Numerical Performance of Time-Frequency Transforms in Lossy Audio Coding

Michael Olsen[*]

June 7, 2015

Abstract

Time-frequency analysis and the transforms that it gives rise to play an important role in digital signal processing. In lossy audio compression, for instance, it has been found that working in the transform domain leads to methods that achieve a much higher level of signal compression than could be achieved in the temporal domain. This article gives an introduction to time-frequency analysis through the basic theory of Fourier analysis, wavelets and Gabor frames. The results of a numerical investigation into lossy audio file compression comparing the discrete wavelet, Gabor and cosine transforms are then discussed.

Contents

[*]The University of California, San Diego

Acknowledgement

First and foremost I would like to thank my advisors Prof. Michael Holst and John Moody. Without the assistance of the both of you, I would not even have known where to begin. I deeply thank you both for taking time out of both of your busy schedules to guide me through this process over the past nine months.

1 Introduction

In time-frequency analysis, we are primarily concerned with analyzing and manipulating signals f, where f is some function from $\mathbb{R} \to \mathbb{R}$ or from $\mathbb{R} \to \mathbb{C}$. Typically x represents time and $f(x)$ represents a measurement of some phenomenon occurring at time x such as the amplitude of an electrical current, the displacement of a vibrating string or the change in air pressure caused by a sound wave propagating through the air.

Although it is not difficult to discretize an analog (continuous) signal, via sampling the signal at regular intervals, it can be quite storage intensive. As has been shown in the Shannon-Whittaker Sampling Theorem (see [1] Section 2.4), in order to be able to accurately recreate a discretized signal, it needs to be sampled at twice the rate of the highest frequency component. Thus, in the case of digital audio, since the top of the range of human hearing is around 20,000 Hz (cycles/second), an analog audio signal would need to be sampled at least 40,000 times per second to ensure that every audible frequency component can be accurately recreated during any subsequent reconstruction. This is one of the primary reasons that CD quality audio has a sample rate of 44,100 Hz.

While storing 44,100 samples per second on a modern computer is totally feasible, it still adds up to a large amount of disk space when taking into consideration a large digital audio collection. Thus, being able to compress a digital audio file to a minimal size while still retaining a high perceptual quality becomes quite important. Most modern lossy compression algorithms use an orthogonal time-frequency transform such as the discrete wavelet or cosine transforms. This paper, after giving a mathematical introduction to time-frequency analysis and a few of the main time-frequency transforms, investigates whether or not using an over-complete frame transform of an audio file will result in time-frequency domain properties that can be exploited and therefore lead to better file compression results.

2 Background

2.1 Fourier Series

In Fourier analysis, we are concerned with whether we can take a function f and represent it via a collection of simpler functions. Since, from a musical point of view, many signals are periodic and repeat over some period of fixed length $a > 0$, it would be desirable if we could represent a signal f as a linear combination of sinusoidal functions. More specifically, given a signal $f : \mathbb{R} \to \mathbb{R}$, with fixed period $a > 0$, we would like to represent f in the form

$$\sum_{n=1}^{\infty} a_n \sin(2\pi nx/a) + b_n \cos(2\pi nx/a), \tag{2.1}$$

where $a_n, b_n \in \mathbb{R}$ and x represents time. Also, using Euler's formula $e^{2\pi inx/a} = \cos(2\pi nx/a) + i\sin(2\pi nx/a)$, we could also express this sum as

$$\sum_{n=1}^{\infty} \alpha_n e^{2\pi inx/a}, \tag{2.2}$$

where the coefficients $\alpha_n \in \mathbb{C}$.

Can we represent such a function f in the form of equations 2.1 and 2.2? Furthermore, if so, which functions f can we represent in this manner. Finally, given a series representation of such a function f, will it actually convergence to the value of the original function for all values x?

Since the issues of convergence of Fourier series can be quite subtle due to differences between different classes of functions and the different methods in which convergence can be considered to deal with those classes of functions, we will approach the topic in a way that allows for convergence of a large class of functions with respect to an integral norm. This is referred to as mean-square convergence or, more precisely, convergence in the L^2-sense.

From linear algebra, we know that in an N-dimensional vector space V, any vector $v \in V$ can be represented as a linear combination of orthogonal basis vectors $\{e_1, \ldots, e_N\}$ (see [1] Section 0.2, for further information). Furthermore, if we have a vector space equipped with an inner product and we have an orthonormal basis (a basis in which every basis element has unit norm) for that inner product space, then the following property from [1] holds:

Theorem 1. *Suppose V_0 is a subspace of a finite inner product space V. Suppose $\{e_1, ..., e_N\}$, where N is a positive integer, is an orthonormal basis for V_0, then*

$$v = \sum_{j=1}^{N} \langle v, e_j \rangle e_j. \tag{2.3}$$

Proof. Since $\{e_1, ..., e_N\}$ is an orthonormal basis for V_0, for any $v \in V_0$ we can write v as a linear combination of the basis elements of V_0, thus $v = \sum_{j=1}^{N} \alpha_j e_j$ for some scalars α_j. If we take the inner product of both sides with a certain basis vector e_k, we have

$$\begin{aligned}
\langle v,\, e_k \rangle &= \langle \sum_{j=1}^{N} \alpha_j e_j,\, e_k \rangle \\
&= \sum_{j=1}^{N} \alpha_j \langle e_j,\, e_k \rangle && \text{by the linearity of the inner product} \\
&= \alpha_k \langle e_k,\, e_k \rangle && \text{since the } e_i\text{s are orthonormal} \\
&= \alpha_k.
\end{aligned}$$

Now, substituting our inner product expression for α back into the original expression for v gives the desired result. □

It should be noted that the preceding theorem holds in an infinite dimensional, separable Hilbert space but the proof in that setting is more involved because one has to deal with the convergence of an infinite summation (see [8] Section 4.4.3 for further information).

Can we find an orthonormal basis that is made up of periodic functions such as 2.1 and 2.2? As the following theorem from [11] shows, we can get part way there since the sequence of exponentials over the integers satisfy orthogonality relations.

Theorem 2. *Given $a > 0$, the collection of functions of the form $\{e^{2\pi inx/a}\}_{n\in\mathbb{Z}}$ satisfy the following orthogonality relations*

$$\int_0^a e^{2\pi inx/a} e^{-2\pi imx/a} = \begin{cases} 0 & \text{if } m \neq n, \\ a & \text{if } m = n. \end{cases} \tag{2.4}$$

Proof. Let $a > 0$, then if $m = n$, we have

$$\begin{aligned}
\int_0^a e^{2\pi inx/a} e^{-2\pi imx/a}\, dx &= \int_0^a e^{2\pi i(n-m)x/a}\, dx \\
&= \int_0^a e^0\, dx \\
&= \int_0^a dx = a.
\end{aligned}$$

Otherwise, if $m \neq n$, letting $k = n - m$, we have

$$\begin{aligned}
\int_0^a e^{2\pi inx/a} e^{-2\pi imx/a}\, dx &= \int_0^a e^{2\pi i(n-m)x/a}\, dx \\
&= \int_0^a e^{2\pi ikx/a}\, dx \\
&= \frac{-ia}{2\pi k} \left[e^{2k\pi ix/a} \right]_0^a \\
&= \frac{-ia}{2\pi k} (1 - 1) = 0 && \text{since } e^{2k\pi i} = 1 \text{ for all values of } k.
\end{aligned}$$

□

Now that we have an orthogonal sequence (which can easily be made orthonormal), we need to know if we can use this sequence as the basis of an inner product space that contains the functions that we are interested in representing. If so, we can confidently represent any function f in that space in the form of 2.2 (or 2.1).

Luckily, such a space does exist and we define it below.

Definition 1. *For an interval $a \leq t \leq b$, the space $L^2([a,\, b])$ is the set of all square integrable functions defined on $[a,\, b]$; in other words,*

$$L^2([a,\, b]) = \left\{ f : [a,\, b] \to \mathbb{C};\ \int_a^b |f(t)|^2\, dt < \infty \right\}, \tag{2.5}$$

with inner product given by

$$\langle f,\, g\rangle = \int_a^b f(t)\overline{g(t)}\, dt \qquad \text{for } f,\, g \in L^2([a,\, b]). \tag{2.6}$$

(definition from [1] pages 4-5).

Using the fact that $\{e^{2\pi inx/a}\}_{n\in\mathbb{Z}}$ satisfies the following definition from [11] (for any finite interval $a > 0$), we have almost accomplished our goal.

