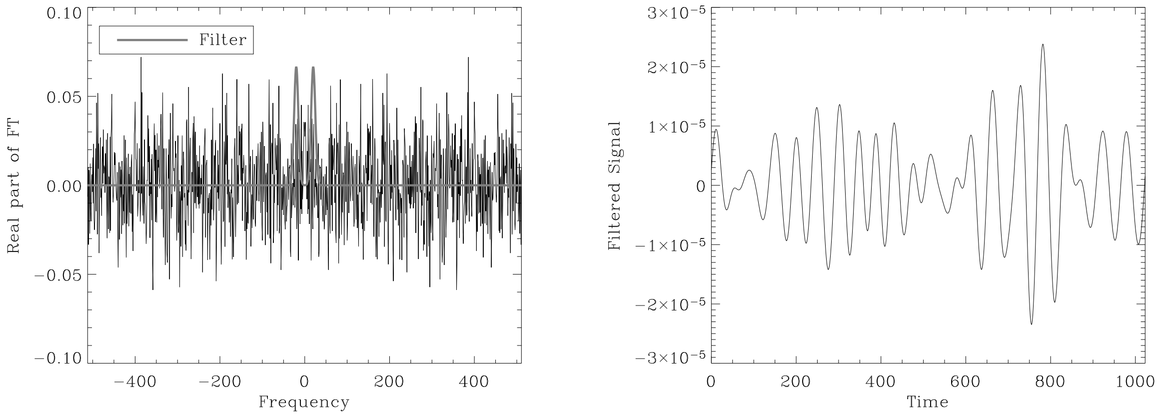

Fig. 1.— The left panel is the Fourier transform of a random signal characterized by a variance $\sigma^2 = 1$ with a Gaussian filter overplotted. The right panel is the inverse transform of the filtered data. The result shows a "signal" arising from noisy data.

The purpose of this example is to illustrate the need for a tool to measure the significance of filtering products. You might imagine that it's possible to determine when signal in a Fourier transform is significant in a signal with normally distributed noise. You'd be right. For now, just note that whatever tools we develop, we'll need a way to determine the significance of the transform products.

1.2. Why develop wavelets?

The Fourier transform is really useful for finding periodic signals in data. This is because the Fourier transform is rooted in periodic functions: sines and cosines. The transform integral looks like this:

$$F(\omega) \propto \int_{-\infty}^{\infty} f(t)\cos(\omega t)dt$$

Speaking in a sloppy fashion, the value of $F(\omega)$ is large when the function $f(t)$ is similar to $\cos(\omega t)$ and small when it is not. This sort of reasoning shows up repeatedly throughout what follows. Thus, a signal that has an angular frequency ω will have large values because the function $\cos(\omega t)$ is well matched to its shape. For non-periodic functions, we might suspect the Fourier transform might not be quite as successful. So, let's consider the function $g(t) = \sin(t^2)$. Figure 2 shows the graph of this function and the power of the corresponding Fourier transform.

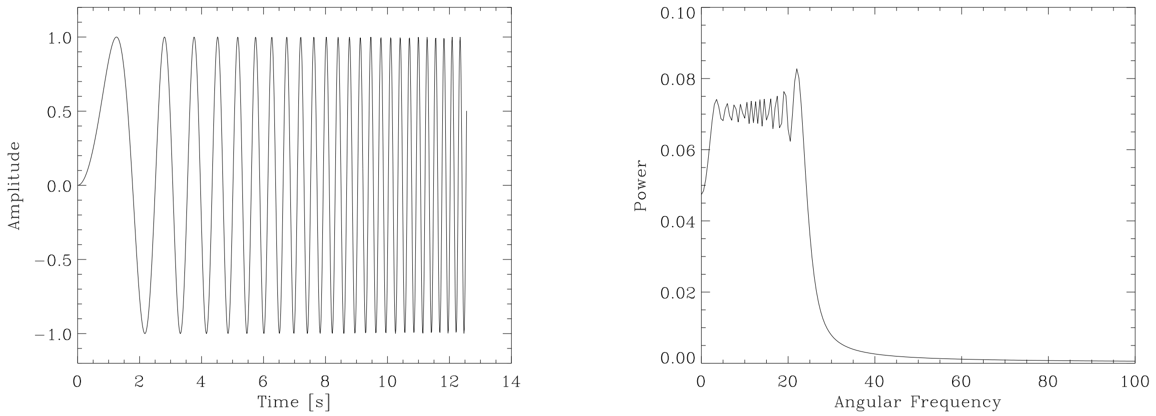

Fig. 2.— The left panel plots the function $g(t) = \sin(t^2)$ and the right panel shows the power of the Fourier transform, $|\widehat{g}(\omega)|^2$.

1.3. The Windowed Fourier Transform

By looking at the signal and its transform, you immediately see that there is power at a lot of different frequencies. This happens many times in signal processing, but what makes this signal unique is that the frequencies appear at specific parts of the signal: low frequencies appear in the first part and higher frequencies appear in the later parts. However, the Fourier transform has no way of representing this feature; there is no way to tell, from the transform data alone, where in the signal a particular frequency is important. One solution is to develop the "windowed" Fourier transform, that is to only look at small sub-regions of the data and examine the frequencies present in that window. Let's consider, a Gaussian weight function on the signal. Gaussian window functions eliminate some pathologies associated with sharp edges of the windowing functions. Let's consider three windows on our data:

$$
\begin{aligned}
w_1(t) &= \exp\left(-\frac{2t^2}{\pi^2}\right) \\
w_2(t) &= \exp\left(-\frac{2(t-2\pi)^2}{\pi^2}\right) \\
w_3(t) &= \exp\left(-\frac{2(t-4\pi)^2}{\pi^2}\right)
\end{aligned}
$$

These are shown in Figure 3 as the black, thick grey and thick black envelopes respectively (dotted lines) and the resulting signals are shown as the solid lines of the same colors. The Fourier transforms of the windowed signals are shown in the bottom panel in comparison with the original signal. Note that the three windows select individual regions of the frequency space, illustrating the property that we want: time localization of the frequency analysis.

So, why talk about wavelets. It might surprise you, but we've been discussing wavelets for

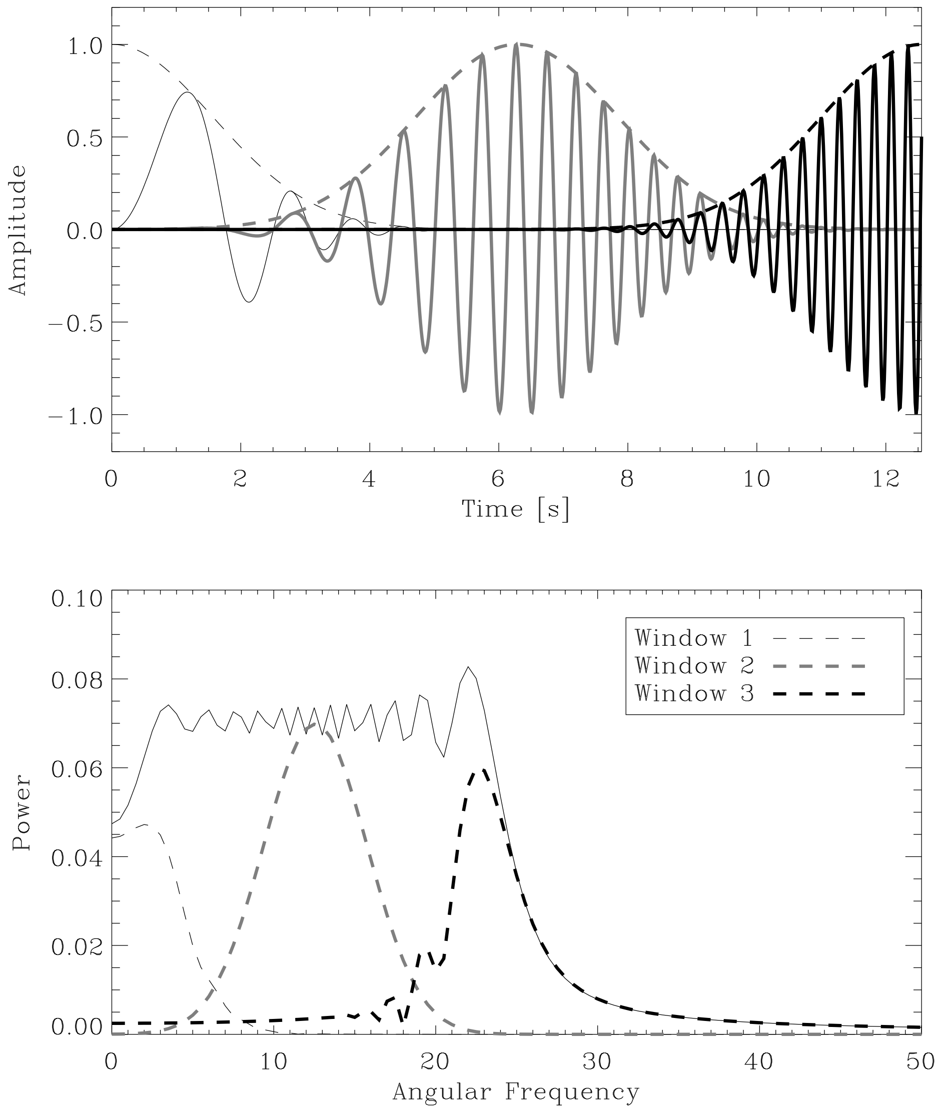

Fig. 3.— The three windowed subsections (w_1, w_2 and w_3 as described in the text) are plotted as dashed curves of light black, grey, and heavy black curves respectively. The resulting windowed signals are plotted as solid lines of the light black, gray and heavy black respectively. The bottom panel shows the Fourier transform of the original signal in thin black and the windowed regions in dashed curves of solid black, gray and heavy black lines. This shows that the windowed Fourier transform can be used to localize the signal analysis in time.

several paragraphs now. The windowed Fourier transform can be recast in terms of wavelets, with a few slight modifications. Let's examine what happened mathematically. Instead of the original

Fourier equation, we've modified the Fourier transform to look like

$$F'(\omega) \propto \int_{-\infty}^{\infty} w(t)g(t)\cos(\omega t)dt$$

where $w(t)$ is a weight function. The form of the weight function is $w(t) = \exp(-\tau^2/2)$ with $\tau = (t-b)/a$ where b is the offset of the Gaussian and a is the width of the Gaussian. If we set $g(t) = 1$, we can deduce what the effect of $w(t)$ has on the basis functions $\cos(\omega t)$. Because of our experience with Fourier transforms and the convolution theorem in particular, we can proceed without any explicit calculation[1]:

$$F'(\omega) = \widehat{W}(\omega') \circledast \delta(\omega' - \omega) + \widehat{W}(\omega') \circledast \delta(\omega' + \omega) = \widehat{W}(\omega) + \widehat{W}(-\omega)$$

Here, $\widehat{W}(\omega)$ is the Fourier transform of the Gaussian weight function, also a Gaussian and the delta functions arise from the transform of the cosines. So, what has happened? By restricting the region of analysis in the time domain, we have broadened our frequency basis function, which was originally a delta-function, by convolving it with a Gaussian. This is one of the key properties of wavelets: they restrict their attention to areas that are finite in *both* the frequency and time domains. In contrast, the Fourier transform implicitly relies upon an infinite extent in the time domain and an infinitely small region of the frequency domain. So, in that sense, the windowed Fourier transform is a kind of wavelet analysis. We'll need to make this more rigorous and add in some other restrictions.

