1 Continuous Wavelets - Nathan Williams

The wavelet transform is a technique to represent a function in terms of simpler "basis" functions; at a basic conceptual level it is much like the Taylor expansion or Fourier transform of a function. Indeed, much like the Fourier transform, it has many important applications in compression and signal processing. It would be irresponsible for us to launch into the theory of wavelets without at least a brief overview of the background material required, and so we have split the necessary definitions and results across several introductory sections. The first section gives a few relevant facts pertaining to $L^p$ spaces, while the second covers the theorems we will need from Fourier analysis. We then provide some motivation for the study of wavelets by appealing to Heisenberg’s uncertainty principle, introduce the windowed Fourier transform, and finally consider continuous wavelet transforms. We will not cover important wavelet construction tools such as multiresolution analysis in this paper.

1.1 $L^p$ Spaces

We will review here, without proof, a few important properties of $L^p$ spaces that will be used when we discuss wavelets. In this paper, we will exclusively use the Lebesgue measure and integral, and functions will be assumed to be from $\mathbb{R} \to \mathbb{C}$. For more detail, we refer the reader to [2], [4], and [5].

Definition 1 Let $p > 0$ be a real number or $+\infty$. $L^p(I)$ is the space of measurable functions $f$ for which $|f(t)|^p$ is integrable over the interval $I$. When
$p \geq 1$, these spaces form complete normed linear spaces with the following norms:

- $\|f\|_p = (\int_I |f(t)|^p dt)^{\frac{1}{p}}$ for $p < +\infty$
- $\|f\|_p = \inf \{c|\text{measure}\{x|f(x)| \geq c\} = 0\}$ for $p = +\infty$

For our applications we will be concerned only with $L^1$, $L^2$, and $L^\infty$. We will use the notation $C^p(I)$ to denote the space of functions that are $p$ times continuously differentiable on the interval $I$.

**Theorem 1** (Hölder’s inequality) Let $f \in L^p(I)$ and $g \in L^q(I)$ with $\frac{1}{p} + \frac{1}{q} = 1$. Then $fg \in L^1(I)$, and $\int_I |f(t)g(t)|dt \leq \|f\|_p \|g\|_q$ (the case where $p = q = 2$ is called the Cauchy-Schwarz inequality).

**Theorem 2** (Fubini’s Theorem) If $f$ is measurable and $\int \int f(x,y)dA < \infty$, then the order of integration can be interchanged and $\int \int f(x,y)dA = \int \int f(x,y)dydx = \int \int f(x,y)dx$. 

### 1.2 The Fourier Transform

As the Fourier transform is central to understanding the wavelet transform, we will cover without proof certain elementary properties. For proofs and an introduction to the Schwartz space which allows the extension of the transform from $L^1$ to $L^2$, we refer the reader to [2] and [4]. Essentially, the Fourier transform takes a function in the time domain and returns its spectrum, an associated function in the frequency domain. We will make use of the definition of the Fourier transform used in signal processing; the pure mathematics definition would use the conventions for angular frequency.

**Definition 2** Let $f \in L^1(\mathbb{R})$. Then the Fourier transform $\hat{f}$ or $\mathcal{F}[f]$ of $f$ is given by $\hat{f}(\gamma) = \int_\mathbb{R} e^{-2\pi i \gamma t} f(t)dt$ with $\|f\|_\infty \leq \|\hat{f}\|_1$. Furthermore, if $\hat{f} \in L^1(\mathbb{R})$, then $f(t) = \mathcal{F}[\hat{f}(\gamma) d\gamma]$. We can extend this definition to functions in $L^2$ using a limiting process such that it agrees with the Fourier transform on functions in $L^1 \cap L^2$ and where convergence of the inverse transform is taken in the $L^2$ sense.

**Theorem 3** (Parseval’s Formula) If $f, g \in L^1(\mathbb{R}) \cap L^2(\mathbb{R})$, then $\int_\mathbb{R} \hat{f}(\gamma)\overline{\hat{g}(\gamma)}d\gamma = \int_\mathbb{R} f(t)\overline{g(t)}dt$. In particular, $\|\hat{f}\|_2 = \|f\|_2$ (this is called Plancherel’s Formula, and shows that the Fourier transform conserves energy).
Theorem 4 The following are properties of the Fourier transform.

- \( \hat{f}^k(\gamma) = (2\pi i \gamma)^k \hat{f}(\gamma) \), where \( f^k \) is the \( k \)th derivative of \( f \)
- \( \mathcal{F}[f(x - a)] = e^{-2\pi i a \gamma} \hat{f}(\gamma) \)
- \( \mathcal{F}[f(ax)] = \frac{1}{|a|} \hat{f}(\frac{\gamma}{a}) \), for \( a \neq 0 \)

1.3 Heisenberg’s Uncertainty Principle

It is natural at this point to question the necessity of another method of representing functions; after all, Fourier series have proved exceedingly successful in many applications. In order to provide some motivation for different techniques, we will consider the uncertainty principle—a well-known, popularized, and often abused result from physics. It is perhaps most commonly stated as ”one can’t know both the position and the momentum of a particle to arbitrary precision.” We will make this mathematically precise in this section as a statement about functions.

Definition 3 Given a function \( f \in C^1(\mathbb{R}) \) with \( \|f\|_2 = 1 \) and \( \lim_{x \to \infty} x |f(x)|^2 = 0 \), let

- \( \sigma_f^2 = \int_{\mathbb{R}} x^2 |f(x)|^2 \, dx \) be the energy dispersion of \( f \) in time, and
- \( \sigma'_f^2 = \int_{\mathbb{R}} \gamma^2 |\hat{f}(\gamma)|^2 \, d\gamma \) be the energy dispersion of \( f \) in frequency.

Note that in quantum mechanics, which is where this formula naturally arises, \( f \) will be the state function of a particle that lies on \( \mathbb{R} \) and \( \int_a^b |f(x)|^2 \, dx \) gives the probability that the particle’s position is in the interval \((a, b)\). The normalization requirement on \( \|f\|_2 \) ensures that this probability is well-defined—that is, the particle is somewhere in \( \mathbb{R} \) with probability 1. Considering the Fourier transform of \( f \), we obtain the corresponding probability \( \int_a^b |f(x)|^2 \, dx \) that the particle has momentum in \((a, b)\)–and Plancherel’s Formula takes care of the required normalization. We can think of \( \sigma_f^2 \) as the variance or uncertainty of the particle’s position about 0 of \( f \)–if \( f \) is spread out across \( \mathbb{R} \) this value will be large, while if it is concentrated around the origin it will be small. The same applies to momentum and \( \sigma'_f^2 \).

Theorem 5 (The Uncertainty Principle) Given an \( f \) that satisfies the above conditions, \( \sigma_f^2 \sigma'_f^2 \geq \frac{1}{16\pi^2} \)
Proof (adapted from [2], [3], and [4]). We begin with the first given, that 
\( \|f\|_2 = 1 \). Integrating by parts, we have

\[
1 = \int_{\mathbb{R}} |f(x)|^2 \, dx = \left[ x |f(x)|^2 \right]_{-\infty}^{\infty} - \int_{\mathbb{R}} x(|f(x)|^2)' \, dx = - \int_{\mathbb{R}} x(|f(x)|^2)' \, dx
\]

where the term \( [x |f(x)|^2]_{-\infty}^{\infty} \) has vanished because \( \lim_{x \to \pm \infty} x |f(x)|^2 = 0 \).