Definition 2. *A collection of functions $\{g_n(x)\}_{n\in\mathbb{N}}$, L^2 on an interval I is a (general) orthogonal system on I provided that*

a. $\displaystyle\int_I g_n(x)\overline{g_m(x)}\, dx = 0$ *if* $n \neq m$, *and*

b. $\displaystyle\int_I g_n(x)\overline{g_n(x)}\, dx = \int_I |g_n(x)|^2\, dx > 0.$

The collection $\{g_n(x)\}_{n\in\mathbb{N}}$ is a (general) orthonormal system on I provided that it is an orthogonal system on I and

a. $\displaystyle\int_I g_n(x)\overline{g_m(x)}\, dx = 0$ *if* $n \neq m$, *and*

b. $\displaystyle\int_I g_n(x)\overline{g_n(x)}\, dx = \int_I |g_n(x)|^2\, dx = 1.$

We have already shown that $\{e^{2\pi inx/a}\}_{n\in\mathbb{Z}}$ satisfies the orthogonal system requirements of Definition 2. It can easily be shown that making the slight modification $\left\{a^{(-1/2)}e^{2\pi inx/a}\right\}_{n\in\mathbb{Z}}$ yields an orthonormal system over any finite interval of length $a > 0$.

Now, we just need to determine whether for any arbitrary function $f \in L^2([a,\, b])$, we can write f in the form given in Theorem 1. The following theorem from [11] gives us the conditions under which we can do so but first we introduce some notation.

Definition 3. *Given $n \in \mathbb{N}$, we say that a function $f(x)$ defined on an interval I is C^n on I if it is n-times continuously differentiable on I. $f(x)$ is C^∞ on I if it is C^n on I for every $n \in \mathbb{N}$.*

We say that $f(x)$ is C_c^n on I if it is C^n on I and compactly supported, C_c^0 on I if it is C^0 on I and compactly supported, and C_c^∞ on I if it is C^∞ on I and compactly supported.

Definition 4. *Given a collection of functions $\{g_n(x)\}$, L^2 on an interval I, the span of $\{g_n(x)\}$, denoted span$\{g_n(x)\}$, is the collection of all finite linear combinations of the elements of $\{g_n(x)\}$. The mean-square closure of span$\{g_n(x)\}$, denoted $\overline{span}\{g_n(x)\}$ is defined as follows: A function $f(x) \in \overline{span}\{g_n(x)\}$ if for every $\varepsilon > 0$, there is a function $g(x) \in span\{g_n(x)\}$ such that $\|f - g\|_2 < \varepsilon$.*

Definition 5. *Let $\{g_n(x)\}$ be an orthonormal system on I. Then $\{g_n(x)\}$ is complete on I provided that every function $f(x)$ defined on I is in $\overline{span}\{g_n(x)\}$.*

And now the theorem:

Theorem 3. *Let $\{g_n(x)\}$ be an orthonormal system on an interval I. Then the following are equivalent.*

a. $\{g_n(x)\}$ is complete on I.

b. For every $f(x)$, L^2 on I,

$$f(x) = \sum_{n \in \mathbb{N}} \langle f, g_n \rangle g_n(x) \tag{2.7}$$

in L^2 on I.

c. Every function $f(x)$, C_c^0 on I, is in $\overline{span}\{g_n(x)\}$.

d. For every function $f(x)$, C_c^0 on I,

$$\|f\|_2^2 = \int_I |f(x)|^2 \, dx = \sum_{n \in \mathbb{N}} |\langle f, \, g_n \rangle|^2. \tag{2.8}$$

Proof. See [11] pages 54 - 55. □

Now, we just need to show that $\{e^{2\pi inx/a}\}_{n \in \mathbb{Z}}$ is complete. This result comes in the form of the following theorem from [11] Section 2.3 whose proof is omitted for brevity's sake.

Theorem 4. *The trigonometric system $\{e^{2\pi inx/a}\}_{n \in \mathbb{Z}}$ is complete on any interval of length $a > 0$.*

Thus, with a complete, orthonormal system, a function f is equal to its projection onto the inner product space whose basis is the orthonormal system (in the L^2-sense). This means that the L^2-norm of f is equal to the L^2-norm of its series expansion and therefore will agree with its series expansion for all but finitely many points x. Now, we can define the Fourier series and Fourier coefficients of a signal f.

Definition 6. *Given a function $f(x)$, L^2 on $[0, \, a]$, the Fourier series of $f(x)$ is given by*

$$f(x) = \sum_{n \in \mathbb{N}} c(n) e^{2\pi inx/a}, \tag{2.9}$$

where we call the set $\{c(n)\}_{n \in \mathbb{N}}$ the Fourier coefficients of $f(x)$.

(Definition from [11] integrating the results from above).

Since we are in an inner product space, we know from the proof of Theorem 1 that the Fourier coefficients are given by $c(n) = \langle f, \, e^{2\pi inx/a} \rangle$ and are thus led to the following:

Definition 7. *Given a function $f(x)$, L^2 on $[0, \, a]$ the Fourier coefficients of $f(x)$ are defined by*

$$c(n) = \frac{1}{a} \int_0^a f(x) e^{-2\pi inx/a} \, dx, \quad \textit{for } n \in \mathbb{Z}. \tag{2.10}$$

Since we have found that we can successfully represent a signal on a finite interval as a sum of sines and cosines, it follows that if the signal is an audio signal over time, then the Fourier series of f breaks f down into a linear combination of its frequency components and the Fourier coefficients give us the strengths (amplitudes) of those frequency components. Thus, if we want a simple method to compress a signal, we can find the Fourier series of that signal and then throw out any coefficients with values below a desired threshold.

While Fourier analysis is very useful for a great many real life applications, it does have some inherent limitations. First, since we are breaking the signal down into trigonometric components that do not have compact support (i.e. the interval on which they are non-zero is infinite), we are only getting a picture of what the frequency components of the signal are overall (i.e. for the entire signal). If we want to know what the frequency composition of the signal is at a certain point in time, the Fourier series of the signal is not going to give us the answers. One way to get around this limitation is to try and find an orthonormal system that is made up of building blocks that do have compact support. This notion leads to the idea of wavelets.

2.2 Wavelets

Although Fourier series are useful for many applications, sometimes we want to know about or need to work with localized sections of a signal. For example, suppose that a signal has a spike of high frequency energy at one or more points over an interval of interest and we would like to remove those irregularities. The Fourier series of the signal will tell us that high frequency content exists in the signal but it will not tell us where it exists. Therefore, if we arbitrarily remove high frequency coefficients from the Fourier series of the signal, they will be removed from the entire signal and we may loss important high frequency content from a different portion of the signal.

As was mentioned at the end of the previous section, the reason that Fourier series are not suited to this task is because the basis functions do not have compact support. Since they are trigonometric functions, they oscillate infinitely. This leads to the question of whether or not we can find basis functions with compact support that do still satisfy the orthogonality relations that made the sequence $\{e^{2\pi inx/a}\}_{n\in\mathbb{Z}}$ so useful to work with. The answer is yes and one such example is the Haar basis which is historically considered to be the first orthonormal wavelet basis.

Basically, we will start with a single building block and will shift it in time and scale it in frequency to create a family of basic building blocks with which we can work. Before defining the Haar basis, we need to define the three fundamental operations in time-frequency analysis: translation, modulation and dilation.

Definition 8. *We define the following operations on functions $f : \mathbb{R} \to \mathbb{C}$.*

$$\begin{aligned} \textit{Translation: } & T_a f(x) = f(x-a), && a \in \mathbb{R}. \\ \textit{Modulation: } & M_b f(x) = e^{2\pi ibx} f(x), && b \in \mathbb{R}. \\ \textit{Dilation: } & D_r f(x) = r^{1/2} f(rx), && r > 0. \end{aligned}$$

(Definition courtesy of [5].)

Regarding an audio signal, translation is an operation that shifts a signal in time, modulation shifts the frequency of the signal and dilation either narrows or widens the signal in time. In wavelet analysis, dilation and translation are the main time-frequency operations.

Now we can define the basic building blocks with which we will be working:

Definition 9. *The Haar scaling function is defined as*

$$\phi(x) = \begin{cases} 1 & \text{if } 0 \le x < 1, \\ 0 & \text{elsewhere.} \end{cases} \tag{2.11}$$

(Definition courtesy of [1].)

This is clearly just the characteristic function on the interval [0, 1] otherwise known as the unit step function. It is not very useful by itself however using the dilation and translation operations defined above we are able to create a useful system of functions. We now give the resulting definition from [11].

Definition 10. *Let $\phi(x)$ be defined as in Definition 9, and for each $j, k \in \mathbb{Z}$, define*

$$\phi_{j,k}(x) = 2^{j/2}\phi(2^j x - k) = D_{2^j} T_k \phi(x). \tag{2.12}$$

The collection $\{\phi_{j,k}(x)\}_{j,k\in\mathbb{Z}}$ is referred to as the Haar system on $\mathbb{R}$. For each $j \in \mathbb{Z}$, the collection $\{\phi_{j,k}(x)\}_{k\in\mathbb{Z}}$ is referred to as the system of scale j Haar scaling functions.