Let's consider the example of the function $g(t) = \sin(t^2)$ some more. Furthermore, let's consider the family of Gaussian weight functions $w(t, a, b) = \exp(-\tau^2/2)$ with $\tau = (t-b)/a$ once again. Instead performing analysis for all angular frequencies ω, let's consider a fixed frequency ω_0 and ask the question 'Where does the signal have frequency ω_0?' This is a question that wavelet analysis is well suited for answering.

To approach this problem, we'll just calculate the Fourier cosine transform at a single frequency $\omega_0 = 20$ rad s^{-1} for a variety of a and b in the weight function. The results of this analysis are shown in Figure 4. The figure shows that there is a unique width (a_{max} and time offset b_{max} for which the response is maximized and we can say, vaguely, that in a width a_{max} around time b_{max}, the signal looks most like a cosine wave with frequency 20 radians per second.

2. The Wavelet Transform

Throughout the past section we have used a measure of how similar two functions are to each other:

$$\langle f(t), g(t)\rangle = \int_{-\infty}^{\infty} f(t)g^*(t)dt \tag{1}$$

[1]The symbol $\circledast$ is used for convolution and $*$ is used for complex conjugation

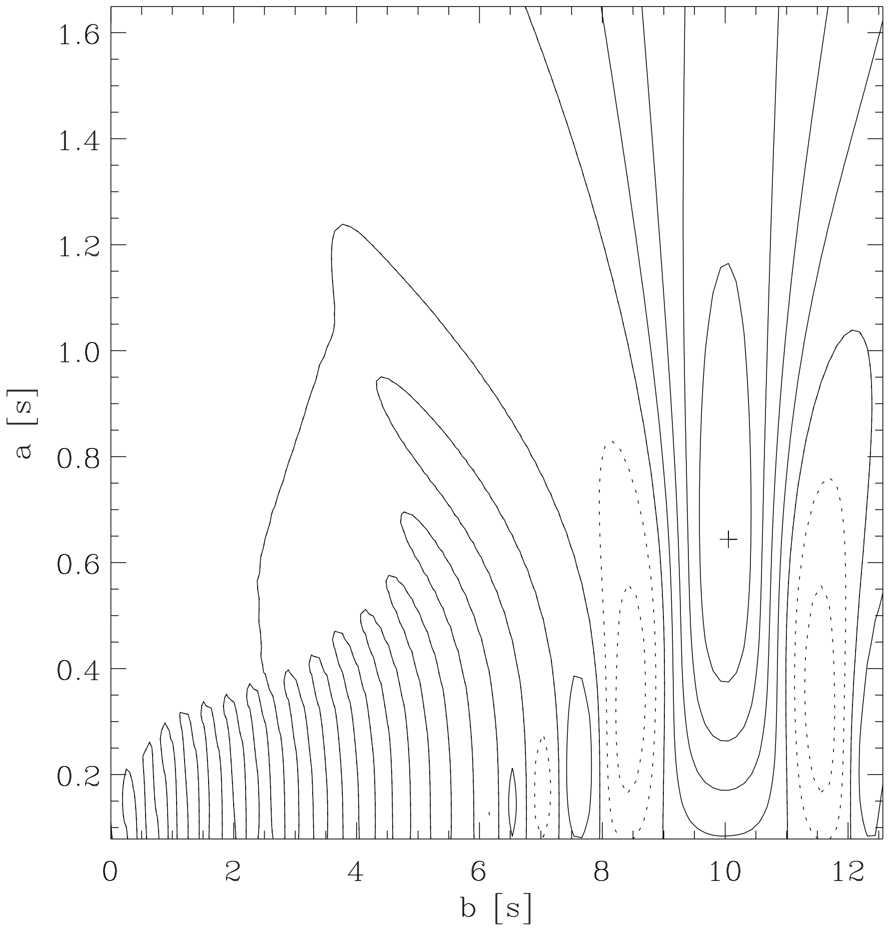

Fig. 4.— Fourier cosine transform for $g(t) = \sin(t^2)$ at $\omega_0 = 20$ for a variety of a and b values in the window function. The maximum value of the transform is marked with a cross. The contours descend in increments of 20% of the maximum from the maximum. Negative contours are dotted. The cross represents range where the signal looks most like a cosine wave with $\omega_0 = 20$.

The complex conjugate ($*$) is thrown in for compatibility with complex functions. Again, if the functions are similar, the absolute value of $\langle f(t), g(t)\rangle$ will be large. If they are different, the value will be small. Remember, the value is just a real (or maybe complex) number. Examine the notation: the angle brackets are reminiscent of the notation for an inner (or dot) product in vector geometry or bra-ket notation in quantum mechanics for a reason. The language of functional analysis that is used here draws explicit parallels between vectors in a vector space and and functions in a "function space." This integral is a way of expressing the dot product between two functions. Also note in a vector space that the length of a vector is measured by taking the square root of the vector dotted with itself: $||f(t)||^2 \equiv \langle f(t), f(t)\rangle$. The same is true here:

$$||f(t)||^2 = \int_{-\infty}^{\infty} f(t)f^*(t)dt = \int_{-\infty}^{\infty} |f(t)|^2 dt$$

This is called the "L^2 norm of function space." All functions with a finite value of $||f||$ are called L^2 functions, and are also known to the rest of the world as "square-integrable." This is important because it means the functions are friendly and well behaved. In Quantum Mechanics the bra-ket notation was used to project a blended wave function onto "basis" states *e.g.* the states of the H I atom. The bra-ket product $\langle f|\psi\rangle$ was used to calculate the amplitude of one basis vector in a mixed state, so this notation is *exactly* the context in which bra-ket notation was developed in quantum.

Returning to wavelets, we'll define the wavelet transform of a function f with respect to a wavelet ψ as $\langle f, \psi\rangle$. In particular, the wavelet is required to be a function of $\tau = (t-b)/a$:

$$\psi(\tau) = \psi\left(\frac{t-b}{a}\right)$$

The wavelet $\psi(\tau)$ is referred to as the *mother wavelet.* In general the wavelet is scaled to

$$\psi = \frac{1}{\sqrt{a}}\psi\left(\frac{t-b}{a}\right) \tag{2}$$

so that the L^2 norm is the same for all wavelets derived from a given mother wavelet. Such a group is called a *wavelet family.* So, the full result of the wavelet transform is the inner product between the function f and the wavelets, with a full range of a and b. We'll define this result as $W(a,b)$ making this the standard notation for a wavelet transform:

$$W(a,b) \equiv \langle f(t), \psi(a,b,t)\rangle = \int_{-\infty}^{\infty} f(t)\frac{1}{\sqrt{a}}\psi^*\left(\frac{t-b}{a}\right) dt \tag{3}$$

The result of the transform, $W(a,b)$ is, as the notation implies, a function of a and b and the amplitude represents how well matched the wavelet is to the input function as a function of a and b. We define the *wavelet power* as $P(a,b) \equiv |W(a,b)|^2$.

At this point, the discussion implies that *any* L^2 function can be a wavelet (like a Gaussian), but we want to impose a further restriction. We'll require that

$$\int_{-\infty}^{\infty} \frac{1}{\sqrt{a}}\psi\left(\frac{t-b}{a}\right) dt = 0$$

This is, purportedly, why wavelets are so named: they wave above and below the axis so that their total integral is zero. That could just be an old mathematicians tale. This is often expressed in terms of the "admissibility criterion" for wavelets which requires the integral

$$C_\psi = \int_{-\infty}^{\infty} \frac{|\widehat{\psi}(\omega)|^2}{\omega} d\omega$$

to be finite. Here, $\widehat{\psi}(\omega)$ is the Fourier transform of the wavelet $\psi(\tau)$. As an aside, the admissibility criterion is a little less stringent than the requirement that the wavelet have zero integral but if the wavelet integrates to zero, then $\widehat{\psi}(0) = 0$ since there is no total power, and the wavelet will fulfill the admissibility criterion. The reason for this requirement is that it allows the reconstruction of a signal from the results of a wavelet transform. The inverse transform of the wavelet function is

$$f(t) = \frac{1}{C_\psi} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} W(a,b)\psi(a,b,t) \frac{da\, db}{a^2}. \tag{4}$$

This looks complicated, but rest assured, you will not have to calculate these explicitly.

Without further ado, let's look at some wavelets! Perhaps the closest wavelet to what we've been examining is the Morlet Wavelet after the mathematician that popularized it: Susan Wavelet. The wavelet is defined as

$$\psi(\tau) = \pi^{-1/4} \exp(i\omega_0 \tau) \exp(-\tau^2/2) \tag{5}$$

This is rather familiar! It's just the Gaussian envelope on a cosine and sine function. ω_0 is a dimensionless frequency. The only difference between this function and the one we considered at the end of the previous section is that the sines and cosines are fixed with respect to the wavelet and not with respect to the signal.

The Morlet wavelet is complex-valued. In contrast, a family of wavelets that's often used which is strictly real is the derivative of a Gaussian class of wavelets. They are defined as

$$\frac{-1}{\sqrt{\Gamma(m+1/2)}} \frac{d}{d\tau} \exp\left(-\frac{\tau^2}{2}\right)$$

We'll focus on the case where $m = 2$ so that this equation becomes

$$\sqrt{\frac{4}{3\pi}} (1-\tau^2) \exp\left(-\frac{\tau^2}{2}\right)$$

This is often called the Mexican Hat Wavelet. Figure 5 plots a Morlet wavelet and two derivative of Gaussian wavelets (DOG).