We can rewrite this as

\[
- \int_{\mathbb{R}} x(f(x)\overline{f(x)})' \, dx = - \int_{\mathbb{R}} x f'(x)\overline{f(x)} + x f(x)\overline{f'(x)} \, dx
\]

by the triangle inequality. We have now shown that \( 1 \leq 2 \int_{\mathbb{R}} |x| |f(x)||f'(x)| \, dx \).

Applying the Cauchy-Schwarz inequality yields

\[
1 \leq 2 \left( \int_{\mathbb{R}} x^2 |f(x)|^2 \, dx \right)^{1/2} \left( \int_{\mathbb{R}} |f'(x)|^2 \, dx \right)^{1/2}
\]

By Plancherel’s Formula and the differentiation theorem we see that

\[
\int_{\mathbb{R}} |f'(x)|^2 \, dx = \int_{\mathbb{R}} |\hat{f}'(\gamma)|^2 \, d\gamma = \int_{\mathbb{R}} |2\pi i \hat{f}(\gamma)|^2 \, d\gamma = 4\pi^2 \int_{\mathbb{R}} \gamma^2 |\hat{f}(\gamma)|^2 \, d\gamma = 4\pi^2 \sigma_f^2
\]

Plugging this into the above we get

\[
1 \leq 2 \left( \int_{\mathbb{R}} x^2 |f(x)|^2 \, dx \right)^{1/2} \left( \int_{\mathbb{R}} |f'(x)|^2 \, dx \right)^{1/2} = 4\pi \sigma_f \sigma_{\hat{f}}
\]

and we can therefore conclude that \( \sigma_f^2 \sigma_{\hat{f}}^2 \geq \frac{1}{16\pi^2} \).

In fact, the uncertainty principle holds for functions under less stringent conditions, but the proof is somewhat more involved.

Given a function over time, it is easy to read off time information—say, when large peaks occur. Frequency information, however, is relatively hidden. Conversely, when considering the Fourier transform of the function, frequency information is displayed, but it is difficult to determine for how
long and when the frequencies appear in time. Ideally, we would like a representation for our function that allows us to read off all of this information, much as musical notation tells the musician what notes to play when and for how long. The uncertainty principle shows us that the Fourier transform may not contain the information that we want in usable form, and so we will now consider the windowed Fourier transform and wavelet transforms.

1.4 The Windowed Fourier Transform

We can try to improve time and frequency localization by breaking our original function into many small pieces, and then taking the Fourier transform of each piece. Several of our algorithms make use of exactly this discrete technique, and will be discussed in more detail later in this paper. Continuously, however, we can take a carefully chosen window function (a common choice is the Gaussian to achieve the lower bound from the uncertainty principle) and define a family of windows that are translated and frequency modulated versions of that window. The windowed Fourier Transform is defined as the integral of each of these windows against the original function. Though the resulting function seems redundant (after all, we’ve taken a function that is indexed by one continuous variable and made it indexed by two), it is conceivable that by applying this process we might lose information—for example, we might overlook low frequency signals that span multiple pieces. The following theorem states that we can indeed reconstruct the original function from the transform and that the transform has the same energy as the original function.

More formally, we have the following theorem, due to Gabor [2].

**Theorem 6** Given $w \in L^1 \cap L^2$ with $|\hat{w}|$ an even function and $\|w\|_2 = 1$, let

- $w_{\lambda b}(t) = w(t - b)e^{2\pi i \lambda t}$, with $\lambda, b \in \mathbb{R}$, and
- $W_f(\lambda, b) = \int_{\mathbb{R}} f(t)\hat{w}_{\lambda b}(t)dt \forall f \in L^2$.

Then we have,

- An analogue of Plancherel’s formula about energy conservation holds:

  $$\int_{\mathbb{R}^2} |W_f(\lambda, b)|^2 d\lambda db = \|f\|_2^2$$
The original function can be reconstructed from the wavelet transform:

\[ f(x) = \int \int_{\mathbb{R}^2} W_f(\lambda, b) w_{\lambda b}(x) d\lambda db \]

Due to the similarity in proof between this and the analogous theorem for the continuous wavelet transform, we will only provide a proof for the wavelet transform.

1.5 The Continuous Wavelet Transform

In the windowed Fourier transform, the window functions are translated and can have different numbers of oscillations, but the size of the window is fixed. In other words, the energy dispersion in time is fixed and so the ability to localize in time is limited by the uncertainty principle. We now turn to a natural extension, the continuous wavelet transform, in which the window function can be translated, dilated, and contracted to gain additional resolution.

We consider a function \( \psi \in L^1 \cap L^2 \) such that

- \( \int_{\mathbb{R}} \frac{|\hat{\psi}(\lambda)|^2}{|\lambda|} d\lambda = K < + \infty \)
- \( \|\psi\|_2 = 1 \)

This \( \psi \) is called the mother wavelet. We can then create a doubly indexed family of wavelets by

\[ \psi_{ab}(t) = \frac{1}{|a|^{1/2}} \psi(\frac{t-b}{a}), a, b \in \mathbb{R}, a \neq 0 \]

In much the same way as with the windowed Fourier transform, we consider for \( f \in L^2 \) the wavelet coefficients

\[ C_f(a,b) = \int_{\mathbb{R}} f(t) \overline{\psi_{ab}(t)} dt \]

**Theorem 7** With these assumptions and definitions, we have the following.
• An analogue of Plancherel’s formula about energy conservation holds:

\[
\frac{1}{K} \int \int_{\mathbb{R}^2} \frac{|C_f(a, b)|^2}{a^2} dadb = \|f\|_2^2
\]

• The original function can be reconstructed from the wavelet transform:

\[
f(x) = \frac{1}{K} \int \int_{\mathbb{R}^2} \frac{C_f(a, b)\psi_{ab}(x)}{a^2} dadb
\]

Proof (taken from [2]). Using Parseval’s formula, we can write \(C_f(a, b)\) as follows

\[
C_f(a, b) = \int_{\mathbb{R}} f(t)\hat{\psi}_{ab}(t)dt = \int_{\mathbb{R}} \hat{f}(\lambda)\hat{\psi}_{ab}(\lambda)d\lambda
\]

Now, by properties of the Fourier transform we have

\[
\hat{\psi}_{ab}(t) = \int_{\mathbb{R}} e^{-2\pi i \lambda t} \frac{1}{|a|^{1/2}} \psi(t - \frac{b}{a})dt = \sqrt{|a|} e^{-2\pi i \lambda b} \hat{\psi}(a\lambda)
\]

Plugging this in to the above we get

\[
C_f(a, b) = \sqrt{|a|} \int_{\mathbb{R}} \hat{f}(\lambda)e^{-2\pi i \lambda b} \hat{\psi}(a\lambda)d\lambda = \sqrt{|a|} \int_{\mathbb{R}} e^{2\pi i \lambda b} \hat{f}(\lambda)\hat{\psi}(a\lambda)d\lambda = \sqrt{|a|} \mathcal{F}_{\lambda}[\hat{f}(\lambda)\hat{\psi}(a\lambda)](b)
\]

where the notation in the final equation refers to the inverse Fourier transform with respect to \(\lambda\).