Now that we have a system, lets identify the space in which they operate using a definition from [1]:

Definition 11. *Suppose j is any nonnegative integer. The space of step functions at level j, denoted by V_j, is defined to be the space spanned by the set*

$$\{..., \phi(2^j x + 1), \phi(2^j x), \phi(2^j x - 1), \phi(2^j x - 2), ...\},$$

over the real numbers. V_j is the space of piecewise constant functions of finite support whose discontinuities occur at $k/2^j$ for $k \in \mathbb{Z}$.

Thus, V_0 is the family of piecewise continuous unit step functions with discontinuities at possibly integer values and V_1 is the family of piecewise continuous unit step functions with possible discontinuities at $k/2$ for $k \in \mathbb{Z}$. It also follows that $V_0 \subset V_1$, since $\phi(x) = \phi(2x) + \phi(2x-1)$ and that argument extends inductively to show

$$V_0 \subset V_1 \subset \cdots \subset V_j \subset \cdots$$

Furthermore, as the following theorem from [1] shows us, the scale j Haar scaling functions form an orthonormal basis of V_j.

Theorem 5. *The set of functions $\{2^{j/2}\phi(2^j x - k)\}$ is an orthonormal basis of V_j.*

Proof. In order to proof the theorem, it suffices to show that the functions $\{2^{j/2}\phi(2^j x - k)\}$ are orthonormal since, by Definition 11, they clearly span V_j. Now, fix $j \geq 0$ and suppose that $k = m$, then

$$\begin{aligned}
\langle 2^{j/2}\phi(2^j x - k),\, 2^{j/2}\phi(2^j x - m)\rangle &= \langle 2^{j/2}\phi(2^j x - k),\, 2^{j/2}\phi(2^j x - k)\rangle \\
&= \int_{-\infty}^{\infty} \left(2^{j/2}\phi(2^j x - k)\right)^2 dx \\
&= 2^j \int_{-\infty}^{\infty} \left(\phi(2^j x - k)\right)^2 dx \\
&= 2^j \int_{\frac{k}{2^j}}^{\frac{k+1}{2^j}} dx \\
&= 2^j \left[x\right]_{\frac{k}{2^j}}^{\frac{k+1}{2^j}} = 1.
\end{aligned}$$

Otherwise, if $k \neq m$, then $2^{j/2}\phi(2^j x - k)$ and $2^{j/2}\phi(2^j x - m)$ have disjoint support, so $\langle 2^{j/2}\phi(2^j x - k),\, 2^{j/2}\phi(2^j x - m)\rangle = 0$ and the result is proved. □

Now, if we have a function that can be closely approximated by the system V_j for sufficiently large j, if a spike occurs in our function and we would like to eliminate it, then we need a method with which we can isolate it from the rest of the function. Thus, we would like to remove all components of width $1/2^j$ which means that we need to see which components can be represented as linear combinations of components of V_{j+1}, V_{j+2},...

We can make use of the following linear algebra concepts from [1] to move towards this goal.

Definition 12. *Suppose V_0 is a subspace of an inner product space V. The orthogonal complement of V_0, denoted $V_0^{\perp}$, is the set of all vectors in V that are orthogonal to V_0; that is*

$$V_0^{\perp} = \{v \in V; \langle v,\, w\rangle = 0 \textit{ for every } w \in V_0\}.$$

Theorem 6. *Suppose V_0 is a finite dimensional subspace of an inner product space V. Each vector $v \in V$ can be written uniquely as $v = v_0 + v_1$, where v_0 belongs to V_0 and v_1 belongs to $V_0^{\perp}$; that is,*

$$V = V_0 \oplus V_0^{\perp}.$$

Proof. See [1] page 18. □

Therefore, for our space of piecewise continuous unit step functions V_j, we need to find the orthogonal complement of V_{j-1} in V_j to identify the parts of the function that can only be made from the components of V_j. The resulting family of functions are called the Haar wavelets as the following definition and theorem from [1] show.

Definition 13. *The Haar wavelet is the function*

$$\psi(x) = \phi(2x) - \phi(2x-1). \tag{2.13}$$

Theorem 7. *Let W_j be the space of functions of the form*

$$\sum_{k\in\mathbb{Z}} a_k \psi(2^j x - k), \quad a_k \in \mathbb{R}, \tag{2.14}$$

where we assume that only a finite number of a_k are nonzero. W_j is the orthogonal complement of V_j in V_{j+1} and

$$V_{j+1} = V_j \oplus W_j.$$

Proof. See [1] pages 164 - 165. □

Thus, we have a method for isolating spikes in a function that can be represented using the family of unit step functions V_j. In real life though, we would like to be able to apply these techniques to real world functions that could be smooth and continuous. As the following theorem from [1] will show, this is possible in theory.

Theorem 8. *The space $L^2(\mathbb{R})$ can be decomposed as an infinite orthogonal direct sum*

$$L^2(\mathbb{R}) = V_0 \oplus W_0 \oplus W_1 \oplus \cdots . \tag{2.15}$$

In particular, each $f \in L^2(\mathbb{R})$ can be written uniquely as

$$f = f_0 + \sum_{j=0}^{\infty} w_j,$$

where f_0 belongs to V_0 and w_j belongs to W_j.

Thus, for any L^2 function that we would like to work with, we can approximate it arbitrarily closely using a suitably large j. Then we can decompose it into the orthogonal direct sum given above and begin eliminating unwanted components that meet certain criteria. With the dilation operation, this type of analysis is known as a multi-resolution analysis because we can "zoom" in on certain features in time. Therefore, wavelets give us a method of overcoming the lack of localization inherent in Fourier series.

Sometimes, however, a basis can be restrictive and does not give us the flexibility that we may desire. Suppose, for example, that we are compressing a signal and then transmitting it to a destination over a communications line. If we use either Fourier series or wavelets, we are transmitting the exact amount of information necessary for the receiver to reconstruct the signal. What happens if one or more pieces of data are either lost or modified during transmission? In that case, it might not even be possible for the receiver to reconstruct the signal effectively or the signal may be incorrectly reconstructed. Thus, it might be desirable to transmit more data than is theoretically necessary to give our transmission some redundancy and a greater chance of being reconstructible. It is thinking along these lines that led to the idea of frames and, in particular, Gabor frames.

2.3 Gabor Frames

Gabor systems, and more specifically Gabor frames, are an interesting class of functions generated by finding an appropriate window function $g \in L^2(\mathbb{R})$ and then constructing a family of functions from translations and modulations of that window function g. When an appropriate system has been created, one is then able to analyze a signal f in terms of the time-frequency plane $\mathbb{R}^2$ and determine where the energy of f is concentrated to a certain level of accuracy. Such a representation has uses in signal transmission, analysis, de-noising and compression. Before we can discuss Gabor systems and Gabor frames however, we first need to introduce some general frame theory.

We begin with the definition of a Hilbert space taken from [5] and then provide the general definition of a frame given by [4]:

Definition 14. *A Hilbert space $\mathcal{H}$ is a normed inner product space whose norm is induced by the dot product of the space. A Hilbert space $\mathcal{H}$ must also be complete with respect to that induced norm.*

Definition 15. *A frame for a Hilbert space $\mathcal{H}$ is a sequence of vectors $\{x_i\} \subset \mathcal{H}$ for which there exist constants $0 < A \leq B < \infty$ such that, for every $x \in \mathcal{H}$,*

$$A\|x\|^2 \leq \sum_i |\langle x,\, x_i\rangle|^2 \leq B\|x\|^2. \tag{2.16}$$

The constants A and B are known respectively as the lower and upper frame bounds. We will often presume without stating it that A and B are the optimal frame bounds for a frame. A frame is called a tight frame if the optimal lower and upper frame bounds are equal and is called a Parseval frame if $A = B = 1$.

The following definition from [4] makes clear the relationship between frames and bases.

Definition 16. *Let $\mathcal{H}$ have dimension n and let $\{x_i\}_{i=1}^k$ be a frame for $\mathcal{H}$ having k elements. The redundancy of the frame is the quantity $\frac{k}{n}$, which must be greater than or equal to 1.*

Thus, if $k/n = 1$, we have $k = n$ and $\{x_i\}_{i=1}^k$ is a basis for $\mathcal{H}$, otherwise, if $k/n > 1$, we have $k > n$ and it is clear that $\{x_i\}_{i=1}^k$ is not a basis for $\mathcal{H}$ since it contains more vectors than the dimension of $\mathcal{H}$.