3. Using Wavelets to Detect Signals

You may have noticed that the wavelet transformation moves a perfectly good 1-D time series into a 2-D array of power as a function of scale (a) and offset (b). This seems like a large expansion of an otherwise compact dataset and you may have serious doubts about how we'll possibly make it to data compression. We'll get around to this. Here, we'll use illustrate how wavelets can be used to detect signals. One common type of problem in astronomical data analysis is finding a Gaussian signal in a noisy set of data. Consider the signal in the left-hand panel of Figure 6. Two Gaussians have been injected into the signal. One has signal-to-noise in one second of 0.8 and a width of 10

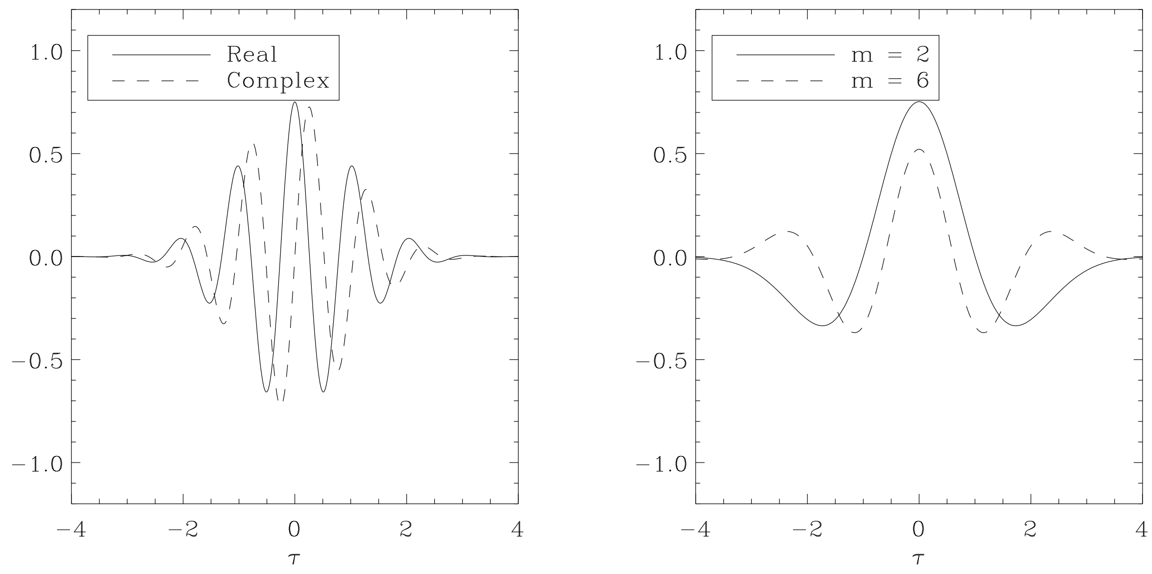

Fig. 5.— The left panel plots a Morlet wavelet with $\omega_0 = 6$ and the right panel plots two derivative of Gaussian wavelets (DOG) with $m = 2$ and $m = 6$.

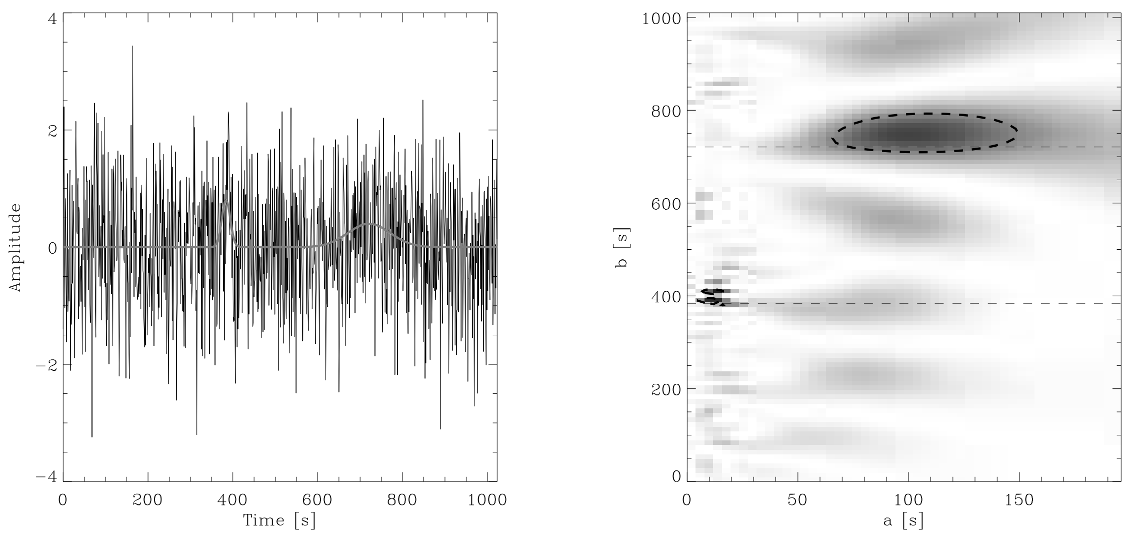

Fig. 6.— The left panel plots a test signal with low signal to noise. The signal without noise is plotted with a thick gray line. The right panel plots a grayscale image of the power in the wavelet coefficients using a DOG wavelet with a degree of $m = 2$. The dashed black contours are the 99% confidence limit on the noise producing the wavelet coefficients of amplitude shown. The horizontal dotted lines indicate the locations at which the signal is located.

s. This pulse is centered at 387 s. The other Gaussian has a signal-to-noise of 0.4 in 1 second and a width of 50 s and is centered at 721 s.

Since the signal is a Gaussian, using a DOG wavelet is a good idea. This sort of reasoning is quite common with wavelets: the wavelets are chosen based on how well matched they are to the expected signals. (Some people deride this choice as arbitrary, but the snipey wavelet reply is just to say that the choice of a Fourier basis or a Legendre basis is equally arbitrary.) We use a wavelet with $m = 2$ and perform a transform for a range of $a \in [0, 200]$ and $b \in [0, 1024]$. This produces wavelet coefficients as a function of a and b. The wavelet power is just the absolute value of the coefficients squared (use the modulus squared for complex coefficients). A grayscale image of the wavelet power is shown in the left-hand panel of Figure 6. The image shows significant power at the offsets (b) corresponding to the two signal pulses and the scales (a) at which the power peaks shows that the peak at 387 s is narrower than the one at 721 s. Black contours indicate the 99% confidence interval in the significance of our signal detection.

How are we determining whether the power in the wavelet coefficients is "significant?" We accomplish this by determining the response of the wavelet transform to a random signal, $n(t)$. Then, since the wavelet transform is a linear transform[2] , we can break up an input signal $f(t)$ into signal components $s(t)$ and noise components $n(t)$:$f(t) = s(t) + n(t)$. If the wavelet transform of the composite signal shows a response larger than we'd expect for a random signal, we'll suspect true signal at the scale and offset where the excess occurs. Before we get to actually evaluating the response of the Wavelet transform to noise, we need to make a few statements about how the transform is actually calculated.

3.1. Implementing the Continuous Wavelet Transform

As written, the wavelet transform involves a lot of calculation and you might suspect that a lot of it is redundant. You'd be right. Again. To speed things up, the first thing to look at in the wavelet transform is to view the transform as a convolution with b as the lag parameter. And when we hear convolution we think Fourier Transforms! If we regard a as fixed, then we can write down the inner product of the functions in terms of b:

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

So, if we take the Fourier transform of this equation, we get

$$\widehat{W}(\omega_a, \omega_b) = \sqrt{a} \widehat{f}(\omega) \widehat{\psi^*}(a\omega)$$

[2]In the sense that the transform of a sum is the sum of the transforms, and the transform of the product of a scalar and a function equals the scalar times the transformed function.

which is just the convolution theorem of Fourier transform wrapped up with the scaling theorem which says

$$\psi\left(\frac{t}{a}\right) \Longleftrightarrow a\widehat{\psi}(a\omega)$$

The notation $\Longleftrightarrow$ indicates the two functions are a Fourier transform pair. Note that the evaluation of the wavelet transform is particularly easy if you know the explicit form of the wavelet's Fourier transform. Then, for each scale, the wavelet coefficients can be calculated by simply evaluating the transformed wavelet in the frequency domain, multiplying by the transformed data to do the convolution, and inverse transforming. The Fourier transforms of the two wavelets we've mentioned above are:

$$\text{Morlet: } \pi^{1/4} H(\omega) \exp\left[\frac{(a\omega-\omega_0)^2}{2}\right] \qquad \text{DOG: } \frac{-i^m}{\sqrt{\Gamma(m+1/2)}}(a\omega)^m \exp\left[-\frac{(a\omega)^2}{2}\right]$$

Here, $H(\omega)$ is the Heaviside step function which has $H(\omega) = 0$ for $\omega < 0$ and 1 otherwise.

Hence, a quick algorithm for implementing the continuous wavelet transform in the case where the Fourier transform can be explicitly evaluated is

1. Fourier transform the data.
2. For each scale a, multiply the scaled transform of the mother wavelet times the transformed data. See §3.3 for a discussion of what scales need to be sampled.
3. Transform each scale back.

3.2. The Noise Response of the Wavelet Transform

To determine the response of a wavelet transform to signal, we consider the power in the wavelet transform, $P = |W(a,b)|^2$, for a random input signal $n(t)$.

However, as pointed out in the previous section (which is why it is where it is), the wavelet transform can be thought of as a convolution between a wavelet of scale a and a function as a function of lag parameter b. Then,

$$\mathcal{E}(P) = \mathcal{E}\left(|n(t) \circledast \psi^*(a,b,t)|^2\right)$$

However, the expectation value of this function can be Fourier transformed to split up the convolution:

$$\mathcal{E}\left(|n(t) \circledast \psi^*(a,b,t)|^2\right) = \mathcal{E}\left(\int_{-\infty}^{\infty} |n(\omega)|^2 |\psi(a,b,\omega)|^2 d\omega\right) = \int_{-\infty}^{\infty} \mathcal{E}(|\widehat{n}(\omega)|^2)|\widehat{\psi}(a,b,\omega)|^2 d\omega \qquad (6)$$

The first equality holds by transforming into the Fourier domain and following the proof of the Power theorem (Parseval's Theorem).

$$\begin{aligned}
|n(t) \circledast \psi^*(a,b,t)|^2 &= [n(t) \circledast \psi^*(a,b,t)]\,[n(t) \circledast \psi^*(a,b,t)]^* \\
&= \int_{-\infty}^{\infty} \widehat{n}(\omega)\widehat{\psi}^*(\omega)\exp(-i\omega t)d\omega \int_{-\infty}^{\infty} \widehat{n}^*(\omega')\widehat{\psi}(\omega')\exp(i\omega' t)d\omega' \\
&= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \widehat{n}(\omega')^*\widehat{n}(\omega)\widehat{\psi}(\omega')\widehat{\psi}(\omega)^* \exp\left[i(\omega-\omega')t\right]\, d\omega\; d\omega' \\
&= \int_{-\infty}^{\infty} |\widehat{n}(\omega)|^2|\widehat{\psi}(\omega)|^2 d\omega
\end{aligned}$$