The proof of the energy conservation equation is relatively simple, given this form of \(C_f(a, b)\). First, we can reverse the order of integration using Fubini’s theorem and the observation that \(C_f(a, b)\) is in \(L^2\) because we assumed \(\psi \in L^1\), so \(\hat{\psi}\) is bounded. Then we have

\[
\frac{1}{K} \int \int_{\mathbb{R}^2} \frac{|C_f(a, b)|^2}{a^2} dadb = \frac{1}{K} \int \int_{\mathbb{R}^2} \frac{|C_f(a, b)|^2}{a^2} dbda
\]

\[
= \frac{1}{K} \int \int_{\mathbb{R}^2} \frac{\sqrt{|a|} \mathcal{F}_{\lambda}[\hat{f}(\lambda)\hat{\psi}(a\lambda)](b)^2}{a^2} dbda = \frac{1}{K} \int \int_{\mathbb{R}^2} \frac{\mathcal{F}_{\lambda}[\hat{f}(\lambda)\hat{\psi}(a\lambda)](b)^2}{|a|} dbda
\]

Now, by Parseval’s formula we can remove the inverse Fourier transform, yielding
\[
\frac{1}{K} \int \int_{\mathbb{R}^2} \frac{|\hat{f}(\lambda)|^2 |\hat{\psi}(a\lambda)|^2}{|a|} d\lambda da
\]

We can again use Fubini’s theorem to interchange the order of integration

\[
\frac{1}{K} \int |\hat{f}(\lambda)|^2 \int_{\mathbb{R}^2} \frac{|\hat{\psi}(a\lambda)|^2}{|a|} d\lambda da
\]

By assumption, \( \int_{\mathbb{R}} \frac{|\hat{\psi}(\xi)|^2}{|\xi|} d\lambda = K \). By the change of variable \( \xi = a\lambda \), \( \int_{\mathbb{R}} \frac{|\hat{\psi}(a\lambda)|^2}{|a|} da = K \) and so

\[
\frac{1}{K} \int \int_{\mathbb{R}^2} \frac{|C_f(a,b)|^2}{a^2} d\lambda db = \int_{\mathbb{R}} |\hat{f}(\lambda)|^2 = \int_{\mathbb{R}} |f(\lambda)|^2
\]

The last equation follows again from Parseval’s formula, and we have therefore shown conservation of energy.

The proof of reconstruction is somewhat more complicated. To even make sense of what we mean by reconstruction, we will restate the theorem in more precise terms. If

\[
f_\epsilon(x) = \frac{1}{K} \int_{b \in \mathbb{R}} \int_{|a| \geq \epsilon} \frac{C_f(a,b)\psi_{ab}(x)}{a^2} d\lambda db
\]

then \( f_\epsilon \to f \) in the \( L^2 \) sense as \( \epsilon \to 0^+ \).

We will work from the inside out, using our expression for \( C_f(a,b) \). Let

\[
J(a) = \int_{\mathbb{R}} C_f(a,b)\psi_{ab}(x) db = \sqrt{|a|} \int_{\mathbb{R}} \mathcal{F}_\lambda[\hat{f}(\lambda)\hat{\psi}(a\lambda)](b)\psi_{ab}(x) db
\]

\[
= \sqrt{|a|} \int_{\mathbb{R}} \hat{f}(\lambda)\hat{\psi}(a\lambda)\mathcal{F}_b[\psi_{ab}(x)](\lambda) d\lambda
\]

where we have used Parseval’s formula to move the transform. By our earlier calculation of \( \hat{\psi}_{ab}(\lambda) \), we know that

\[
\mathcal{F}_b[\psi_{ab}(x)](\lambda) = \sqrt{|a|} e^{2\pi i \lambda b} \hat{\psi}(a\lambda)
\]

and so we can substitute in to obtain

\[
J(a) = |a| \int_{\mathbb{R}} \hat{f}(\lambda)|\hat{\psi}(a\lambda)|^2 e^{2\pi i \lambda b} d\lambda
\]
We will now add the second integral to the mix by defining
\[ g_\epsilon(x) = \int_{|a| \geq \epsilon} \frac{J(a)}{a^2} da = \int_{|a| \geq \epsilon} \frac{\hat{f}(\lambda)|\hat{\psi}(a\lambda)|^2 e^{2\pi i a\lambda} d\lambda}{|a|} \]

To use Fubini’s theorem and interchange the order of integration, we must show the this function is integrable on \((|a| \geq \epsilon) \times \mathbb{R}\). Reversing the order to check and substituting as before \(\xi = a\lambda\), we have
\[ A = \int_{\mathbb{R}} |\hat{f}(\lambda)| \int_{|a| \geq \epsilon} |\hat{\psi}(a)|^2 |a| da d\lambda = \int_{\mathbb{R}} |\hat{f}(\lambda)| \int_{|\xi| \geq |\lambda|} |\hat{\psi}(\xi)|^2 |\xi| d\lambda d\lambda \]

We will now break this integral into two parts to show that it is finite. If \(|\lambda| \leq 1\),
\[ A_1 = \int_{-1}^{1} |\hat{f}(\lambda)| \int_{|\xi| \geq |\lambda|} \frac{|\hat{\psi}(\xi)|^2 |\xi|}{|\lambda|} d\lambda d\xi \leq K \int_{-1}^{1} |\hat{f}(\lambda)| d\lambda \leq K\sqrt{2} \|f\|_2 \]

where the inequalities follow by the assumptions on \(\psi\) and the Cauchy-Schwarz inequality followed by Parseval’s formula. This is then finite because \(K\) is finite and \(f\) is in \(L^2\). Otherwise, when \(|\lambda| \geq 1\),
\[ A_2 = \int_{|\lambda| \geq 1} |\hat{f}(\lambda)| \int_{|\xi| \geq |\lambda|} \frac{|\hat{\psi}(\xi)|^2 |\xi|}{|\lambda|} d\lambda d\xi \leq \int_{|\lambda| \geq 1} \frac{|\hat{f}(\lambda)|}{|\lambda|} \int_{|\xi| \geq |\lambda|} |\hat{\psi}(\xi)|^2 |\xi| d\lambda d\xi \]
\[ = \frac{\|\psi\|_2^2}{\epsilon} \int_{|\lambda| \geq 1} \frac{|\hat{f}(\lambda)|}{|\lambda|} d\lambda \leq \frac{\|\psi\|_2^2 \|f\|_2}{\epsilon} \left(\int_{|\lambda| \geq 1} \lambda^{-2} d\lambda\right)^{1/2} < +\infty \]

where we have again applied the Cauchy-Schwarz inequality and used the fact that \(\int_{|\lambda| \geq 1} \lambda^{-2} d\lambda\) is finite. Thus, we can apply Fubini’s theorem and reverse the order of integration in our expression for \(g_\epsilon(x)\).
\[ g_\epsilon(x) = \int_{|a| \geq \epsilon} \frac{\hat{f}(\lambda)|\hat{\psi}(a\lambda)|^2 e^{2\pi i a\lambda} d\lambda}{|a|} = \int_{\mathbb{R}} \hat{f}(\lambda)e^{2\pi i \lambda x} \left(\int_{|a| \geq \epsilon} \frac{|\hat{\psi}(a\lambda)|^2}{|a|} da\right) d\lambda = \mathcal{F}[f\theta_\epsilon](x) \]
where
\[ \theta_\epsilon(\lambda) = \int_{|a| \geq \epsilon} \frac{|p\hat{\psi}(a\lambda)|^2}{|a|} da \]