Given a frame for a Hilbert space $\mathcal{H}$ it is natural to wonder how one can express the vectors in $\mathcal{H}$ in terms of the frame elements. The answer comes from the following proposition from [4]:

Proposition 1. *Let $\{x_i\}_{i=1}^k$ be a frame for $\mathcal{H}$. Then there exists a frame $\{y_i\}_{i=1}^k$ such that every $x \in \mathcal{H}$ can be reconstructed with the formula:*

$$x = \sum_{i=1}^k \langle x,\, y_i\rangle x_i = \sum_{i=1}^k \langle x,\, x_i\rangle y_i. \tag{2.17}$$

Such a frame $\{y_i\}_{i=1}^k$ is called a dual frame to $\{x_i\}_{i=1}^k$.

Proof. See [4] pages 100 - 102 □

As the preceding proposition tells us, each frame has one or more dual frames associated with it. Yet, the problem of how to find this dual frame still remains to be answered. If we do not have a method of identifying the dual frame corresponding to a certain frame then the reconstruction formula provided in Proposition 1 is of little practical value. In order to overcome this obstacle we need to introduce the functional operators relating to our frame $\{x_i\}_{i=1}^k$. First, we define the analysis operator (definition from [4]).

Definition 17. *Let $\{x_i\}_{i=1}^k \subset \mathcal{H}$. The matrix (operator) $\Theta : \mathcal{H} \to \mathbb{C}^k$ defined by*

$$\Theta x = \begin{bmatrix} \langle x,\, x_1\rangle \\ \langle x,\, x_2\rangle \\ \vdots \\ \langle x,\, x_k\rangle \end{bmatrix} = \sum_{i=1}^k \langle x,\, x_i\rangle e_i, \tag{2.18}$$

is called the analysis matrix (or operator) of $\{x_i\}_{i=1}^k$, where $\{e_i\}_{i=1}^k$ is the standard orthonormal basis for $\mathbb{C}^k$.

This operator gives us a method for mapping vectors in $\mathcal{H}$ to their corresponding coefficient vectors in $\mathbb{C}^k$. Can we go in the reverse direction? The answer is yes as the following definition from [4] shows.

Definition 18. *The adjoint Θ^* of the analysis operator Θ, which will map $\mathbb{C}^k \to \mathcal{H}$, is called the synthesis operator. The operator $S = \Theta\Theta^* : \mathcal{H} \to \mathcal{H}$ is called the frame operator.*

Using properties of the frame operator S will allow us to reach our goal of definitively finding a dual frame to any frame $\{x_i\}_{i=1}^k$ and thus allow us to utilize the reconstruction formula 2.17. Lets now define a dual frame of a frame $\{x_i\}_{i=1}^k$ using the definition from [4].

Definition 19. *Let $\{x_i\}_{i=1}^k$ be a frame for a Hilbert space $\mathcal{H}$. A sequence $\{y_i\}_{i=1}^k$ in $\mathcal{H}$ is called a dual frame for $\{x_i\}_{i=1}^k$ if $\{y_i\}_{i=1}^k$ satisfies the reconstruction formula 2.17.*

If $y_i = S^{-1}x_i$, $i = 1, 2, ..., k$. then it is called the canonical dual frame. If $\{y_i\}_{i=1}^k$ is not the canonical dual frame, it is called an alternate dual frame.

It can be shown that if S is positive and invertible on $\mathcal{H}$ that $Sx = \sum_{i=1}^{k} \langle x,\, x_i \rangle x_i$ for all $x \in \mathcal{H}$ (See [4] Chapter 3.3 for further information). Then, by noting that $x = S^{-1}Sx$, we get the reconstruction formula:

$$x = S^{-1}Sx = S^{-1}\sum_{i=1}^{k}\langle x,\, x_i\rangle x_i = \sum_{i=1}^{k}\langle x,\, S^{-1}x_i\rangle x_i = \sum_{i=1}^{k}\langle x,\, x_i\rangle S^{-1}x_i \tag{2.19}$$

for all $x \in \mathcal{H}$. Therefore, for any frame $\{x_i\}_{i=1}^{k}$ for $\mathcal{H}$, we can always compute the canonical dual frame $\{S^{-1}x_i\}_{i=1}^{k}$. This leads to the importance of the frame bounds A and B. As detailed in [4] (pages 107-108), if A and B represent the optimal lower and upper frame bounds and the frame operator S, then the eigenvalues of S, λ_{min} and λ_{max} are those optimal frame bounds and the quantity $B/A = \lambda_{max}/\lambda_{min}$ is called the condition number of S and gives an indication of how accurate a numerical computation of S^{-1} will be.

It should be noted that a majority of the general frame theory just discussed can be transferred to the infinite dimensional setting. For a detailed exposition of these results the reader is referred to [2], Chapter 5. Now that we are equipped with some general frame theory, we are ready to move into the main topic of this section: Gabor systems and Gabor frames. First, we will define a Gabor system using the definition from [5].

Definition 20. *A lattice Gabor system, or simply a Gabor system for short, is a sequence in $L^2(\mathbb{R})$ of the form*

$$\mathcal{G}(g,\, a,\, b) = \{e^{2\pi i b n x} g(x - ak)\}_{k,n\in\mathbb{Z}}, \tag{2.20}$$

where $g \in L^2(\mathbb{R})$ and $a,\, b > 0$ are fixed. We call g the generator or the atom of the system and refer to $a,\, b$ as the lattice parameters.

Now, it is clear that a Gabor system provides us with translations and modulations of some window function g in $L^2(\mathbb{R})$, but that alone is not of much use. We would prefer that the sequence have some additional properties that we could exploit, such as being a frame. As it turns out, some Gabor systems can be frames as the following definition from [2] illustrates:

Definition 21. *A Gabor frame is a frame for $L^2(\mathbb{R})$ of the form $\{M_{bn}T_{ak}g\}_{k,n\in\mathbb{Z}}$ where $a,\, b > 0$ and $g \in L^2(\mathbb{R})$ is a fixed function.*

Thus, a Gabor frame is just a Gabor system that also happens to be a frame. Now, in order to use Gabor frames to analyze a signal, we need the explicit definition of the frame operator. We give it using the definition from [3]

Definition 22. *If $\mathcal{G}(g,\, a,\, b)$ is a Gabor frame, then its frame operator is*

$$Sf = \sum_{k,n\in\mathbb{Z}}\sum \langle f,\, M_{bn}T_{ak}g\rangle M_{bn}T_{ak}g, \tag{2.21}$$

for any function $f \in L^2(\mathbb{R})$.

It can be shown that the frame operator commutes with both M_{bn} and with T_{ak} for $k,\, n \in \mathbb{Z}$ and so we also get that S^{-1} commutes with M_{bn} and with T_{ak} which leads to the following lemma from [5]:

Lemma 1. *If $\mathcal{G}(g,\, a,\, b)$ is a Gabor frame for $L^2(\mathbb{R})$, then its canonical dual frame is $\mathcal{G}(\gamma,\, a,\, b)$ where $\gamma = S^{-1}g$.*

Thus, we now have a formula for the inverse frame operator (from [3]):

Corollary 1. *With the assumptions and results from Lemma 1, the inverse frame operator is given by*

$$S^{-1}f = \sum_{k,n\in\mathbb{Z}}\sum \langle f,\, M_{bn}T_{ak}\gamma\rangle M_{bn}T_{ak}\gamma. \tag{2.22}$$

Thus, it is clear that for every Gabor system $\mathcal{G}(G,\, a,\, b)$, we can always find that canonical dual frame using the atom g and the inverse of the frame operator S. With wavelet frames, in contrast, it is typically very difficult to invert the wavelet frame operator S and even when possible the resulting dual frame does not always turn out to be a wavelet frame (The interested reader is directed to [2] Chapter 12 for further information).

Similar to Fourier series, Gabor frames also have a series expansion and coefficient formula. They are presented below (as taken from [3])

Proposition 2. *If $\mathcal{G}(g, a, b)$ is a frame for $L^2(\mathbb{R})$, then there exists a dual window $\gamma \in L^2(\mathbb{R})$, such that the dual frame $\mathcal{G}(g, a, b)$ is $\mathcal{G}(\gamma, a, b)$. Consequently, every $f \in L^2(\mathbb{R})$ possesses the expansions*

$$f = \sum_{k,n \in \mathbb{Z}} \sum \langle f, T_{ak} M_{bn} g \rangle T_{ak} M_{bn} \gamma = \sum_{k,n \in \mathbb{Z}} \sum \langle f, T_{ak} M_{bn} \gamma \rangle T_{ak} M_{bn} g. \tag{2.23}$$

It follows from the properties of inner product spaces that the Gabor coefficients of f with respect to $\mathcal{G}(g, a, b)$ are given by

$$c_{kn} = \langle f, T_{ak} M_{bn} g \rangle. \tag{2.24}$$

Gabor frames give us two benefits over Fourier series and wavelets. The first being the frame property which gives us redundancy in the signal representation yet still allows us to represent the signal as a linear combination of frame elements. Secondly, when we represent a signal using Gabor frames, the choice lattice parameters a and b control the size of the time-frequency lattice with the Gabor coefficients giving the energy in each grid component. Therefore, this system is as close as we can get to localizing a signal in both time and frequency (due to limitations imposed by the Uncertainty Principle, see [3] Chapter 2 for further information).