Returning to the last equality in Equation 6, the expectation value has been passed into the integral which can be done for *random* signals. The expectation value of a known wavelet function is equal to the function (just like the expectation value of 3 in many realizations of 3 is 3). The expectation value for a random signal is determined by characterized by its variance σ^2 so Equation 6 becomes

$$\mathcal{E}\,(P) = \int_{-\infty}^{\infty} \mathcal{E}(|\widehat{n}(\omega)|^2)|\widehat{\psi}(a,b,\omega)|^2 d\omega = \int_{-\infty}^{\infty} \sigma^2|\widehat{\psi}(a,b,\omega)|^2 d\omega = \sigma^2 \tag{7}$$

So, we have shown that the expectation value of wavelet power is equal to the variance of a random signal. The wavelet transform preserves the variance of the noise in the resulting coefficients. We would like to know not just the expectation values but the distribution of the wavelet coefficients and the wavelet power. This is not easy to demonstrate, however, in a statement without proof, the wavelet coefficients for a pure noise signal are distributed according to a normal distribution with variance σ^2 for all a and b. This means that the wavelet power is distributed as a χ^2 distribution with 1 Degree of Freedom *if the wavelet is real* and 2 DOF if the wavelet is complex. This is because a random set of complex numbers has a normal distribution of variance σ^2 in the real plane and the same in the complex plane. In other words, the PDF, ϕ of the wavelet power P is:

$$\phi(P) = \chi^2\left(\frac{Pk}{\sigma^2}, k\right)$$

where $k = 1$ for real wavelets and $k = 2$ for complex wavelets and σ^2 is the expectation value of the noise variance. Thus, using the χ^2 cumulative distribution, we can choose a probability cutoff and require the wavelet power associated with a signal to be greater than this level to be "real." In Figure 6, we looked for what level of wavelet power would be generated by random noise only 1% of the time. Using the `CHISQR_CVF` function in IDL with arguments of 0.01 for the 1% confidence interval and 1 for the degrees of freedom (the DOG is a real wavelet). Then, we put in a contour at this 99% confidence interval showing there is significant power around where the Gaussian input signals are located, as in Figure 6.

3.3. Redundant Information

As we've noted before, the continuous wavelet transform contains a lot of redundant information. In the next section, we will begin discussion the discrete wavelet transform. Like the Fourier transform, we want to know how many sample values are needed to accurately reproduce a signal. In particular, we are concerned with how many scales (a) and offsets (b) are needed to obtain a complete picture of the wavelet transform.

The first thing we could do to minimize redundancy is to eliminate a large amount of the scale information. To capture all the information using wavelet transforms, a signal must be sampled at scales spaced logarithmically with a maximum spacing set by the wavelet being used. In general, if the scales are spaced by scales that increase by a factor of $\sqrt{2}$, all scales will be well sampled for all wavelets. Thus, the scales that should be sampled are $a_j = a_0 \cdot 2^{j/2}$, all the information should be retained. Here, a_0 is some minimum scale of the data under examination.

The reason for this logarithmic stepping in scale is that for $a_l \sim a_m$, the wavelet filters are smoothing on very similar scales and the difference is quite small. This redundancy can be quantified with the inner product between the two wavelet functions

$$\langle \psi(a_l, b_k = 0), \psi(a_m, b_k = 0) \rangle = \int_{-\infty}^{\infty} \psi(a_l, b_k = 0, t) \psi^*(a_m, b_k s = 0, t) dt$$

For $a_l = a_m$ this integral equals 1 (because of wavelet normalization) and falls off from 1 as a_m changes. This behavior is shown in Figure 7. The integral is often called the *overlap integral* and represents how similar the wavelet functions are, which in turn indicates how redundant the information is for the transforms with the two different filters. Since the overlap drops off as a function of the logarithm of the scale parameter, logarithmic sampling of scales is adequate for reconstruction.

In a similar fashion, for a fixed scale a_0 the spacings of b are redundant for very small steps of b compared to how fast the wavelet oscillates. This is because the wavelet smoothes the data on these scales and the sampling is redundant. To parameterize how "fast the wavelet oscillates," we calculate the *Fourier equivalent frequency* of the wavelet, which is defined as the maximum value of the Fourier power of the wavelet: $\omega_{eq} \equiv \max |\widehat{\psi}(\omega)|^2$. This is a sensible definition because it characterizes a wavelet by the dominant frequency of its oscillatory components. This defines the *Fourier wavelength* of the wavelet by $\lambda_f \equiv 2\pi/\omega_{eq}$. For the Morlet and DOG wavelets, the Fourier wavelengths are

$$\lambda_{Mor} = \frac{4\pi a}{\omega_0 + \sqrt{2 + \omega_0^2}} \text{ and } \lambda_{DOG} = \frac{2\pi a}{\sqrt{m + 1/2}}$$

On scales significantly smaller the λ_f, the data are redundant since the wavelet smoothes over these scales. A wavelet should be sampled with $\delta b = \lambda_f/2$, which is the analogue of the Nyquist criterion. The spacings of these points indicate that many values of b must be sampled for small values of a, but for large values of a, only a few values of b are needed for all information.

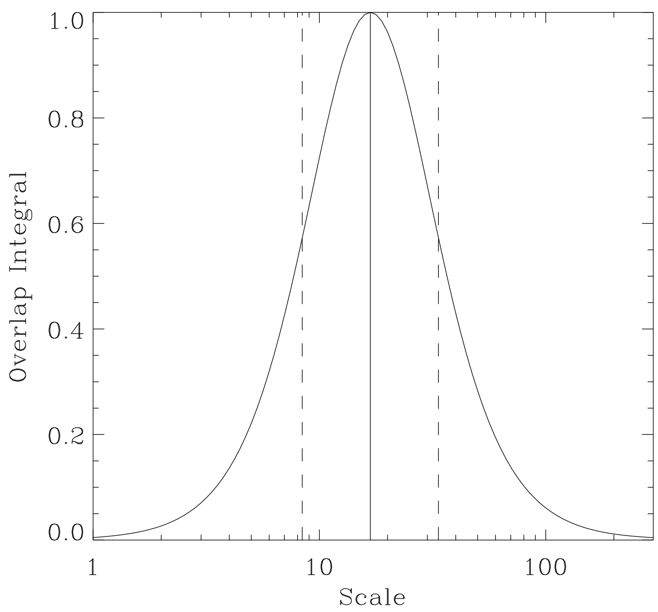

Fig. 7.— The overlap integral, $\langle\psi(a_l, b=0), \psi(a_m, b=0)\rangle$, for a DOG wavelet with $m=2$. The scale a_l is fixed at the value indicated by the solid vertical line. The dashed vertical lines indicate $a_m = a_l/2$ and $a_m = 2a_l$ where the overlap has fallen off significantly.

4. The Discrete Wavelet Transform

As noted previously, continuous wavelets take a 1-D signal and make it into a 2-D representation in wavelet space. This clearly introduces a lot of redundant information. In addition, most signals that we are exploring are discretely sampled, as is the case with real time series. At this point, we will change methodologies to explore the Discrete Wavelet Transform. Unlike most functional analysis where the discrete analysis is just an approximation, Discrete Wavelet transforms actually have more utility than the Continuous variety. To begin with, everything will look like the natural generalization of the continuous, mathematically grounded transforms that we've been discussing. As we progress, we'll move father away from this treatment.

Begin with a vector of data x_i. We'll assume that the data are well gridded, that is to say spaced by a uniform time interval δt. We'll start by considering continuous wavelets with closed form expressions, i.e. the wavelets we've been using. In this case, the wavelet coefficients are sampled on a grid of a and b values, with b being spaced every δt. Then

$$W(a_j, b_k\delta t) = \sum_{n=0}^{N-1} x_n \frac{1}{\sqrt{a_j}} \psi^* \left(\frac{(n-b_k)\delta t}{a_j}\right). \tag{8}$$

Since our class is focusing on IDL, we'll adopt the Fourier transform conventions of IDL to make this easy to implement. Other languages would use their own conventions.

$$\widehat{x_p} = \frac{1}{N} \sum_{n=0}^{N-1} x_n \exp\left(\frac{-2\pi i n p}{N}\right)$$

Then, the Discrete Fourier Transform can be used with the convolution theorem for a fast calculation of the coefficients.

$$W(a_j, b_k) = \sum_{p=0}^{N-1} \widehat{x_p} \sqrt{a_j}\, \widehat{\psi^*}\left(a_j \omega_p\right) \exp\left(i \omega_p b_k \delta t\right) \tag{9}$$

Here, ω_p is the vector corresponding to the frequencies sampled in the Discrete Fourier transform of x, namely $\omega_p = 2\pi p/N\delta t$ for $p \leq N/2$ and $\omega_p = 2\pi(p-N)/N\delta t$ for $p > N/2$.

The normalization to keep the wavelet at constant power for all values of the scale a_j is

$$\psi\left(\frac{(n-b_k)\delta t}{a_j}\right) = \sqrt{\frac{\delta t}{a_j}}\, \psi_0\left(\frac{(n-b_k)\delta t}{a_j}\right)$$

where ψ_0 is the mother wavelet.

4.1. Reconstruction Using Discrete Wavelets

Like the Fourier transform, the wavelet coefficients $W(a_j, b_k)$ can be used to reconstruct a signal from the constituent parts. However, since the wavelet transform is convolving the signal at some fixed minimum scale a_0, detail on levels smaller than a_0 will be lost. Even when sampled with minimum redundancy, the wavelet transform still contains more information than the original signal. As such, the inverse wavelet transform can actually rely upon a *different* function to reconstruct the data. The basic process involves reconstructing using the inverse transform (Equation 4) with a simpler function and a function that corrects for the difference between the original wavelet and the reconstruction function. This is a blatant statement without proof since the details are irksome (but available in Farge's 1992 article in the *Annual Review of Fluid Mechanics*). The utility of delving into this is that a much simpler function can be used to reconstruct the original signal, like the delta function. In the special case that the scales are chosen logarithmically, with a factor of $2^{\delta j}$ between each level, the data can be reconstructed with:

$$x_k = \frac{\delta j (\delta t)^{1/2}}{C_0 \psi_0(0)} \sum_{j=0}^{J_{max}} \frac{\mathrm{Re}\left\{W(a_j, b_k)\right\}}{\sqrt{a_j}},$$

where C_0 is a scaling constant that depends on the wavelet and $\psi_0(0)$ is the amplitude of the mother wavelet at $b_k = 0$. The Re operator indicates that the real part of the expression should be used. Note that in this reconstruction formula, the signal at any point is just a sum over the different scales that give rise to it.