We will show that \( g_\epsilon \to Kf \) (which is equivalent to showing that \( f_\epsilon \to f \)) in \( L^2 \).

\[ \|Kf - g_\epsilon\|^2 = \|\mathcal{F}[K\hat{f} - \hat{f}\theta_\epsilon]\|^2 = \|\mathcal{F}(K - \hat{f}\theta_\epsilon)\|^2 \]

by substituting our expression for \( g_\epsilon \) in and using Parseval’s formula. Then we have by writing out explicitly the norm

\[ \|Kf - g_\epsilon\|^2 = \int_\mathbb{R} |\hat{\hat{\psi}}(\lambda)|^2 |K - \theta_\epsilon(\lambda)|^2 d\lambda \]

We will again estimate the integral in two parts. If \( |\lambda| \leq \epsilon^{-1/2} \),

\[ \theta_\epsilon(\lambda) = \int_{|\xi| \geq \sqrt{\epsilon}} \frac{|\hat{\psi}(\xi)|^2}{|\xi|} d\xi \geq \int_{|\xi| \geq \sqrt{\epsilon}} \frac{|\hat{\psi}(\xi)|^2}{|\xi|} d\xi = K(\epsilon) \]

Now, by our assumption that \( \int_\mathbb{R} \frac{|\hat{\psi}(\lambda)|^2}{|\lambda|} d\lambda = K \), we have that \( 0 \leq K - \theta_\epsilon(\lambda) \leq K - K(\epsilon) \) which tends to 0 as \( \epsilon \to 0^+ \). On the other hand, if \( |\lambda| \geq \epsilon^{-1/2} \), we clearly have that \( 0 \leq \theta_\epsilon(\lambda) \leq K \). Looking at the norm of the different of \( Kf \) and \( g_\epsilon \) and expanding about these cases, we have

\[ \|Kf - g_\epsilon\|^2 = \int_{|\lambda| \leq \epsilon^{-1/2}} |\hat{\hat{\psi}}(\lambda)|^2 |K - \theta_\epsilon(\lambda)|^2 d\lambda \]

\[ \leq \int_{|\lambda| \leq \epsilon^{-1/2}} |\hat{\hat{\psi}}(\lambda)|^2 |K - K(\epsilon)|^2 d\lambda \]

\[ = K - K(\epsilon) \|f\|_2^2 + K^2 \int_{|\lambda| \geq \epsilon^{-1/2}} |\hat{\hat{\psi}}(\lambda)|^2 d\lambda \]

and by our earlier observations both terms tend to zero as \( \epsilon \to 0 \).
2 Examples of Discrete Wavelets - Amanda Brown

2.1 The Haar Wavelet

The simplest wavelet is the Haar wavelet. The Haar wavelet was created by Alfred Haar in 1910 and can be used to analyze a signal, which we can represent as an array of data. The Haar wavelet begins by setting the mother wavelet to be

\[
    h(x) = \begin{cases} 
        1 & \text{if } x \in [0, \frac{1}{2}) \\
        -1 & \text{if } x \in \left[\frac{1}{2}, 1\right) \\
        0 & \text{if } x \notin [0, 1) \end{cases}
\]

Then the family of wavelet functions is defined as \( h_n(x) = 2^{j/2} n(2^j x - k) \) with \( n \geq 1 \), \( h = 2^j + K \), \( j \geq 0 \), and \( 0 \leq k \leq 2^j \). So he has defined a function using the mother wavelet function that takes points in the sample and makes an orthonormal closed system out of them. The function is supported on the dyadic interval of \( I_n = [(k)2^{-j}, (k + 1)2^{-j}] \), which is \([0,1)\) when \( 0 \leq k < 2^j \). Then in order to complete the closed interval, Haar sets \( h_0(x) = 1 \) on \([0,1)\). Therefore \( h_n, h_1, \ldots, h_n, \ldots \) is a system of orthonormal functions. The set of functions is continuous and it uniformly converges to \( f(x) \) but by its construction the individual functions are not continuous by themselves. Another short coming of the Haar function, is that it is not useful for approximating all continuous functions - if \( f \) is continuous and has a continuous derivative the fitting approximation is one created by inscribing polygonal lines rather than the Haar approximation. \([7]\)

To illustrate the power and use of the Haar wavelet, it would be instructive to begin with a more simplified explanation of its construction. The easiest way to explain the actual mechanics of the Haar wavelet is to first demonstrate how the scaling and wavelet functions work on only a sample of points. The mother wavelet function, \( \psi \), is defined as

\[
    \psi_{[u,w]}(v) = \begin{cases} 
        1 & \text{if } u \leq r < v \\
        -1 & \text{if } v \leq r < w \\
        0 & \text{otherwise} \end{cases}
\]

Then the wavelet function, \( \psi \) is defined as \( \psi_{[0,1]} = \varphi_{[0,\frac{1}{2})} - \varphi_{(\frac{1}{2}, 1)} \). Therefore on the interval of \([0,1)\) the wavelet function will have a height of 1 in from...
[0, \frac{1}{2}) and -1 from \([\frac{1}{2}, 0)\) and 0 otherwise. Then we can expand the wavelet algorithm by generalizing to an interval other than the interval from 0 to 1 by defining the wavelet as

$$
\psi_{[u,w)}(v) = \begin{cases} 
1 & \text{if } u \leq r < v \\
-1 & \text{if } v \leq r < w \\
0 & \text{otherwise}
\end{cases}
$$

with \(v\) defined as the midpoint of the new interval of \(u\) to \(w\), \(v = \frac{u+w}{2}\), and now the wavelet definition is equal to the wavelet family defined previously.

Now the function for two adjacent steps at heights \(s_0\) and \(s_1\), is

$$
\tilde{f} := s_0 \cdot \varphi_{[0, \frac{1}{2})} + s_1 \cdot \varphi_{[\frac{1}{2}, 1)}
$$

$$
:= s_0 \cdot \frac{1}{2}(\varphi_{[0,1)} + \psi_{[0,1)}) + s_1 \cdot \frac{1}{2}(\varphi_{[0,1)} - \psi_{[0,1)})
$$

$$
:= \frac{s_0 + s_1}{2} \cdot \varphi_{[0,1)} + \frac{s_0 - s_1}{2} \cdot \psi_{[0,1)}.
$$

Now we have developed a simple approximation of a function using the Haar wavelet that reveals new information about the function. The first coefficient, \(\frac{s_0 + s_1}{2}\), measures the average of the function, and the second one, \(\frac{s_0 - s_1}{2}\), measures the change in the function. Their significance becomes more important when the function is approximated recursively.