3 Numerical Investigation

3.1 Audio Coding

Compression is a technique in which one tries to store a digital file with a smaller amount of data than is already being used. There are two main compression schemes: lossless compression and lossy compression. Lossless compression is a form of compression where the original file can still be completely recovered from its compressed version which means that there is no loss of data. Lossless compression typically takes advantage of statistical redundancy in a data file using an encoding scheme like Huffman coding. Contrarily, lossy compression entails throwing out data from the original file, that is deemed to be removable, in order to maximize space savings by getting the compressed version of the file to be as small as possible. This is, however, a non-invertible operation and it is not possible to recover the original file from the compressed version.

In audio file compression (more commonly called audio coding) the most common lossy file coding scheme is the MPEG 1 Layer III (MP3) digital coding format. This audio coding scheme exploits a variety of techniques including the use of psychoacoustic modeling to exploit masking properties of the human ear in identifying certain transform coefficients that can be eliminated without audible loss of sound quality. Additionally, bit allocation and entropy coding are also employed to maximize the space savings after the inaudible coefficients have been removed.

3.2 Motivation

The purpose of this numerical investigation is to investigate three different discrete time-frequency transform techniques and determine which technique yields the most suitable transform (time-frequency) domain representation of digital music files. Suitability is determined by applying a simple lossy coding algorithm using each transform technique and compared observed quality measurements. The investigation consisted of transforming digital audio files from the temporal domain into the transform domain using the different transform techniques. The controllable parameters of each transform technique were varied over a suitably large range of values to determine the best parameter choice for each audio file and transform technique. Next, the transform domain signals were thresholded by nullifying a certain percentage of low energy transform coefficients. Then, the signal-to-noise ratio (SNR) and compression ratio of the coded audio files were recorded. Finally, the results of the numerical process were analyzed to determine the best transform technique per file and overall.

3.3 Transform Types

The three transforms that were chosen for testing were the fast wavelet transform (FWT), the modified windowed discrete cosine transform (MWDCT), and the discrete Gabor transform (DGT). These three transforms were chosen to be able to determine if a frame-based transform (the DGT) would perform better than basis-based transforms (the FWT and the DCT). In other words, does overcompleteness give additional flexibility in the transform domain representation that can be exploited to achieve better compression ratios with a higher sound quality level?

Fast Wavelet Transform: The FWT is a discrete wavelet transform that uses a filter bank approach to reduce the computational complexity of the discrete wavelet transform. A short overview of the FWT will be provided (the interested reader is directed to [6], chapter 7 for further detail). The FWT uses a multi-resolution approximation procedure similar in spirit to the multi-resolution analysis presented with the Haar wavelet in section 2.2. The algorithm uses a filter called a conjugate mirror filter which acts as the wavelet scaling function. This scaling function is then used to construct an orthonormal wavelet basis of $L^2(\mathbb{R})$. With that basis, any finite signal $f \in L^2(\mathbb{R})$ can be expanded using the basis functions. The algorithm, commonly known as Mallat's algorithm, is presented below (from [6]):

Theorem 9 (Mallat's Algorithm)**.** *At the decomposition*

$$a_{j+1}[p] = \sum_{n=-\infty}^{\infty} h[n-2p]a_j[n], \tag{3.1}$$

$$d_{j+1}[p] = \sum_{n=-\infty}^{\infty} g[n-2p]a_j[n]. \tag{3.2}$$

At the reconstruction,

$$a_j[p] = \sum_{n=-\infty}^{\infty} h[p-2n]a_{j+1}[n] + \sum_{n=-\infty}^{\infty} g[p-2n]d_{j+1}[n]. \tag{3.3}$$

The details of the math underlying this algorithm is beyond the scope of this article. For the interested reader, $a[n]$ is defined in equation 7.21 on page 306, $h[n]$ is defined in equation 7.29 on page 309 and $g[n]$ is defined in equation 7.71 on page 323 of [6].

Finally, combining Mallat's algorithm with filters h and g such that h is a bi-orthogonal low-pass filter and g is bi-orthogonal high-pass filter, one gets the FWT which has a computational complexity of $\mathcal{O}(N)$ for a signal of length N.

Modified Windowed Discrete Cosine Transform: The discrete cosine transform (DCT) is related to the discrete Fourier transform (DFT) due to the fact that it also uses trigonometric functions as a basis. As the name suggests, the first noticeable difference is that it only uses cosine functions. We make that specific using the definition from [7].

Definition 23. *The basis functions of the discrete cosine transform are given by*

$$a_{nk} = c(k)\sqrt{\frac{2}{M}}\cos\left[\left(n+\frac{1}{2}\right)\frac{k\pi}{M}\right], \tag{3.4}$$

where

$$c(k) \equiv \begin{cases} 1/\sqrt{2} & \text{if } k = 0, \\ 1 & \text{otherwise}, \end{cases} \tag{3.5}$$

where M is the length of one block of a signal $x(n)$ and a_{nk} is the nth sample of the kth basis function with n, k both ranging between 0 and $M-1$.

The main difference between the DCT and the DFT is that the DCT has twice as many frequency components as the DFT over the same period (i.e. over the interval $[0, \pi]$ for a signal of periodicity π).

The modified discrete cosine transform is a discrete cosine transform that is based on a filter bank implementation that uses an oddly-stacked filter bank (this means shifting frequency of the basis functions defined above by $\pi/2M$ where M is the number of frequency channels. This frequency shifted version of the DCT is known as the DCT-IV). The definition of the WMDCT used in this paper's implementation and taken from [9] is given below:

Definition 24. *With discrete signal f of length L, number of frequency channels M and number of translation coefficients N, the following holds for $m = 0, ..., M-1$ and $n = 0, ..., N-1$*

$$c(m+1, n+1) = \begin{cases} \sqrt{2}\sum_{l=0}^{L-1} f(l+1)\cos\left(\frac{\pi}{M}\left(m+\frac{1}{2}\right)l+\frac{\pi}{4}\right)g(l-nM+1) & \text{if } m+n \text{ is even,} \\ \sqrt{2}\sum_{l=0}^{L-1} f(l+1)\sin\left(\frac{\pi}{M}\left(m+\frac{1}{2}\right)l+\frac{\pi}{4}\right)g(l-nM+1) & \text{if } m+n \text{ is odd,} \end{cases} \tag{3.6}$$

where g is a suitable window function.

As noted in [6], the DCT-IV has an algorithmic complexity of $\mathcal{O}(N\log_2(N))$. It is also important to note that the MDCT is used in the MPEG-1 Layer III (MP3) audio coding algorithm.

Discrete Gabor Transform: Given a finite discrete signal f, let M be the number of frequency channels, a be the translation increment, and L be the length of transform performed which is the smallest integer multiple of a and M that is larger than the length of f. With the restriction that $L = Mb = Na$ where a, b, N, $M \in \mathbb{Z}$, the discrete Gabor transform is given in [9] by

$$c(m+1, n+1) = \sum_{l=0}^{L-1} f(l+1)\overline{g(l-an+1)}e^{-2\pi ilm/M} \tag{3.7}$$

where $m = 0, ..., M-1$ and $n = 0, ..., N-1$ and $l - an$ is computed modulo L. For a complex signal f, the DGT returns an $M \times N$ lattice of complex time-frequency coefficients. Since the wave files used in this investigation are real-valued, the real DGT was used which only returns non-negative frequency channels. This results in $M' = \lfloor M/2 \rfloor + 1$ frequency channels being returned. Since the DGT returns conjugate pairs of complex-valued coefficients, the real DGT only returns the positive $(a + bi)$ complex coefficients of each conjugate pair. The algorithmic complexity of the DGT is given as $\mathcal{O}(NM\log_2(N))$ in [10] where M and N are as defined above. Thus, if M is close to N, the DGT approaches an algorithmic complexity of $\mathcal{O}(N^2\log_2(N))$ which is by far the largest of the three transforms being tested.

3.4 Thresholding

Lossy compression was accomplished by thresholding small valued coefficients in the transform domain. The thresholding technique consisted of nullifying the smallest $x\%$ of coefficients from the transform domain representation of the signal by means of a hard thresholding algorithm provided by [9] resulting in $(1-x)\%$ of the total coefficients being retained for x values between 0.05 and 0.95.