Given the possibility of reconstruction, a wavelet decomposition seems to present the opportunity to filter the data. In short, a filter would consist of setting the power at a given scale to zero, thereby eliminating that scale from the reconstruction. In Fourier filtering, this is accomplished by setting unwanted frequency components to zero. One problem immediately crops up: unlike the Fourier basis functions, the wavelets that we have been using are *non-orthogonal* which prevents reconstruction. This means that the wavelets coefficients on different scales contain redundant information that, when summed into a reconstruction, interferes in precisely the correct fashion to return the original signal. If a scale is blindly set to zero, wavelets at different scales may contribute erroneous power in the reconstruction. The solution to this problem is to construct wavelets that are *orthogonal*, so that a single wavelet coefficient represents the total power of a given structure at a fixed scale. This proves to be difficult with the wavelets we are dealing with. There are *no* closed form expressions for continuous wavelets that are orthogonal for all a and b. Through judicious selection of a set of a grid of a_j and b_k, orthogonal wavelets can be constructed. This is reminiscent of a discrete wavelet transform; and indeed, we must operate in the discrete wavelet transform in order to make any progress.

5. Orthogonal Wavelets

In the continuous case, the statement of orthogonality of wavelets is just

$$\langle \psi(a_i, b_i, x), \psi(a_j, b_j, x)\rangle = \delta(a_i - a_j, b_i - b_j)$$

In the discrete case, the inner product between two functions is

$$\langle f(x_k), g(x_k)\rangle = \sum_{k=0}^{N-1} f(x_k) g^*(x_k).$$

This is just the analog of Equation 1 realized in the discrete case. Everything we said about the behavior of the inner product there should hold in the discrete case. A similar definition of orthogonality to the continuous case follows for the discrete case with $\langle \psi(a_i, b_i, x_k), \psi(a_j, b_j, x_k)\rangle = \delta_{a_i,a_j}\delta_{b_i,b_j}$. To illustrate the utility of orthogonal wavelets with respect to filtering, we can draw a parallel with vector spaces. In an orthogonal basis of a vector space, the dot product between two different basis vectors is zero. A vector is constructed as the sum of the projections onto the basis vectors times the basis vectors. For discrete functions with an orthogonal basis, the projection onto the ith basis function is just $\langle f, \psi(a_i, b_i)\rangle$ and the function is reconstructed much more simply than the non-orthogonal case: $f = \sum_{ij}\langle f, \psi(a_i, b_j)\rangle \psi(a_i, b_j)$. (Once again, this sum may look quite familiar if we re-write the inner product in the notation of quantum mechanics: $|f\rangle = \sum_{ij} |\psi_{ij}\rangle\langle\psi_{ij}|f\rangle$.) There is because there is no interference between terms in this sum because the overlap integrals (sums) between the orthogonal functions are zero (this is the definition of orthogonal). Filtering is accomplished by dropping "unwanted" terms from the sum and reconstructing. Since the basis functions are orthogonal, there is no excess information added in. This is highly analogous to Fourier filtering.

It remains to construct an orthogonal wavelet. So far, we've made no assumptions of continuity or differentiability, though all our wavelets have been smooth so far. The simplest orthogonal wavelets are not continuous. We shall begin our discussion with the Haar wavelet which is a discontinuous, orthogonal wavelet. The mother wavelet is defined on the interval between 0 and 1 as

$$\psi_0(\tau) = \begin{cases} 1, & 0 \le \tau < 1/2 \\ -1, & 1/2 \le \tau < 1. \end{cases}$$

The wavelet is well normalized in that $\int_{-\infty}^{\infty} \psi_0(\tau) d\tau = 0$ and $\int_{-\infty}^{\infty} |\psi_0(\tau)|^2 d\tau = 1$. The left-hand panel of Figure 8 plots the wavelet. The right-hand panel shows how the Haar wavelet is "interpolated" onto a fixed grid of points for the Discrete Wavelet Transform. The sampling of the function at the $\times$s is quite easy in this case!

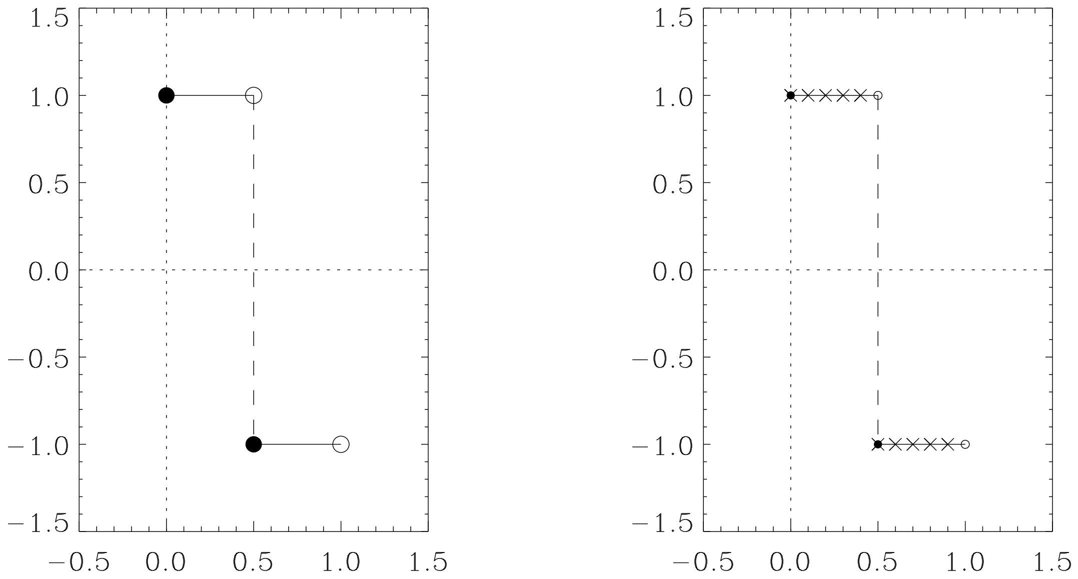

Fig. 8.— The Haar wavelet. The left-hand panel plots the wavelet as it is defined and the right-hand panel shows the ease of sampling the wavelet at discrete points, marked by $\times$s.

This wavelet is orthogonal to other Haar wavelets provided that for $\tau = (t-b)/a$, $a = 2^{-k}$ for some integer value of k and $b/a \equiv c$ is also an integer. The negative sign on k is just for the sake of notation later. For the mother wavelet case (i.e. $a = 1$) b must be an integer, so other Haar wavelets at this scale are just the mother wavelet translated by integer values so that the wavelets don't overlap. In this case, the product of one wavelet by another is identically zero for all values of t, showing these are orthogonal. For different scales, let's examine the inner product of ψ_0 with the wavelet that has $k = 1$ and $c = 1$ which has the form

$$\psi_{1,1}(t) = \sqrt{2}\begin{cases} 1, & 1/2 \leq t < 3/4 \\ -1, & 3/4 \leq t < 1. \end{cases}$$

We have used the notation $\psi_{k,c}$ to denote this wavelet where k and c are integers determining the scale and offset respectively. The inner product of this with the mother wavelet ($\psi_{0,0}$) is just the -1 (from $\psi_{0,0}$) times the integral of $\psi_{1,1}$ over the range [1/2,1) which is zero by inspection. For all values of k and c that are integers, the wavelets will be orthogonal. This example may be aided by studying some examples of the Haar wavelet for different values of k and c which appears in Figure 9.

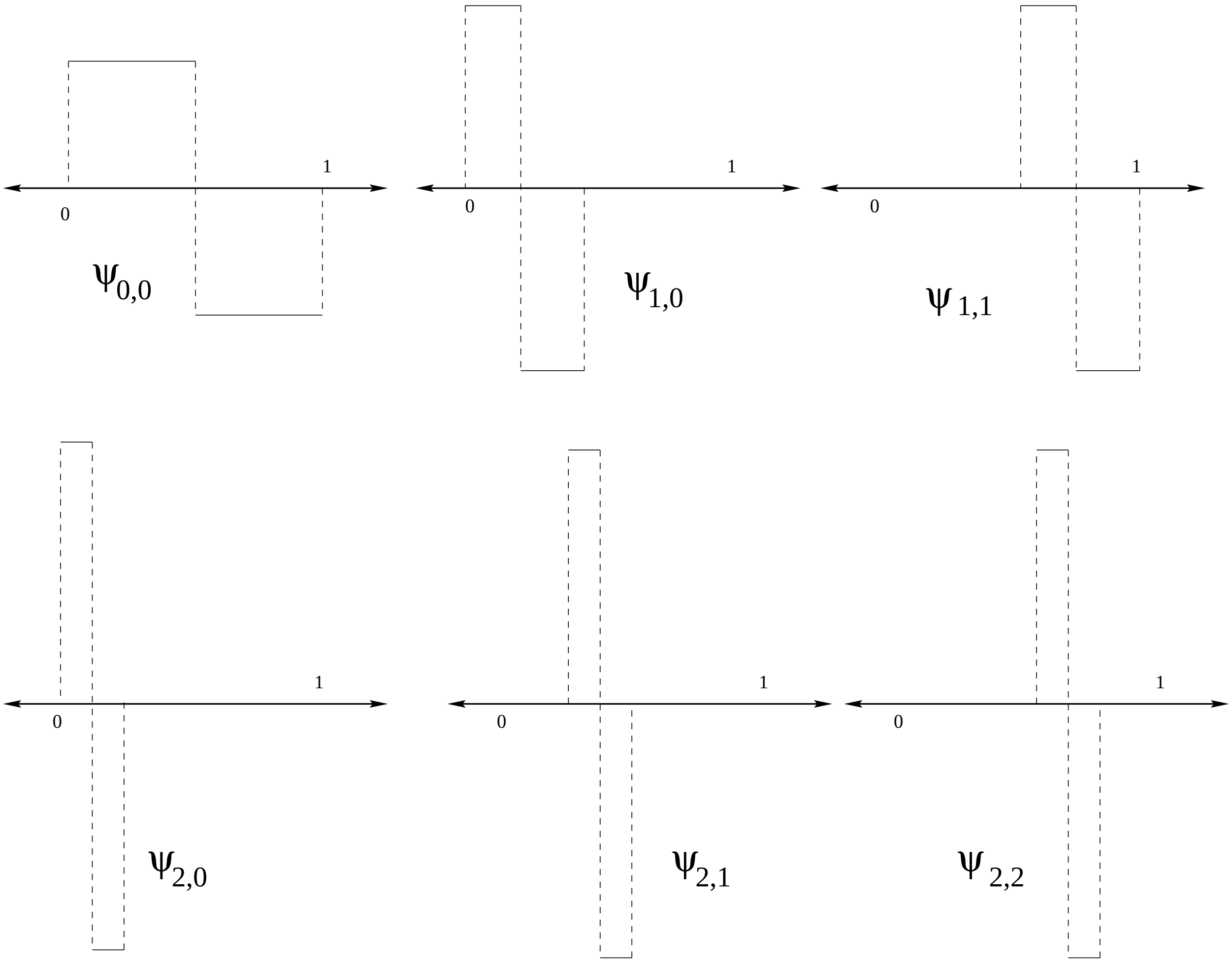

Fig. 9.— Examples of the Haar wavelet for different values of k and c. For larger k, the wavelet is narrower and taller. For larger values of c, the wavelet moves from 0 to 1. The wavelet $\psi_{2,3}$ is not shown but is necessary to complete the set of $\psi_{2,c}$.