Now, we can extend our knowledge of the Haar wavelet to more practical operations: the ability to analyze data from a signal or function, not just a small sample of points. In order to do this we will use the Fast Ordered Haar wavelet transform which begins with an array of \(2^n\) entries and has \(n\) iterations of the basic transform described above. These iterations allow for the approximation of the function or signal, and show the average and difference for each section of the function or signal. Then for a positive integer \(n\), and for each index \(l \in 0, \ldots, n\) define the scaling functions as:

$$
\varphi_{k}^{(n-l)}(r) := \varphi_{[0,1)}(2^{(n-l)}[r - (k)2^{(l-n)}])
$$

$$
= \begin{cases} 
1 & \text{if } k2^{(l-n)} \leq r < (k + 1)2^{(l-n)} \\
0 & \text{otherwise}
\end{cases}
$$
and the wavelet function as:

\[
\psi_k^{(n-l)}(r) := \psi_{[0,1)}(2^{(n-l)}[r - (k)2^{(l-n)}])
\]

\[
= \begin{cases} 
1 & \text{if } k2^{(l-n)} \leq r < (k + \frac{1}{2})2^{(l-n)} \\
-1 & \text{if } (k + \frac{1}{2})2^{(l-n)} \leq r < (k + 1)2^{(l-n)} \\
0 & \text{otherwise}
\end{cases}
\]

The last step before constructing the transform is establishing a one-dimensional array of sample values with the length \(2^n\). We can index these as \(\vec{a}^{(n)} = (a_0^{(n)}, a_1^{(n)}, \ldots, a_{2^n-2}^{(n)}, a_{2^n-1}^{(n)})\) which we set equal to \(\vec{a} = (s_0, s_1, \ldots, s_j, s_{2^n-2}, s_{2^n-1})\), and then we approximate the function as the sum of these arrays times the scaling function, \(\tilde{f}^{(n)} = \sum_{j=0}^{2^n-1} a_j^{(n)} \phi_j^{(n)}\).

Now that we have defined the scaling and wavelet function for the fast ordered Haar wavelet transform we can construct the transform; first the \(l\)th sweep of the transformation begins with the previous array, \(\vec{a}_{(n-[l-1])}, \ldots, a_{(2n-(l-1)-1)}\). Then it applies the basic wavelet transform explained above to each of the pairs of points in the array to get the new coefficients, \(a_k^{(n-l)} := a_{2k(n-[l-1])} + a_{2k+j(n-[l-1])}\), and, \(c_k^{(n-l)} := a_{2k+2}^{(n-[l-1])} - a_{2k+1}^{(n-[l-1])}\) for each pair. Now we have the average and the difference for every set of points in the data. Then we reconstruct the data set as the combination of the lists of averages, and the differences. Next we leave the differences, or the wavelet part of the transform, and once again do the basic wavelet to the scaling function segment of the function. We continue this cycle until the average for the whole function is the first point in the array, and hence it is the only coefficient of a scaling function remaining. Now we have constructed a Haar wavelet transform for the whole set of data points. The first coefficient of the transformed function is the average of the original data points of the function, the second one is the difference in the whole function, the third the difference in the first half the function and the fourth one is the difference of the second half of the function and so on.

We can now show an example of this transform. If we begin with a data set of \(s_n = (3, 1, 0, 4)\), then we will first set the approximated function as:

\[
\tilde{f} = 3\varphi_{[0,\frac{1}{2}]} + 1\varphi_{[\frac{1}{2},\frac{3}{4}]} + 0\varphi_{[\frac{3}{4},1]} + 4\varphi_{[\frac{1}{2},1]}.
\]

Then in the first step of the transform we will find the wavelet and scaling functions for each half of the function:

\[
\tilde{f} = \frac{(3+1)}{2}\varphi_{[0,\frac{1}{2}]} + \frac{(3-1)}{2}\psi_{[0,\frac{1}{2}]} + \frac{(0+4)}{2}\varphi_{[\frac{1}{2},1]} + \frac{(1-4)}{2}\psi_{[\frac{1}{2},1]}.
\]
For the last step, we leave the wavelet functions for the halves and find the scaling and wavelet function for the whole function. Then the transformed function becomes:

$$\tilde{f} = \frac{(2 + 1)}{2} \varphi[0, 1] + \frac{(1 - \frac{3}{2})}{2} \psi[0, 1] + \frac{(3 - 1)}{2} \psi[0, \frac{1}{2}] + \frac{(1 - 4)}{2} \psi[\frac{1}{2}, 1].$$

So now the transformed function gives us the average for the whole function, the change for the whole function, and the change for each half of the function. [6]

### 2.2 Daubechies Wavelet

Daubechies wavelets are the next level in wavelet analysis, and have an important difference from the Haar wavelet. They are continuous, and therefore they more accurately approximate continuous signals. However, you lose the easy computation of the Haar wavelet since the scaling and wavelet functions are much more complicated to compute.

To begin our calculation of the Daubechies wavelets we will develop the scaling function. The initial values are $\varphi(0) := 0$, $\varphi(1) := \frac{1 + \sqrt{2}}{2}$, $\varphi(2) := \frac{1 - \sqrt{2}}{2}$, $\varphi(3) := 0$ which add to one. In other definitions of the Daubechies scaling function the initial values differ and add up to other values such as $\sqrt{2}$ and 2. The scaling function is defined on the interval $[0, 3]$. Then the scaling function, $\varphi$, is defined by the recursive function: $\varphi(r) = h_0 \varphi(2r) + h_1 \varphi(2r - 1) + h_2 \varphi(2r - 2) + h_3 \varphi(2r - 3)$. We can then create a graph of the scaling function:

![Graph of the scaling function](image)

Next we can define the wavelet in terms of the scaling function as $\psi(r) := -h_0 \varphi(2r - 1) + h_1 \varphi(2r) - h_2 \varphi(2r + 1) + h_3 \varphi(2r + 2)$. In addition, because of the bounds on $\varphi$, the wavelet function, $\psi$, is nonzero for values of $r \in (-1, 2)$. The graph for the wavelet function is shown below:
It is important to note that the wavelet and the building blocks only give output at dyadic intervals.

Now that we have the Daubechies wavelet defined, we can use the Fast Daubechies Wavelet Transform to approximate functions and signals. The first step in approximating the function using the Daubechies wavelet transform is to create an approximation of a sample of points from the function by multiples of shifted basic scaling functions like so: \( \tilde{f}(r) = a_{-2}\varphi(r + 2) + a_{-1}\varphi(r + 1) + a_0\varphi(r) + a_1\varphi(r - 1) + a_2\varphi(r - 2) + \ldots + a_{2^n-1}\varphi(r - [2^n - 1]). \)

Then the appropriate way to find \( a_n \) is to set \( a_n \) equivalent to \( s_n \) if we create the set of sample points as a set of points of equal spacing such that \( s(k) \) is the output of the function at the input point of \( k \).

The complication with creating this approximation is that the coefficients for \( a_k \) would extend further than the data set. Therefore we need to expand the data points in some way to alleviate edge effects. There are three common ways to extend the data sample, padding the sample with zeros, adding a periodic extension of the data, or increasing the data using a cubic spline function to approximate what the edge of the data should resemble. These three methods all have their advantages in this order they range from fastest to slowest, and least accurate to most. Therefore for each function finding the correct balance between speed and accuracy is necessary.