3.5 Implementation

Four different audio files, all around one minute in length, were chosen for testing the lossy audio coding scheme. One was a recording of a solo violin playing an arpeggio, the next was an excerpt of a classical symphony piece, the third an excerpt of a pop song and the final was an excerpt of a Latin percussion ensemble.

Implementation of the numerical analysis of the lossy audio coding scheme was approached in multiple steps:

3.5.1 Lossy Audio Coding Algorithm

Firstly, a MatLab function called Compression.m was created. The main purpose of this script was to run the lossy audio coding scheme, using one particular time-frequency transform, against a single audio file and returning numerical results of the process.

As input, the script takes the audio file location, time step interval a, number of frequency channels M, number of filter bank iterations J, time-frequency coefficient thresholding percentage and transform type. The transform type parameter is an integer between 1 and 3 with 1 = FWT, 2 = WMDCT and 3 = DGT. Parameter a is only required for the DGT, parameter J is only required for the FWT and parameter M is required for both the WMDCT and the DGT. For simplicity, the function was programmed to only accept a mono (one channel) audio file.

For the FWT, the Daubechies db8 wavelet was used as the generating wavelet in the function as it generates a compactly supported orthogonal family of wavelets that can be used in the Mallat algorithm (Theorem 9). For the WMDCT, an orthonormal Wilson type window was chosen as [9] indicates that it is an appropriate basis choice for that transform. Finally, for the DGT, a Gaussian window ($\phi(x) = 2^{1/4}e^{-\pi x^2}$) was chosen for the window function as it is known to minimize the Uncertainty Principle (see [3] page 53).

Regarding output, the function returns the compression ratio, the signal-to-noise ratio (see 3.5.3 for definitions of both) and an array representing the lossy coded version of the input audio file. That array can then be separately converted back into an audio file using MatLab's *wavwrite* function.

When the function is executed, the first step is the import of the audio file into a matrix array in memory using MatLab's built-in *wavread* function. Next, if the array is 1-dimensional (a vector) and the file was therefore monophonic, the L^2-norm of the original audio file is calculated and stored. Then, the specified time-frequency transform is performed using the relevant input parameters resulting in the creation of a time-frequency lattice matrix (in the case of the WMDCT and DGT) or a time-frequency filter bank tree cell array (in the case of the FWT).

The amount of time-frequency coefficients to threshold is then calculated based on the size of the time-frequency lattice or filter bank tree and the user provided threshold percentage. The thresholding is then performed using the *largestn* function from [9] which performs a hard-thresholding algorithm (see [6] for further information about hard-thresholding). After thresholding has been performed, the number of non-zero time-frequency coefficients is determined using the *nonzeros* and *numel* built-in MatLab functions and then the compression ratio is calculated.

Since the real DGT returns a matrix of complex-valued coefficients whereas the FWT and WMDCT both return real-valued coefficients, additional steps were necessary when performing the thresholding and CR calculations for the real DGT so that the real DGT could be accurately compared to the other two real-valued transforms. This process involved transforming the complex-valued coefficient matrix into a real-valued coefficient matrix of twice the size which stored the real and imaginary parts of the complex-valued coefficients separately as real numbers. The thresholding and subsequent compression ratio calculations were then performed against this new matrix.

For the FWT, since it returns a cell array, the author found it necessary to perform additional steps to convert the cell array into a matrix array, perform the thresholding on the matrix array and then transform that array back into a cell array prior to performing the inverse transform. These steps were accomplished using the built-in MatLab functions *cell2mat* and *mat2cell*.

After the compression ratio has been calculated, the function then performs the appropriate inverse transform on the thresholded time-frequency lattice or time-frequency filterbank tree. Next, the L^2-norm of the error (the difference in sample values between the original and compressed matrix array representations of the audio file) is calculated and from that value and the value of the L^2-norm of the original signal, the signal-to-noise ratio is computed.

Lastly, the compression ratio, signal-to-noise ratio and matrix array representation of the lossy coded audio file is returned by the function.

*The full MatLab function listing of Compression.m is provided in A.1.

3.5.2 Numerical Analysis Batch Processing Algorithm

In order to numerically analyze the three different transform types and determine which transform type performed best overall against the four chosen audio files it was necessary to run each transform type against each audio file using a wide range of parameter values. For the FWT, the number of filter bank iterations tested are given by $J = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$. For the WMDCT and DGT, the number of frequency channels tested is given by $M = \{32, 64, 128, 256, 512, 1024\}$. For the DGT, the time-step increment a also needed to be chosen. This choice, however, needed to be more judicious than that of the other parameters as a poor choice would lead to a time-frequency lattice containing a number of coefficients far in excess of the original number of sample values in the analyzed audio file. Such a situation would not be conducive to audio coding where one is trying to minimize the number of retained coefficients.

Since both the FWT and WMDCT use orthogonal basis elements, the number of coefficients in the transform domain is equal to the number of sample values in the original audio file. Due to the over-completeness of the DGT, one can end up with more coefficients in the transform domain than original sample values. Therefore, in order to keep the number of coefficients in the transform domain to a minimum, an upper limit of 125% the number of samples in the audio file was set to bound the size of the time-frequency lattice obtained from the transform. From the definition of the DGT, we have $Mb = Na = L$ where L is the length of the transform performed, M is the number of frequency channels, N the number of time-intervals, b the frequency modulation amount and a the time increment. Since it is additionally required that $a < M$, it was found that $MN < 1.25L \implies M/1.25 < a$. Combining this with the requirement that $a < M$ gives the restriction $M/1.25 < a < M$ on the choice of time increment a. After testing the DGT using the smallest, largest and average acceptable a values satisfying this inequality, it was found that the choice $a = M - 1$ yielded the best results. This makes sense when one notices that this choice of a minimizes N and thus results in the smallest possible $M \times N$ lattice for a particular choice of M meeting the above restrictions.

Since, for each file, the FWT would result in a $2 \times 19 \times J$ matrix and the WMDCT and DGT would result in a $2 \times 19 \times M$ matrix (with 19 representing the number of threshold percentages tested and 2 representing the signal-to-noise and compression ratios returned per threshold value and J or M value), it became necessary to automate the process of running these numerical tests and so a MatLab script (RunCompression.m) was written to accomplish this task.

The function of the script is relatively straightforward. First, the file paths and names of the four audio files to be tested are stored in a cell array. A second cell array is created to store the file name without the file type suffix from each of these file paths. The a, J, M and threshold percentage parameter value ranges are stored in four vectors.

Next, a for loop cycles through each transform type and performs the transform against each possible combination of relevant parameters. The results of these tests are stored in matrix arrays programmatically named with the acronym of the transform type used and with the file name.

*The full MatLab script listing of RunCompression.m is provided in A.2.

3.5.3 Performance Measurements

The final process in the numerical analysis was to analyze the results obtained from running the MatLab scripts and determine which transform type best implemented the lossy audio coding scheme.

Two performance measures were used to determine the best output of each transform technique and choice of coefficients. The first was signal-to-noise ratio (SNR) which can be thought of as a measurement of how audible the error resulting from the coefficient thresholding is in the compressed version of the file. The SNR is given by:

$$SNR = 20 \log_{10} \frac{\|s\|_2}{\|s - \hat{s}\|_2} \tag{3.8}$$

where $\|\cdot\|_2$ is the L^2-norm, s is the original audio file, $\hat{s}$ is the compressed audio file, and $s - \hat{s}$ is the error resulting from compression. The second measurement is the compression ratio (CR) which is a measure of how great the space savings of the coding algorithm are in comparison to the size of the original file. The CR is defined by

$$CR = \frac{\text{total number of audio file samples}}{\text{total number of non-zero transform coefficients}}. \tag{3.9}$$

Now, for each audio file, transform type and threshold percentage, the best pair of SNR and CR values per all J or M needed to be determined so that a SNR vs CR plot could be generated which would graphically illustrate the best transform technique overall and/or which transform technique performed best for each threshold value and each audio file.

As the SNR ratios typically varied between 30 and 300 dB and the CR varied from 1 to 20, in order to equally weight both measurements when determining the best result per each threshold percentage, a weighted norm needed to be used.