Decomposition onto this basis is accomplished using the methods outlined above. That is to say, we just take the inner product of the function with each of the wavelets in our infinite basis. Restricting to the interval $[0, 1)$, any function *with zero mean* can be represented on this interval as

$$
\begin{aligned}
f(t) &= \sum_{k=0}^{\infty} \sum_{c=0}^{2^k-1} w_{k,c} \langle f(t), \psi_{k,c}(t) \rangle \\
&= w_{0,0} \int_0^1 f(t) \psi_{0,0} dt \\
&+ w_{1,0} \int_0^1 f(t) \psi_{1,0} dt + w_{1,1} \int_0^1 f(t) \psi_{1,1} dt \\
&+ w_{2,0} \int_0^1 f(t) \psi_{2,0} dt + w_{2,1} \int_0^1 f(t) \psi_{2,1} dt + w_{2,2} \int_0^1 f(t) \psi_{2,2} dt + w_{2,3} \int_0^1 f(t) \psi_{2,3} dt \\
&+ \sum_{k=3}^{\infty} \sum_{c=0}^{2^k-1} w_{k,c} \int_0^1 f(t) \psi_{k,c} dt.
\end{aligned}
$$

Since all the basis function have zero mean, there is no way that their linear combination can have a non-zero mean. To represent any given function, we need to add in an offset equal to the integral of the function across the interval under consideration, which can be thought of as the inner product of the function with a *scaling function* ϕ which is equal to 1 over the unit interval: $\langle f(t), \phi(t) \rangle = \langle f(t), 1 \rangle$. For other wavelets, the scaling function is different. The reason for this is that the wavelet and its scaling function form a pair such that one can be derived from the other. Thus a particular wavelet will have a scaling function determined by that wavelet. We won't articulate the precise relationship here, but we will point out where the scaling function occurs in the most useful cases.

So, now we know how to represent an arbitrary function as the sum of Haar wavelets. Let's apply this method to sample data. Using some clever matrix manipulations, we'll be able to vastly speed up the calculation of wavelet coefficients without having to resort to Fourier transforms. This results in the Fast Wavelet Transform which (when done right) is even FASTER than the Fast Fourier Transform. Like the FFT, let's assume we have 2^N data that we're transforming. Let's associate these data with the unit interval [0,1) so that each datum represents a step of 2^{-N} along this interval. We will need to consider a total of N scales with $2^{(N-1)}$ wavelets at each scale. This means there will be a total of 2^N wavelet coefficients. Immediately, this suggests that the redundancy we had seen in the previous cases is gone. Let's store the wavelet coefficients in a vector with the first element being the scaling coefficient $s_0 = \langle f(t), \phi \rangle$. Then, the wavelet transform can be represented as a matrix multiplication

$$\begin{bmatrix} s_0 \\ w_{0,0} \\ w_{1,0} \\ w_{1,1} \\ w_{2,0} \\ w_{2,1} \\ \vdots \end{bmatrix} = \underbrace{\begin{bmatrix} \longleftarrow & \phi & \longrightarrow \\ \longleftarrow & \psi_{0,0} & \longrightarrow \\ \longleftarrow & \psi_{1,0} & \longrightarrow \\ \longleftarrow & \psi_{1,1} & \longrightarrow \\ \longleftarrow & \psi_{2,0} & \longrightarrow \\ \longleftarrow & \psi_{2,1} & \longrightarrow \\ & \vdots & \end{bmatrix}}_{\equiv \boldsymbol{\Psi}} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ \vdots \end{bmatrix} \tag{10}$$

This equation is the analog of Equation 8 using a matrix multiplication to represent the transform. The matrix $\boldsymbol{\Psi}$ is the interpolation of the wavelet functions onto a discretely sampled grid. Because the Haar functions are so simple, the interpolation is relatively easy. In the case of $N = 3$, the transform matrix looks like

$$\boldsymbol{\Psi} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ \sqrt{2} & \sqrt{2} & -\sqrt{2} & -\sqrt{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \sqrt{2} & \sqrt{2} & -\sqrt{2} & -\sqrt{2} \\ 2 & -2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & -2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & -2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 & -2 \end{bmatrix}$$

The values of each row are increasing $\{1, \sqrt{2}, 2, \cdots, 2^{N/2}\}$ because of the normalization conditions. Since there are 8 points ($N = 3$), each point represents 1/8 of the interval between 0 and 1. Thus for a matrix with element $a_{i,j}$ in row i and column j, each row must have $(\sum_i a_{i,j}^2)/8 = 1$ and every row but the first has $\sum_i a_{i,j} = 0$.

Surprisingly, this matrix is actually an orthogonal matrix so that its transpose is its inverse. Thus, the inverse wavelet transform can be accomplished by multiplying the vector of wavelet coefficients $\mathbf{w}$ by the inverse matrix:

$$\mathbf{x} = \boldsymbol{\Psi}^T \mathbf{w}$$

Maybe this isn't so surprising since each row of the matrix is an orthogonal vector to all the other rows because of the way it was constructed.

5.1. Implementation of the Haar Wavelet Transform

What makes orthogonal wavelet transforms quick to implement is the recognition that the transform matrix $\boldsymbol{\Psi}$ can be broken down into a series of matrix multiplications that are themselves very quick to implement computationally. There are two steps for each scale considered and thus, in this simple implementation, there are $2N$ matrix multiplications. The method works from the smallest scales to the largest scales. At each scale you multiply the data vector by a *convolution matrix* and a *permutation matrix.*

For the Haar wavelet, the convolution matrix looks like this:

$$\mathbf{C}_0 = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 & & & & & \\ 1 & -1 & & & & & \\ & & 1 & 1 & & & \\ & & 1 & -1 & & & \\ & & & & 1 & 1 & \\ & & & & 1 & -1 & \\ & & & & & & \ddots \end{bmatrix}$$

When this matrix operates on a data vector, the first row of each block smoothes the two elements together, so this is called a *smoothing row* and the resulting scalar is called a *smooth component.* The second row records the difference between each pair of elements, and is called called the *detail row* and the resulting scalars are called the *detail components.* The resulting vector contains the smooth and detail components interleaved. The permutation matrix operates to move all the smooth components into the first part of the array and the detail components in the latter half of the array. While this is best accomplished numerically by manipulation of the vector indices, for the sake of completeness, the permutation matrix has the form

$$\mathbf{P}_0 = \begin{bmatrix} 1 & 0 & 0 & \cdots & & & \\ 0 & 0 & 1 & 0 & \cdots & & \\ 0 & 0 & 0 & 0 & 1 & 0 & \cdots \\ & & & \vdots & & & \\ 0 & 1 & 0 & 0 & \cdots & & \\ 0 & 0 & 0 & 1 & 0 & \cdots & \\ & & & \vdots & & & \end{bmatrix}$$

If our array is 0 indexed (e.g. as in IDL), this puts the 0th, 2nd, 4th ... elements in the 0th, 1st, 2nd, ... elements of the new array and 1st, 3rd, 5th, etc. elements into the $(2^{N/2})$th, $(2^{N/2}+1)$th, etc. elements of the new array. The process is then repeated with convolution matrix $\mathbf{C}_1$ and permutation P_1 which operate only on the first half (smooth components) of the resulting vector.

Thus, $\mathbf{C}_1$ has the following structure.

$$\mathbf{C}_1 = \begin{bmatrix} \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 & & & \\ 1 & -1 & & & \\ & & 1 & 1 & \\ & & 1 & -1 & \\ & & & & \ddots \end{bmatrix} & \mathbf{0} \\ \mathbf{0} & \mathbb{I}_{2^{N/2}} \end{bmatrix}$$

and the permutation matrix similarly splits into blocks of size $2^{N/2}$. $\mathbb{I}_{2^{N/2}}$ is the identity matrix with size $N/2 \times N/2$.

This process is iterated N times so that

$$\mathbf{w} = \underbrace{\mathbf{P}_{N-1}\mathbf{C}_{N-1}\mathbf{P}_{N-2}\mathbf{C}_{N-2}\cdots\mathbf{C}_1\mathbf{P}_0\mathbf{C}_0}_{\mathbf{\Psi}}\,\mathbf{x}$$

As indicated by the brace, the product of all the matrices gives the wavelet matrix. This suggests how to build the matrix in the discrete wavelet transform from simple matrices. However, the results of such an operation may be far from intuitive. It helps to view the process step by step. At each level, operation by the matrix $\mathbf{C}$ produces smooth components of the data alternating with detail components of the data. The matrix $\mathbf{P}$ separates them and the process is repeated on the smooth component. In what follows, the smoothed components are given by s's with the data after the first smoothing being s_i and the data after the second smoothing being S_i, and the detail components given values with d's – d_i after the first detail convolution and D_i after the second detail convolution. Then, the whole process can be viewed schematically as:

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \\ y_8 \end{bmatrix} \xrightarrow{\text{Convolve}} \begin{bmatrix} s_1 \\ d_1 \\ s_2 \\ d_2 \\ s_3 \\ d_3 \\ s_4 \\ d_4 \end{bmatrix} \xrightarrow{\text{Permute}} \begin{bmatrix} s_1 \\ s_2 \\ s_3 \\ s_4 \\ \hline d_1 \\ d_2 \\ d_3 \\ d_4 \end{bmatrix} \xrightarrow{\text{Convolve}} \begin{bmatrix} S_1 \\ D_1 \\ S_2 \\ D_2 \\ \hline d_1 \\ d_2 \\ d_3 \\ d_4 \end{bmatrix} \xrightarrow{\text{Permute}} \begin{bmatrix} S_1 \\ S_2 \\ \hline D_1 \\ D_2 \\ d_1 \\ d_2 \\ d_3 \\ d_4 \end{bmatrix} \tag{11}$$

This gives a sense of what data end up where after this kind of Discrete Wavelet Transform.