Once we have padded the data and created the first approximation, the transformation continues by solving the system

\[
\tilde{f}(r) = \sum_{k=0}^{2^n-1} a_k^{(n-1)} \varphi([r/2] - k) + \sum_{k=0}^{2^n-1} c_k^{(n-1)} \cdot \psi([r/2 - 1] - k)
\]

by matrix arithmetic and using the already established functions for \( \varphi(r) \) and \( \psi(r) \). Like the fast Haar wavelet transform, the transform is repeated for \( n \) steps, where the length of the data set is \( 2^n \) (or \( n+1 \) if after sample
has been expanded). After these steps the only $a_k$ coefficient left is the first coefficient in the transformed function and the rest of the function has the wavelet coefficient for the whole function. The next two represent the wavelet transform for the first half and then second half of the function, then the next four separate out the function into fourths and it continues in this manner until only one scaling function remains. Now we have shown how to use Daubechies wavelets to approximate a continuous function with other continuous functions. Later we will discuss the useful applications of the Daubechies wavelet. [6]

2.3 Other Wavelets

Now that we have an understanding of two very useful wavelets it is explanatory to quickly describe a couple other wavelet forms such as the Quadratic and Cubic Spline wavelet, the Gabor wavelet, the Chirp wavelet, and the Mexican Hat wavelet. First spline wavelets are practical for modeling functions, curves and surfaces. They can be either a specific formula or derived by recursion. The quadratic spline example is from a formula while the cubic is a recursive function. The Quadratic Spline is formed by this simple piecewise quadratic formula, and creates the graph shown below: [2]

Then the Cubic Spline is defined first by the scaling functions of: $N_1(t) = \chi[0, 1]$ and $N_m(t) = \frac{t}{m-1}N_{m-1}(t) + \frac{m-t}{m-1}N_{m-2}(t-1)$ where $\chi$ is the characteristic equation. And then the spline wavelets are created by: $S_m(t) = \sum_n q_n N_m(2t-n)$ with $q_n = (-1)^n \sum_{l=0}^{m} (\frac{m}{l}) N_{2m}(n+1-l)$ for $n = 0, 1, \ldots, 3m-2$. [2] This creates a more fluctuating wavelet shown here:
As the iterations increase the spline wavelet approaches the Gabor wavelet: 
\[ g_k(t) = exp(-\alpha^2 t^2) exp(j2\pi f_k t). \]
The Gabor wavelet, pictured below, is the limit of the Cubic Spline wavelet, and are implemented using the formula rather than by iteration. [2]

Furthermore, the Gabor wavelet belongs to the Chirp class of wavelets that are calculated by the function 
\[ c_k(t) = exp(-\alpha^2 t^2) exp(j2\pi f_k t + \frac{1}{2}rt^2) \]
and are pictured here:

Both Chirps and Gabor wavelets are not compactly supported so the very small values are set to zero. [2]
Finally we come to the Mexican hat wavelet which is defined by: \( M(t) = \frac{2}{\sqrt{3}} \pi^{\frac{1}{4}} (1 - x^2) \exp(-\frac{t^2}{2}), \) and pictured above.\(^2\)

It is interesting to notice that the exponential term in the Mexican hat wavelet is the second derivative of the Gaussian function \( \exp(-t^2/2). \) It is easy to see from the graphs and definitions of the wavelet functions that these wavelets all have similar properties. For example, that their areas all sum to one and they can all be shifted and dilated. In addition, these wavelets are all useful in speech and data compression, and experimenting with different wavelets is important to compare and discover which is the most useful.\(^2\)

\section{Wavelet Applications - Adam Steege}

\subsection{Wavelet applications in one dimension}

Both discrete and continuous wavelets have a variety of applications including edge detection, noise reduction, data compression, data transmission, and data comparison. Discrete wavelets tend to be used for data compression, data transmission and basic noise reduction.

Consider the Haar wavelet: looking at the first and center values of a one dimensional Haar transformation yields an image of the data which is the averages of the first and second halves of the set. Going further, one can use the elements which begin each quarter of the transformation data to reconstruct a slightly more detailed image of the original data, ie the averages of each quarter set. Thus, by increasing the dyadic depth of the transformation data which we use for the reconstruction, we gain more and more precision.

Imagine applying this method to streaming media from the internet. For the sake of argument, and because this was the focus of our project, we’ll use sound as an example. A sound file is a one-dimensional list. If a server holds the Haar transform of this list in memory, and sends this data to a user one dyadic level at a time, the user gradually gains a more crisp recording, and they can select based on their available bandwidth the quality of recording which they’d like to receive. While this does not immediately yield a streaming effect, this could be obtained by breaking the sound file into pieces, performing a Haar transform on each piece, then sending the transformation data to the required dyadic depth one section at a time.
Wavelets can also be used for noise reduction and data compression, and these processes are quite similar. Given a set of data, the wavelet transformation of this data will typically have coefficients close to zero, and many coefficients with unnecessary precision. By limiting the number of digits used for these coefficients and deleting coefficients less than an arbitrary threshold, which would be determined based on the file in question, one can preserve the original list with much less information than was originally present. The goal of this process is to eliminate information without having this loss be perceptible by people in a damaging way.

A similar method is used with Fourier analysis in the context of mp3 encoding. The amplitudes of various harmonics are recorded in memory, discarding insignificant harmonics and limiting the precision of all harmonics. The result is a set of amplitudes which can be inversely transformed to produce a sound file almost indistinguishable from the original to anyone but a true audiophile.

### 3.2 Image compression

Wavelets applied in two dimensions have seen recent fame in such contexts as the FBI fingerprint database and the new JPEG2000 image compression software. The FBI has over 2000 terabytes of fingerprint data already, and collects fingerprints at a rate of 30000-50000 per day, with each raw image occupying ten megabytes of space [8]. The WCQ algorithm which they currently use for compression utilizes wavelet transforms, although the precise methods are not available to the public. There is currently a $1 billion contract up for anyone who can develop a compression algorithm even better than the one they have, which preserves the fidelity of the original as well as their current algorithm. Compressing an image file with wavelets is as simple as compressing an audio file with wavelets: transform, cut out small values and limit the precision of the larger ones, and transform back to view the image. When the image is stored as a wavelet transform, the fact that many coefficients have been reduced to zero means that much less information is needed to store the image without losing integrity. To give an example of this, consider the first part of figure 1, an uncompressed fingerprint image. The white marks on the black lines represent sweat glands on the ridges of the fingerprint. These marks are admissible in court as identifying features, and thus a good compression algorithm should preserve these marks [9]. As seen in the second part of the figure, the wavelet compression process with a
data compression ratio of 12.9:1 does not lose the white marks on the ridges of the fingerprint. For comparison, the third part of the figure shows the result of compressing the file at a 12.9:1 ratio with the old JPEG algorithm. Some of the identifying marks which were preserved by the wavelet algorithm have disappeared in the old JPEG algorithm, and blocking artifacts have appeared as well. Unfortunately, since both the new JPEG algorithm and the FBI’s algorithm are closed source, so we do not know precisely what type of two-dimensional wavelet was used in either case.

![FBI fingerprints under various compression techniques](image)

Figure 1: FBI fingerprints under various compression techniques

A further example of data compression can be seen below. We implemented the technique of coefficient thresholding with a two dimensional Haar transformation on the image of Lena. The numbers below each image are the fraction of coefficients which are non-zero after the compression process. This simple implementation produced a decent image at a 4:1 compression ratio, which is significant given the rudimentary nature of the Haar wavelet. Using a different wavelet which was not defined as a strict stepping function would likely eliminate some of the block effects seen in the compressed images.

### 3.3 Captchas

To apply what we knew about wavelets, we chose to investigate captchas. Captchas are security devices put in place to prevent software from creating accounts on various websites, especially email providers. A captcha can come in two forms: audio and visual. Visual captchas are typically blurred or stylized words or sequences of letters and numbers which a user must type in
Figure 2: "Lena" compressed via the Haar transformation at various thresholds

to a text box to verify that they can read the captcha. Audio captchas are
a string of numbers which are read aloud with noise added to the recording.
This is for the purpose of allowing blind computer users to utilize the website
in question. We decided to try to develop software based on wavelets which
would decode audio captchas. We decided to use audio captchas because
they would require noise filtration, edge detection (to parse the sound file
into individual clips of numbers) and data comparison to identify the numbers
being said. We also chose them because they are one dimensional and seemed
like a more tractable problem than visual captchas. Two sources which we
used for captchas were MSN mail and Google mail. Both sources proved to have very difficult captchas to break, though we achieved moderate success with MSN captchas. Upon discovering the complexity of the problem which these captchas posed, we decided to create our own captchas for the purpose of examining the operation of wavelets on a very basic level.