The process for determining the best SNR to CR value for one particular threshold value will be illustrated for the FWT. Let X be a $2 \times J$ matrix storing all the SNR and CR values for each possible filter bank iteration value J. Then for $1 \leq i \leq J$, let $x(i) = (x_{1i}\ x_{2i})^\intercal$ where x_{1i} is the i-th SNR value and x_{2i} is the i-th CR value. Let $\alpha_1 = 1/\|x_1\|_2^2$ be the inverse of the square of the 2-Norm of the J SNR values and $\alpha_2 = 1/\|x_2\|_2^2$ be the inverse of the square of the 2-Norm of the J CR values. Now let

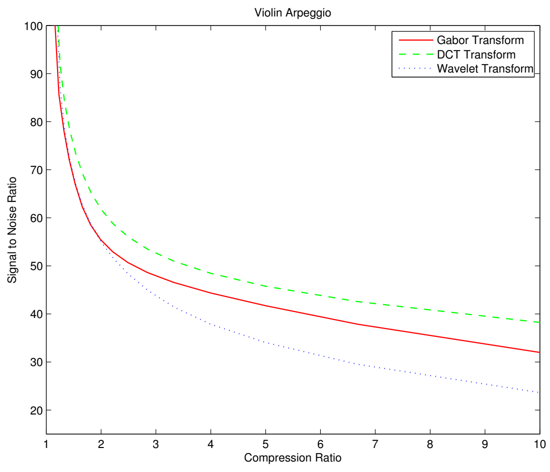
$$D = \begin{bmatrix} \alpha_1 & 0 \\ 0 & \alpha_2 \end{bmatrix} \tag{3.10}$$

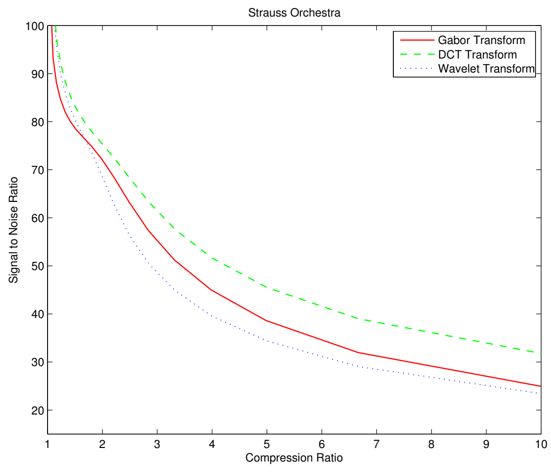
be the weighting matrix. Then the weighted norm (W-norm) is defined as $\|x(i)\|_W = \|x(i)^\intercal D x(i)\|_2$ and the best result is given by $\max\limits_{1 \leq i \leq J} \|x(i)\|_W$.

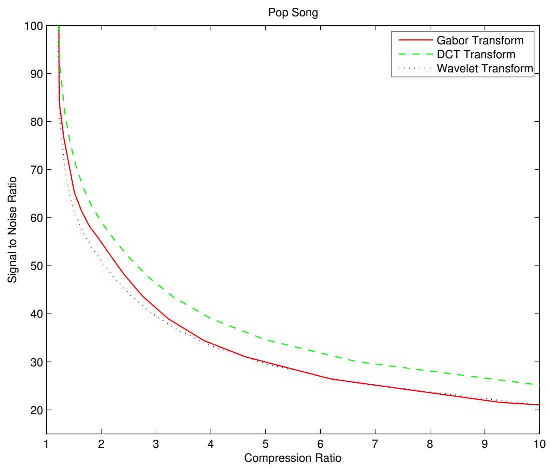
Repeating this process for all threshold percentages then leads to the final 2×19 matrix of SNR and CR results per file per transform type. Using these results, the best SNR vs CR function for each of the three transform techniques per audio file was plotted. A MatLab script was created to plot, per audio file, the best results from each of the three transform techniques. These plots follow in the next section.

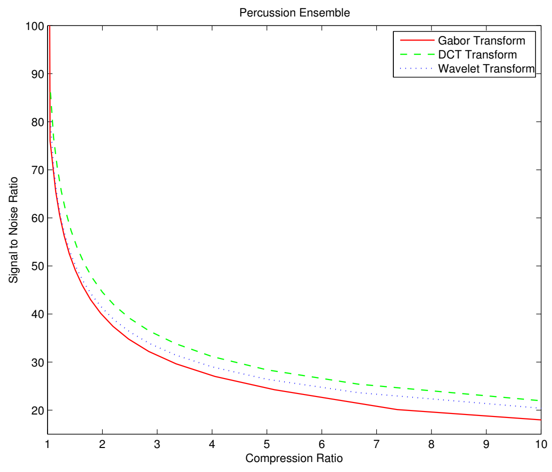
4 Results

The best performance function plot for each audio file and the three transform types is shown below. The best performance function is defined to be the highest SNR and CR overall per each threshold percentage and transform type as was defined in section 3.5.3.









5 Conclusion

It is clear from the plots in the preceding section that the DCT yielded the overall best performance on all four files tested. While the energy localization property of the DGT seemed to offer promise, further analysis into the choice of window function and lattice parameters would be required to determine whether that potential benefit would be able to counter the effect of the larger coefficient lattice that it generates. It is also interesting to note that the DGT behaved particularly poorly at compressing the percussion ensemble audio file. This seems to indicate that the Gaussian window used in the analysis is not particularly well-suited to transforming non-harmonic sounds with a widely distributed frequency spectrum.

Another point to note is that this investigation did not incorporate the use of a psycho-acoustic model or other adaptive thresholding technique or the use of encoding to further reduce the size of the compressed files.

Also, for this investigation, algorithmic complexity of the three transforms was not taken into consideration when rating the performance of the lossy audio coding algorithm. As was noted in Section 3.3, the DGT has the largest computational complexity of the three transform types and the FWT has the lowest computational complexity.

Thus, taking into consideration the performance results and computation considerations, it is quite evident why the DCT is used as the time-frequency transform in the popular MP3 lossy audio coding algorithm. Unless further gains can be made in reducing the computational complexity of the DGT or further maximizing the energy localization in the transform domain lattice, the DGT will remain unimplemented in modern lossy audio coding algorithms.

References

[1] Albert Boggess and Francis J. Narcowich. *A first course in wavelets with Fourier analysis.* John Wiley & Sons, Inc., Hoboken, NJ, second edition, 2009.

[2] Ole Christensen. *An introduction to frames and Riesz bases.* Applied and Numerical Harmonic Analysis. Birkhäuser Boston, Inc., Boston, MA, 2003.

[3] Karlheinz Gröchenig. *Foundations of time-frequency analysis.* Applied and Numerical Harmonic Analysis. Birkhäuser Boston, Inc., Boston, MA, 2001.

[4] Deguang Han, Keri Kornelson, David Larson, and Eric Weber. *Frames for undergraduates*, volume 40 of *Student Mathematical Library.* American Mathematical Society, Providence, RI, 2007.

[5] Christopher Heil. *A basis theory primer.* Applied and Numerical Harmonic Analysis. Birkhäuser/Springer, New York, expanded edition, 2011.

[6] Stéphane Mallat. *A wavelet tour of signal processing.* Elsevier/Academic Press, Amsterdam, third edition, 2009. The sparse way, With contributions from Gabriel Peyré.

[7] Henrique S. Malvar. *Signal Processing with Lapped Transforms.* Artech House, Inc., Norwood, MA, 1992.

[8] S. David Promislow. *A first course in functional analysis.* Pure and Applied Mathematics (Hoboken). Wiley-Interscience [John Wiley & Sons], Hoboken, NJ, 2008.

[9] Zdeněk Průša, Peter L. Søndergaard, Nicki Holighaus, Christoph Wiesmeyr, and Peter Balazs. The Large Time-Frequency Analysis Toolbox 2.0. In Mitsuko Aramaki, Olivier Derrien, Richard Kronland-Martinet, and Sølvi Ystad, editors, *Sound, Music, and Motion*, Lecture Notes in Computer Science, pages 419–442. Springer International Publishing, 2014.

[10] Peter Søndergaard. An efficient algorithm for the discrete gabor transform using full length windows. In *SAMPTA'09*, pages General–session, 2009.

[11] David F. Walnut. *An introduction to wavelet analysis.* Applied and Numerical Harmonic Analysis. Birkhäuser Boston, Inc., Boston, MA, 2002.