Unfortunately, we've replaced a matrix multiply of a $2^N \times 2^N$ matrix by a vector of length 2^N with $2N$ such matrix multiplies. Given that the original state wasn't so hot, the later state seems to have only exacerbated the problem. Fortunately, this is actually the path of success. The general class of operations described in the $\mathbf{P}$ and $\mathbf{C}$ matrices are very fast to implement on computers. However, it is rather tricky. This process is laid out in *Numerical Recipes*; and if you study it carefully (§13.10), you will note that the *partial wavelet transform* (one convolution and one permutation) is actually implemented by doing bit shifting operations on the indices of the array. This can cause headaches. However, when implemented in this supremely clever fashion, the wavelet coefficients can be calculated in $\mathcal{O}(n)$ steps as opposed to the fast Fourier transform which requires $\mathcal{O}(n \log n)$ steps. The reason why this is important is not because you need to really understand what's going on with the guts of the fast wavelet transform, but rather if you try to read *Numerical Recipes* you'll understand why things look so spooky.

5.2. Filtering with the DWT

Filtering is straight forward from this point. As mentioned previously, with an orthogonal basis, you can filter by setting the wavelet coefficients of unimportant scales to zero or significantly reducing them. An example of this filtering is shown in Figure 10. We return to our noisy data set with two Gaussians injected (Figure 6) . The data vector is 2^{10} elements long so we apply the partial wavelet transform 10 times, in reducing the vector to wavelet coefficients. These coefficients are plotted in the left-hand panel of Figure 10. We set all coefficients with wavelet *amplitude* less than 3σ to 0 and perform the inverse transform. The filtered data set appears in the right-hand panel of Figure 10. Instead of the Haar wavelet, this wavelet transform uses the Coiflet wavelet coefficients of degree 3 since they produce wavelets that are smoother and shaped more like a Gaussian. A Coiflet wavelet of degree 3 appears in Figure 11. See §5.3 for more discussion about how to actually implement the Coiflet wavelet.

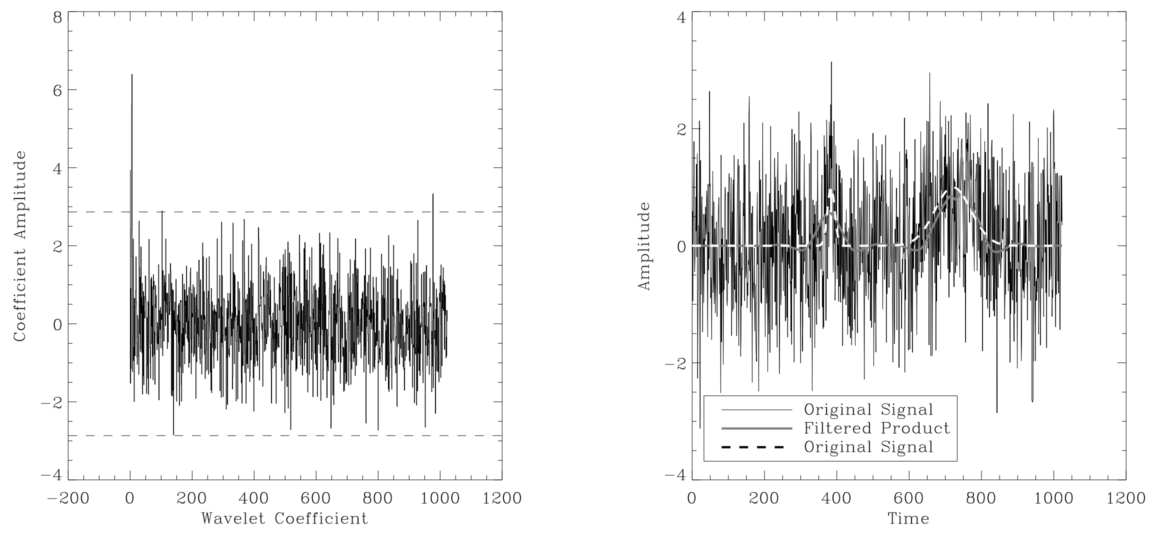

Fig. 10.— The left panel plots wavelet coefficients of a simulated signal using the Coiflet discrete wavelet transform of degree 3. The lines indicate the clipping levels used in the data. The right-hand panel plots the original test signal in black, the true signal in red and the reconstructed signal in blue.

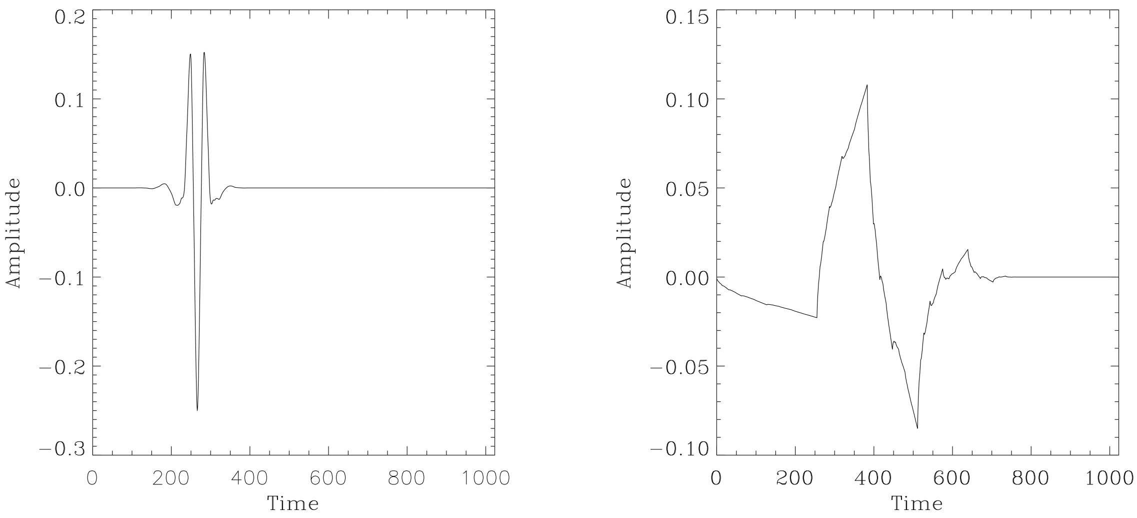

Fig. 11.— Wavelet Examples. The left-hand panel shows a Coiflet wavelet of degree 3 used in the filtering of Figure 10. The right-hand panel shows a Daubechies wavelet with 4 coefficients (see §5.3)
.

5.3. Other Wavelets Families

The Haar wavelet is the simplest wavelet to be implemented in this fashion. Many other wavelet families are produced by using different entries in the convolution matrix. These families are significantly better at analyzing smooth signals but we neglected them for the sake of simplicity. One of the most commonly seen wavelets is the Daubechies wavelet with 4 coefficients:

$$\mathbf{C}_0 = \begin{bmatrix} c_0 & c_1 & c_2 & c_3 & & & \\ c_3 & -c_2 & c_1 & -c_0 & & & \\ & & c_0 & c_1 & c_2 & c_3 & \\ & & c_3 & -c_2 & c_1 & -c_0 & \\ & & & & & & \ddots \end{bmatrix}$$

The constants c_i are given by

$$c_0 = (1+\sqrt{3})/4\sqrt{2}, \quad c_1 = (3+\sqrt{3})/4\sqrt{2}, \quad c_2 = (3-\sqrt{3})/4\sqrt{2}, \text{ and } c_3 = (1-\sqrt{3})/4\sqrt{2}$$

This peculiar set of numbers is the solution to the equations

$$c_0^2 + c_1^2 + c_2^2 + c_3^2 = 1 \text{ and } c_2 c_0 + c_3 c_1 = 0$$

which are required for orthogonality of the matrix, and the numbers also have two moments which vanish:

$$c_3 - c_2 + c_1 - c_0 = 0 \text{ and } 0c_3 - 1c_2 + 2c_2 - 3c_0 = 0.$$

The first equation is the zeroth moment of the wavelet coefficients (which corresponds to the second row of the matrix, see below). The second equation is the first moment, with each wavelet coefficient multiplied by its position in the sum. The second moment would just have the position in the sum squared multiplied by each coefficient and so on. The number of vanishing moments is intrinsically related to how smooth the resulting wavelet is. This wavelet, since it has 4 coefficients is called the Daubechies 4 wavelet. You might wonder what this wavelet looks like.

It's fairly straight forward to produce a wavelet simply by putting 1 in the appropriate position in the wavelet coefficient matrix and inverse transforming. This will produce the wavelet corresponding to that entry in $\mathbf{w}$. The correct position in the matrix corresponds to the first "detail" coefficient in the final wavelet array (the D_1 in Equation 11). In general, if there are n wavelet coefficients, the 1 should go in the entry of the array that's the next largest power of 2 greater than or equal to n. For example for the Daubechies wavelet of order 6, the 8th element should be 1. To determine the scaling function, the first element of the array should be set to 1 before the inverse wavelet transform is performed.

The Daubechies 4 wavelet appears in the right-hand panel of Figure 11. A Coiflet wavelet of degree 3 (Figure 11, left) has 18 wavelet coefficients. If all you are given is a set of these wavelet coefficients, the matrix $\mathbf{C}_0$ can be built up algorithmically. The coefficients are inserted in the first row of $\mathbf{C}_0$, beginning in the first column. Then, the next row is filled with the coefficients in *reverse* order with every other coefficient multiplied by -1. The next row is the same as the first row, except it is padded in front by two columns of zeros. The rows are often called the *smoothing row* (for the original coefficients) and the *detail row* (for the reversed, modulated coefficients). The zeros are inserted so that the start of the smoothing row always falls on the diagonal. The smoothing row represents the scaling function, ϕ, sampled with the smallest possible number of points to retain all the properties of the wavelet and the detail row represents the wavelet,ψ, sampled with the smallest number of points. This simple algorithm illustrates an informal relationship between the two functions, but the actual relationship is buried deeper in mathematics.

6. Higher Dimensions

To this point, we've dealt exclusively with wavelets in 1 dimension which is fine for some simple analysis, but astronomy relies upon multidimensional data. There are two ways to generalize our results. Returning to the continuous form of the transform for a moment, it's possible to write down the inner product metric in higher dimensions:

$$\langle f, g \rangle = \int_V f(\mathbf{r})g(\mathbf{r})dV$$

With this definition, everything follows *mutatis mutandi* (changing what needs to be changed) to reproduce all the results in the Continuous wavelet transform section. It's also possible to make a wavelet that's not isotropic and this introduces angular dependence so that the wavelet coefficients include a function of angle rather than just scale and offset.