To produce our homemade captchas, we started with a base set of recorded numbers. We had several recordings of each digit being spoken, and created software which took a random sampling of these digits and built a captcha sound file from these pieces. Noise was then added via two methods. The first was stochastic noise (white noise) and the second was cut-and-paste noise using a sound file as a noise source. The latter method took a random clip from a noise source file and added the amplitudes from this clip to the amplitudes in a random spot from the new captcha. Each time this addition was performed, the amplitudes being added were scaled by a factor less than one to control how much noise was created. This process was repeated an arbitrary number of times specified by the user.

4 Solving captchas - Trevor Burnham

The challenge we’re faced with is this: We have one captcha (a sound vector containing some finite number of digits), and a bank of reference digits with corresponding identifiers (e.g. “7” or “1”). Each captcha digit has an unknown identifier identical to that of one of the reference digits. We want to identify the captcha digits by associating each with the most similar reference digit. There are two functions that we need: A windowing function $W$ that splits a captcha into its digits, and a comparison function $C$ that takes two sounds and returns a number expressing how similar the two digits sound.

More formally, suppose we have one “unknown” vector, $c$, and a finite list\(^1\) of “known” digit vectors, $\{r_1, \ldots, r_k\}$, with a corresponding list of identifiers $\{i_1, \ldots, i_k\}$. (The $c$ stands for captcha, the $r$ stands for reference.) We obtain a list of unknown digit vectors by $\{d_1, \ldots, d_j\} = W(c)$. For each $1 \leq a \leq j$, we take $x_a = x$ s.t. $C(d_a, r_x) \geq C(d_a, r_b)$ for every $b$ such that $1 \leq b \leq k$. Define the corresponding identifier of the digit $d_a$ to be $i_{x_a}$.

\(^1\)We use the term “list” here and throughout this section to refer to an ordered set; furthermore, curly braces always indicate lists, not unordered sets. This usage is borrowed from Mathematica.
4.1 What is sound?

Before constructing the windowing and comparison functions, it may be useful to elucidate on precisely how sounds are represented digitally.

Physically, a sound vector of length $N$ corresponds to $N$ measures of air pressure. Humans cannot hear air pressure itself; instead, we hear changes in air pressure. We call constant air pressure “silence.”

The simplest kinds of sounds—those that we hear as “constant” sounds, such as a single note being held—are, in fact, sine waves, with the frequency of the wave determining the pitch. So, a sound vector that samples the function

$$\sin(2\pi h)$$

at some reasonably high sampling rate is perceived as a single note with a frequency of $h$ Hz. (For instance, if $h$ is 440, the note sounds like a middle A.)

An interesting duality exists between pitch and tempo: To a machine, there is no distinction between a series of pulses that repeat 500 times per minute and the same pulses repeating 500 times per second, other than the obvious time difference. To the human ear, however, the former sounds like a series of pulses (frequency produces rhythm) while the latter sounds like a single note (frequency produces pitch). We measure frequency only as an indication of pitch in our project.\(^2\)

For our purposes, sounds are vectors in $\mathbb{R}^n$ ($n$ is referred to as the “length” of the vector throughout this paper) representing local air pressure sampled at some constant rate (in the case of the MSN captchas, 8000 Hz), where 0 represents the “silence” level and deviations are within some well-bounded range (say, -1 to 1).

4.2 The Discrete Fourier Transform

Recall the definition of the Fourier transform of a function (section 2). More widely used in sound processing is the discrete Fourier transform (DFT).\(^3\)

---

\(^2\)For an interesting look at modern rhythm detection techniques, including the wavelet transform, see William A. Sethares’ Rhythm and Transforms (2007).

\(^3\)The term “Fast Fourier Transform” (FFT) is often used synonymously; it refers to a version of the algorithm that has been modified for faster processing, but yields approximately equivalent results.
Given a vector $s$ of length $N$, the DFT vector $X$ is defined by:

$$X[n] = \sum_{k=0}^{N-1} s[k] e^{-2\pi i nk/N} \quad n = 0, 1, \ldots, N - 1$$

The DFT has an intuitive meaning, as each $X[n]$ provides information on a particular frequency component of $s$. A few details should be noted, however: The DFT element $X[f]$ contains information on frequency $fR$, where $R$ is the resolution defined as $\frac{h}{N}$ ($h$ is the sampling rate, in Hz). Note that the terminology is somewhat misleading: Higher resolutions imply less precision, since all frequencies represented in the DFT are integer multiples of the resolution. Furthermore, because the DFT can only contain information on frequencies from $R$ to $NR$, there is always the possibility of frequency data being excluded. For this reason, and others, the DFT is not quite invertible; the Inverse Fourier Transform (IFT) only approximates the original data, sometimes crudely.

### 4.3 The MSN captcha

The focus of our application was attempting to solve the audio captchas used by Microsoft’s MSN service. These captchas consist of a sequence of ten digits, each read by one of two voices, a male and a female (usually the female). There is also background noise, both during and between the digits, which appears to consist of a mixture of artificially generated white noise and “babble” from the male voice. Although the number of digits is consistent, their length and the amount of time separating them varies.

### 4.4 The windowing function

In order to fruitfully compare the captcha digits to a reference library\footnote{The development of a windowing function to maximize comparisons to a reference library poses a chicken or the egg problem: Which comes first? That is, how does one extract useful reference digits without an effective windowing function? In order to step around this, our reference digits were extracted “by hand,” manually setting the intervals. Eventually, our MSN reference library consisted of 130 digits with corresponding identifiers.}, we need a function that forms accurate “windows” for digit extraction. As we will see, extracting too few digits or too many is disastrous, as is extracting overlapping digits (e.g. part of a “nine” with part of a “seven”).
We exploit the difference in frequency characteristics of voice content versus background noise. In particular, the background noise of the MSN captchas tend to have a more even distribution of frequencies, with less sharp peaks. The windowing function goes through several steps:

1. The given captcha vector \( c \) is separated into a list of \( n \) smaller “snippets” \( \{c_1, \ldots, c_n\} \) such that each \( c_i \) is of length corresponding to 1/16th of a second of sound\(^5\). That is, \( \text{length}(c_i) = \frac{h}{16} \), where \( h \) is the sampling rate (in Hz) associated with the sound. (If necessary, \( c \) is truncated so that its length can be evenly divided by \( \frac{h}{16} \).) Then the maximum of the DFT of each \( c_i \) is taken, forming the list \( m = \{m_1, \ldots, m_n\} \).\(^6\)

2. As a first pass, each snippet \( c_i \) is considered to be “significant” (that is, part of a digit as opposed to mere background noise) if and only if

\[
    m_i > \text{mean}(m) - \frac{\text{stddev}(m)}{10}.
\]

3. Adjacent snippets are considered to be part of the same digit if both are significant. Digits are expanded to include additional snippets if they fall below a minimum length (0.3 seconds of sound).

This windowing process comprises the function \( W \), which proved reasonably successful in tests against MSN captchas, typically extracting 9 out of 10 digits without denoising.