To use the DWT, the data will be contained in a p dimensional array with each dimension having a length that is a power of two. The wavelet transform is inherently a one dimensional operation. The wavelet transform is simply applied to each dimension successively, and the order does not matter! Thus, to transform the columns of a two-dimensional matrix $\mathbf{M}$, instead of just a column vector, the wavelet transform is just

$$\mathbf{W_c} = \mathbf{\Psi M}$$

since matrix multiplication operates independently on each of the columns. The resulting wavelet coefficients have been subscripted with a $\mathbf{c}$ to indicate that the columns have been transformed. Then, the rows of the matrix must be transformed. To bring the matrix $\mathbf{\Psi}$ to bear on the rows, the resulting matrix $\mathbf{W_c}$ needs to be transposed since $\mathbf{\Psi}$ operates on columns and then transposed back. So, the final wavelet coefficients $\mathbf{W}$ are computed by

$$\mathbf{W} = \left[\mathbf{\Psi}\left(\mathbf{\Psi M}\right)^T\right]^T.$$

If you recall a property of matrix algebra, $(\mathbf{ab})^T = \mathbf{b}^T\mathbf{a}^T$, then this last equation becomes more illuminating:

$$\mathbf{W} = \left[\mathbf{\Psi M}^T\mathbf{\Psi}^T\right]^T = \mathbf{\Psi M \Psi}^T = \mathbf{\Psi M \Psi}^{-1}.$$

This is a diagonalization of matrix $\mathbf{M}$ with respect to the basis vectors in $\mathbf{\Psi}$, also known as a projection onto the basis vectors. With this identification, the parallel between vector geometry and functional analysis has come full circle! Also note that it does not matter whether you transform the rows or the columns first, which you can check with the methods used above.

Unfortunately, matrix multiplication doesn't generalize to higher dimensions, so the matrix methods in the DWT don't follow easily. Instead, the general strategy is to view a higher dimensional array as being comprised of a bundle of individual vectors of data and to apply the wavelet transform to each one individually. Then the matrix is transposed using the generalized idea of transposition where the vectors along one dimension are permuted into another.

6.1. Using the 2-D Discrete Wavelet Transform

Using the methods described above, let's do a 2-D wavelet transform. Starting with a simple image which consists of a 2-D Gaussian (Figure 12), we perform a Coiflet transform of order 3, first along the rows and then the columns. The resulting image contains the wavelet coefficients that represent the Gaussian. The power in both domains is the same, much as it is for the 1-D wavelet transform and we can check this for the image that

$$\sum_{i,j}\mathbf{M}_{ij}^2 = \sum_{i,j}\mathbf{W}_{ij}^2$$

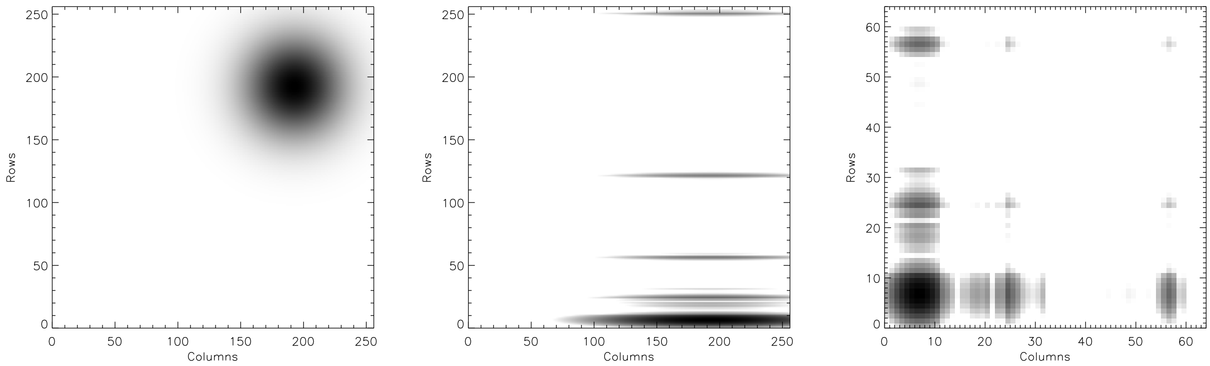

Fig. 12.— The left panel shows an input image of a Gaussian in gray scale. Note that the axes of the image are inverted along the y-axis with respect to how we usually write matrices (but in accordance with how we write coordinate axes). The middle panel shows the image after wavelet transforming each column in the image. The right-hand panel shows a subsection of the transformed image after transforming along the rows. The middle and right-hand panels are shown with a logarithmic color stretch. Note the information in the image is compacted into a small number of wavelet coefficients.

In this case, filtering follows the same pattern as was done for the 1-D case. First, we generate a noisy image of the Gaussian by adding randomly distributed noise with variance σ^2 to every pixel so the peak signal to noise level in the image is 1. The image is then wavelet transformed with a Coiflet of order 3 and wavelet coefficients with an absolute value less than 4σ are set to zero. This is because there are 256^2 pixels in the image so we expect only a few 4σ outliers, but many 3σ outliers. The coefficients are then transformed back to filtered image. The process is shown in Figure 13.

7. Image Compression

Back at the beginning of this document, I mentioned that wavelets could be used for the compression of images. This is because they are good at encoding rather detailed information in a few elements. Filtering and compression have a lot to do with each other. In the case of filtering, you want to move to the wavelet domain to increase the significance of your signal relative to the noise. In compression, you want to move to the wavelet domain so that you can identify the most important characteristics in the image: those that have the most power in the wavelet domain. To compress an image, you just keep some fraction of the "important" components of the image. There are two methods of compression. In both cases, the wavelet coefficients are ranked according to power (the square of the coefficient). In the first case, the top n coefficients are retained and the remainder are set to zero. In the second case, the precision of the coefficients is reduced (i.e. floating point numbers would be reduced to integers). Both of which achieve a substantial compression of

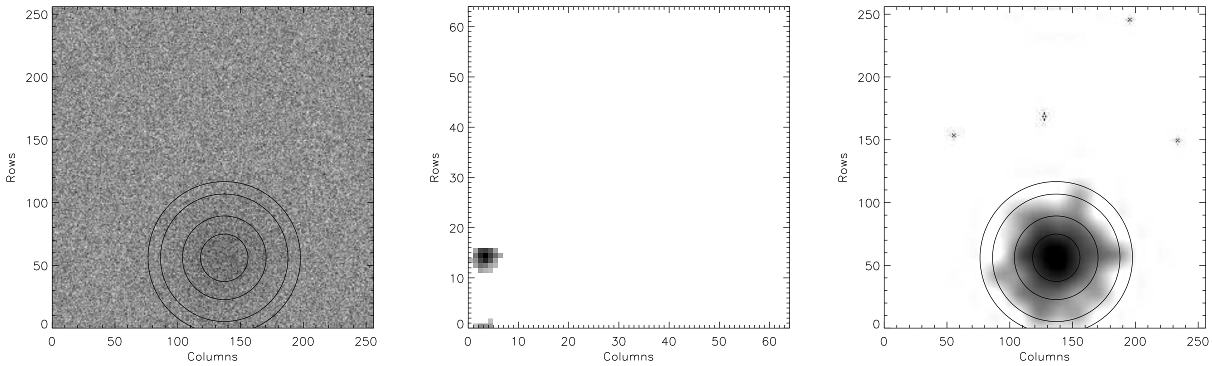

Fig. 13.— The left panel shows an input image of a Gaussian in gray scale with noise added. The Gaussian *is* in there! Contours show the shape of the Gaussian with the lowest being 0.1 times the peak value. The middle panel shows a subsection of the wavelet image indicating where the Gaussian maps to in the wavelet domain. The right-hand panel shows the image after the filtering is complete with the same contours to indicate the original shape of the Gaussian. The small features are from noise with enough power to avoid being filtered.

the data.

Note that this isn't particular to wavelets. The Fourier transform can be used for compression as well. Some image compression algorithms use the Discrete Cosine Transform to extract out filtered information, like the JPEG standard. The JPEG-2000 standard uses wavelets instead. The degree of compression is determined by the fraction of the wavelets you keep. As an example, Figure 14 shows an image of a distinguished astrophysicist. The middle panel shows the image reconstructed from top 1% of the wavelet coefficients ranked by power and the right-hand panel shows the image reconstructed from 10% of the wavelet coefficients. The quality of reconstruction is significantly better in the latter case and shows that there really isn't much information in Figure 14. But you probably knew that anyways...

A. Reference Material

For the sake of completeness, here are the coefficients for the Coiflet wavelet of order 3 that we've been using so much:

```
-0.0037935129, 0.0077825967, 0.023452696, -0.065771911
-0.061123388, 0.40517692, 0.79377722, 0.42848349, -0.071799822,
-0.082301927, 0.034555029, 0.015880545, -0.0090079761, -0.0025745177
0.0011175187, 0.00046621695, -7.0983303e-05, -3.4599774e-05
```

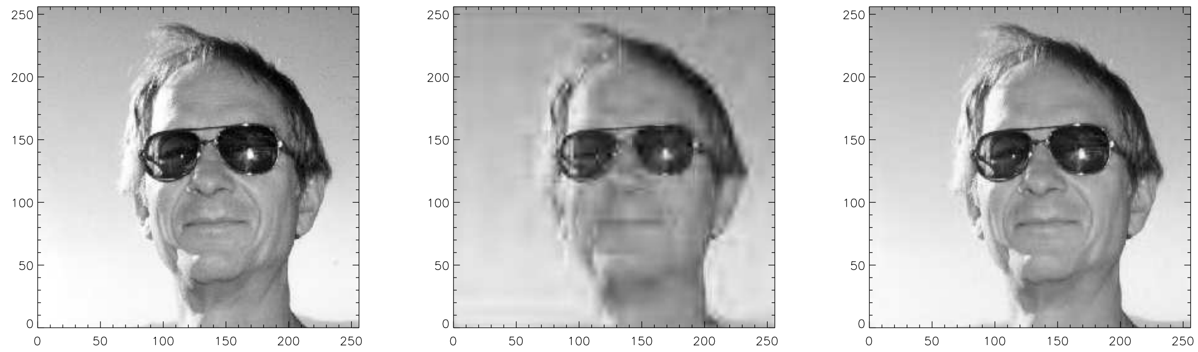

Fig. 14.— Demonstration of Image compression using the Coiflet wavelet. The left-hand panel shows the original image, the middle panel shows a compression of 100× and the right-hand panel shows a compression of 10×.

REFERENCES

Farge, M. 1992, Annual Review of Fluid Mechanics, 24, 395

Graps, A. 1995, "An Introduction to Wavelets" in IEEE Computational Science and Engineering, vol 2., num. 2.

Press, W. H., Teukolsky, S. A., Vetterling, W. T., & Flannery, B. P. 1992, Numerical recipes in C: the art of scientific computing. Cambridge: University Press, ©1992, 2nd ed., §13.10

Torrence, C. & Compo, G. P. 1998, Bulletin of the American Meteorological Society, vol. 79, Issue 1, pp.61-78, 79, 61