### 4.5 The comparison function

We want to construct a function \( C : \mathbb{R}^m \times \mathbb{R}^n \to \mathbb{R} \) \( (m, n \in \mathbb{Z}) \) such that \( C(s_1, s_2) \) is positively associated with human-perceived similarity between the two vectors (played as sounds). Doing so precisely is an open problem, and any complete solution would make captchas obsolete: Any obfuscatory noise that would hamper an algorithmic attempt to solve the captcha would also render the digits inaudible to humans. Some efforts have been made to emulate human hearing through neural networks and sophisticated statistical models; but for our purposes, a simpler approach will suffice.

\(^5\)The value 1/16 is arbitrary, but was selected after testing several competing values for greatest effectiveness in the case of the MSN captchas.

\(^6\)This step is commonly known as a “windowed Fourier transform.” It ensures that the Fourier transform is performed with consistent resolution, and ensures that short bursts of data are not overlooked.
4.6 A nave comparison method: The dot product

The most obvious approach to gauge similarity between two vectors is to take the dot product. There are two problems, however: The dot product can only compare vectors of identical length, and the dot product is shift-variant.

The first problem may be trivially addressed by truncating the larger vector to the same length as the smaller. The second, however, is a fatal flaw. Consider the effect of shifting even a simple sine curve slightly: Whereas previously peaks were multiplied by peaks, yielding large numbers, now the result is far smaller. Testing showed that the result of comparison between identical, slightly shifted sounds was often worse than comparisons between entirely distinct sounds.

One may try to address this by taking repeatedly shifting the vector and taking the dot product, then using the highest result, or (more cleverly) aligning the “peaks” before comparison. However, this would not correct other changes—such as “stretching” the vector using a slowing or speeding algorithm—that would be imperceptible to a human yet devastating to the dot product.

4.7 A frequency-based comparison method: The Fourier transform

A shift-invariant measure of a sound vector’s content can be obtained using the Discrete Fourier Transform (DFT). Intuitively, the absolute values of the DFT vector X correspond to the significance of each of the sound’s frequencies. In order to compare the “commonality” of frequencies in two sounds, we perform the following simple operation:

\[ C(s_1, s_2) = -||X_1 - X_2||^2 \]

where \( X_1 = DFT(s_1) \) and \( X_2 = DFT(s_2) \). Note that the negation of the value is taken because of our application considers larger “closeness” values to be better, ultimately identifying digits based on the greatest match.

An ideal improvement would be to create a continuous function \( F \), based on the DFT vector and a “wiggle” parameter (a real number corresponding to a number of Hz), such that \( F[f] \) would be the sum of the magnitudes of all frequencies within “wiggle” of \( f \). Then for two such functions \( F_1 \) and \( F_2 \) corresponding to the DFTs of two sounds, the expression \( \int |F_1 - F_2| df \) would approximate the shared “frequency area.”
Unfortunately, generating and manipulating such functions in real time is computationally problematic. As a stop-gap against the resolution problem, the comparison function was modified to truncate the longer sound vector to the length of the smaller before taking its DFT.

4.8 Results (no de-noising)

Four MSN captchas were run through the windowing function and the Fourier comparison function. We measured the number of digits placed in their correct position, and divided by the total number of digits (10, for all MSN captchas) to define accuracy.

Because there are 10 digits in each captcha and 10 possible digits (the numerals 0 through 9), if we assume that digits are randomly assorted, then we would expect to achieve 1 digit per captcha (10% accuracy) out of sheer chance.

<table>
<thead>
<tr>
<th>MSN captcha</th>
<th>Fourier compare (%) accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>60</td>
</tr>
<tr>
<td>2</td>
<td>80</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>40</td>
</tr>
<tr>
<td>5</td>
<td>80</td>
</tr>
<tr>
<td>6</td>
<td>70</td>
</tr>
<tr>
<td>7</td>
<td>80</td>
</tr>
<tr>
<td>8</td>
<td>40</td>
</tr>
<tr>
<td>9</td>
<td>40</td>
</tr>
<tr>
<td>10</td>
<td>40</td>
</tr>
<tr>
<td>11</td>
<td>80</td>
</tr>
<tr>
<td>12</td>
<td>40</td>
</tr>
<tr>
<td>Mean</td>
<td>55.0</td>
</tr>
</tbody>
</table>

The low results in captcha 3 are accounted for by an early flaw in the windowing: The second actual digit was mistaken for noise. As a result, all subsequent digits were misidentified. One has to ask: Could the windowing function be improved by making vocal sound more distinct from background noise?
4.9 Improving precision through de-noising

If there were no background noise, then windowing would be trivial: Digits would be determined uniquely by filling gaps of zeros. Furthermore, one expects that comparisons would be less prone to error, as the Fourier vector would reliably describe the spoken digit.

However, the distinction between “noise” and “data” is a purely human one; there is no obvious mechanical criterion for separating the two. Even so, it was hoped that the unwanted noise could be reduced through a “de-noising” function \( D : \mathbb{R}^n \to \mathbb{R}^n \), which operates as follows:

1. The vector \( s \) is padded (symmetrically) with zeros to the nearest power of 2.
2. The vector is converted, via the D4 Daubechies wavelet transform, into its wavelet representation, \( w \).
3. The standard deviation \( \sigma \) of the wavelet vector \( w \) is obtained, and elements of \( w \) that fall below the threshold \( 1.4\sigma \) are truncated to 0.
4. \( s \) is restored via the inverse transform of \( w \) followed by (symmetric) truncation to its original length.

Listening tests found no discernable change in the result, although an examination of DFTs showed subtle effects.

4.10 Results (with wavelet de-noising)

Twelve MSN captchas were run through the same process as above, this time de-noising each known and unknown digit prior to comparison. (That is, rather than comparing \( s_1 \) and \( s_2 \), we now compared \( D(s_1) \) to \( D(s_2) \).)
<table>
<thead>
<tr>
<th>MSN Captcha</th>
<th>No de-noising (% accuracy)</th>
<th>Daubechies de-noising (% accuracy)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>60</td>
<td>60</td>
</tr>
<tr>
<td>2</td>
<td>80</td>
<td>70</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>40</td>
<td>40</td>
</tr>
<tr>
<td>5</td>
<td>80</td>
<td>90</td>
</tr>
<tr>
<td>6</td>
<td>70</td>
<td>80</td>
</tr>
<tr>
<td>7</td>
<td>80</td>
<td>80</td>
</tr>
<tr>
<td>8</td>
<td>40</td>
<td>40</td>
</tr>
<tr>
<td>9</td>
<td>40</td>
<td>40</td>
</tr>
<tr>
<td>10</td>
<td>40</td>
<td>50</td>
</tr>
<tr>
<td>11</td>
<td>80</td>
<td>90</td>
</tr>
<tr>
<td>12</td>
<td>40</td>
<td>50</td>
</tr>
<tr>
<td>Mean</td>
<td>55.0</td>
<td>57.5</td>
</tr>
</tbody>
</table>

While our de-noising process appears to have had a small beneficial effect, the difference is not statistically significant.

### 4.11 Conclusions

On a technical level, our application was not a success. However, it served as a fertile testing ground for a wide range of original ideas, spurring us on as we exhausted the momentum of the new.

### References

1. Daubechies, I. Ten Lectures on Wavelets.
5. Walnut, D. An Introduction to Wavelet Analysis.