A Appendices

A.1 Contents of function Compression.m

```
function [CR, SNR, CF] = Compression( FN, a, M, J, CompPct, TType )
%This function runs a compression scheme that transforms a mono audio file
%into a time-frequency representation using one of three time-frequency
%transforms and then compresses the audio file by thresholding (nullifying)
%a certain percentage of the transform domain coefficients
%%%%%%%%%%%%%%%% Author: Michael Jorgen Olsen, 4/29/2015 %%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%% input parameters %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% FN = audio file name (including path if not in the default MatLab
%      directory)
% a = time increment (only for Gabor transform). Takes integer values >= 1
% M = number of frequency channels in T-F representation (for Gabor and
%     WMDCT). Takes integer values >= 1
% J = number of filterbank iterations (only for wavelet fransform). Takes
% integer values >= 1
% CompPct = percentage of threshold coefficients to nullify. Takes values
%           between 0 and 1)
% TType = transform type. 1 = FWT (fast wavelet transform), 2 = WMDCT
%         (windowed modified discrete cosine transform), 3 = DGT (discrete
%         Gabor transform)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%% output parameters %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% CR = compression ratio. Defined to be: (audio file length in
%      samples)/(number of nonzero coefficients after thresholding)
% SNR = signal to noise ratio. Defined to be: 20log((input norm)/(error
%       norm)
% CF = array storing compressed file (can use wavwrite afterwards to create
%      an actual audio file)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% read in audio file. y = array of sample values
y = wavread(FN);
% get length of file and number of channels. L(1) = length, L(2) = number
% of channels
L = size(y);

% only run compression algorithm on mono audio files
if L(2) == 1
    % Calculate L2 norm of original signal
    InputL2Norm = norm(y,'fro');

    % Compute coefficients for the appropriate transform
    if TType == 1
        % set filterbank tree properties (use DB8 wavelet)
        wtDef = {'db8',J,'dwt'};
        % perform FWT
        [coefs, wfInfo] = wfbt(y,wtDef);
    elseif TType == 2
        winLen = dwiltlength(L(1),M);
        % get an orthogonal window of the appropriate length
        g = wilorth(M,winLen);
        % perform WMDCT
        coefs = wmdct(y,g,M);
    elseif TType == 3
        % perform DGT
        complexCoefs = dgtreal(y,'gauss',a,M);
    else
        % notify user that they've not selected a valid transform id number
        error('Compression:transChk','The provided transform id is not valid. Only values 1
            = FWT, 2 = WMDCT and 3 = DGT are recognized.')
    end

    % if using Wavelet transform
    if TType == 1
        % transfer coefficients to a matrix
        wfbtMatrix = cell2mat(coefs);
        % store the dimensions of each cell column
        columns = zeros(1,length(coefs));
        for i = 1:length(coefs)
            columns(i) = length(coefs{i});
        end
        % set percentage of transform coefficients to retain
        x = floor(CompPct * length(wfbtMatrix));
        % threshold the matrix
        wfbtThreshMatrix = largestn(wfbtMatrix,x);
        % transform matrix back to cell array
        threshCoefs = mat2cell(wfbtThreshMatrix,columns,1);
        % calculate the compression ratio
        CR = L(1)/numel(nonzeros(wfbtThreshMatrix));
    elseif TType == 2
        % set percentage of transform coefficients to retain
        x = floor(CompPct * numel(coefs));

        % store the thresholded version of the T-F lattice
        threshCoefs = largestn(coefs,x);

        % calculate the compression ratio
        CR = L(1)/numel(nonzeros(threshCoefs));
    else
        % set percentage of transform coefficients to retain
        x = floor(CompPct * numel(complexCoefs));

        % store the thresholded version of the T-F lattice
        threshCoefs = largestn(complexCoefs,x);

        % since the real DGT returns complex coefficients, extract the real
        % and imaginary parts of the complex coefficients and store them
        % all in a matrix twice the size of the complex coefficient matrix
        coefs = zeros(2*size(threshCoefs,1),size(threshCoefs,2));
        coefs(1:size(threshCoefs,1),:) = real(threshCoefs);
        coefs((size(threshCoefs,1)+1):(2*size(threshCoefs,1)),:) = imag(threshCoefs);

        % calculate the compression ratio based on the number of non-zero
        % values in the real/imag pair coefficient matrix
        CR = L(1)/numel(nonzeros(coefs));
    end

    % Perform inverse transform to create compressed file
    if TType == 1
        % calculate the inverse wavelet transform
        tmp = iwfbt(threshCoefs,wfInfo);
        % return array of compressed file that is same length as input file
        CF = tmp(1:L(1),1);
    elseif TType == 2
        % calculate the appropriate dual window
        gd = wildual(g,M);
        % calculate the inverse WMDCT transform
        tmp = iwmdct(threshCoefs,gd);
        % return array of compressed file that is same length as input file
        CF = tmp(1:L(1),1);
    elseif TType == 3
        % calculate the inverse DGT transform and create return array
        CF = idgtreal(threshCoefs,'gaussdual',a,M,L(1));
    else
        error('Compression:transChk','The provided transform id is not valid. Only values 1,
             2, 3 are recognized.')
    end

    % calculate the residual difference between the compressed and original
    % signals
    ResL2Norm = norm(y−CF,'fro');

    % calculate the SNR ratio of signal to noise in compressed signal
    SNR = 20∗log10(InputL2Norm/ResL2Norm);
else
    % notify user that they must provide mono audio file
    error('Compression:fileChk','The provided audio file must be mono')
end

end
```

A.2 Contents of script RunCompression.m

```
% this script runs the lossy coding algorithm contained in Compression.m
% against four different monophonic audio files using a broad range of
% parameters for each of the three different time−frequency transforms used
% in Compression.m. For each file and transform type, the results are
% output to a 3−dimension matrix array whose name includes the acronym of
% the transform type used and also includes the name of the audio file
% processed.

%%%%%%%%%%%%%%%% Author: Michael Jorgen Olsen, 4/29/2015 %%%%%%%%%%%%%%%%

% string array of file locations to be processed
fNames = cell(1,4);
fNames{1} = 'F:\Mike\Documents\MATLAB\147_demo.wav';
```

```
fNames{2} = 'F:\Mike\Downloads\SQAM_FLAC\Waves\SingleInstruments\08-Mono.wav';
fNames{3} = 'F:\Mike\Downloads\SQAM_FLAC\Waves\PopMusic\69-Mono.wav';
fNames{4} = 'F:\Mike\Downloads\SQAM_FLAC\Waves\Orchestra\65-Mono.wav';
% string array to hold just file names without paths
sfNames = cell(1,4);
% populate sfNames with file names
% will be used for naming results matrices
for i = 1:4
    strTempName = fNames{i};
    strBackSlash = strfind(strTempName,'\');
    strEnd = strfind(strTempName,'.wav');
    strTempName = strrep(strTempName(strBackSlash(end)+1:strEnd-1),'-','');
    sfNames{i} = strTempName;
end

% parameters to be passed to Compression.m
% number of frequency channels (used for WMDCT and DGT)
M = [32 64 128 256 512 1024];
% time increment interval, M - 1 is best choice (used for DGT)
a = M - 1;
% number of filterbank iterations (used for FWT)
J = [1 2 3 4 5 6 7 8 9 10];
% number of threshold values (95% - 5%)
CRs = [0.95 0.9 0.85 0.8 0.75 0.7 0.65 0.6 0.55 0.5 0.45 0.4 0.35 0.3 0.25 0.2 0.15 0.1 0.05];

% cycle through transform types
for t = 3:3
    % set parameter holding first 3 chars of output matrix name
    if t == 1
        strType = 'Fwt';
    elseif t == 2
        strType = 'Dct';
    else
        strType = 'Dgt';
    end
    % cycle through audio files
    for k = 1:length(fNames)
        %allocate a 3-dimensional array to hold SNR and Comp Ratio per M
        strTempName = sfNames{k};
        strFileName = fNames{k};
        numChannels = length(M);
        numIterations = length(J);
        % create
        if t == 1
            str = sprintf('%sResults%s = zeros(2,%d,%d);',strType,strTempName,length(CRs),numIterations);
        else
            str = sprintf('%sResults%s = zeros(2,%d,%d);',strType,strTempName,length(CRs),numChannels);
        end
        eval(str);

        % if FWT
        if t == 1
            % run for all J values for FWT
            for h = 1:numIterations
                % run for all CR values
                for i = 1:length(CRs)
                    % create dynamic cmd to run function w/correct parameters and populate
                    % correct row in correct matrix with resultant SNR and CR values
                    str = sprintf('[CompRat SigToNoise OutFile] = Compression(''%s'',%d,%d,%d,%f,%d);',strFileName,0,0,J(h),CRs(i),t);
                    eval(str);
                    str = sprintf('%sResults%s(1,%d,%d) = CompRat;',strType,strTempName,i,h);
                    eval(str)
                    str = sprintf('%sResults%s(2,%d,%d) = SigToNoise;',strType,strTempName,i,h);
                    eval(str)
                end
            end
        else %if DCT or DGT
            % run for all M values for DCT and DGT
            for h = 1:numChannels
                % run for all CR values
                for i = 1:length(CRs)
                    % create dynamic cmd to run function w/correct parameters and populate
                    % correct row in correct matrix with resultant SNR and CR values
                    if t == 2
                        str = sprintf('[CompRat SigToNoise OutFile] = Compression(''%s'',%d,%d,%d,%f,%d);',strFileName,0,M(h),0,CRs(i),t);
                        eval(str);
                        str = sprintf('%sResults%s(1,%d,%d) = CompRat;',strType,strTempName,i,h);
                        eval(str)
                        str = sprintf('%sResults%s(2,%d,%d) = SigToNoise;',strType,strTempName,i,h);
                        eval(str)
                    else
                        str = sprintf('[CompRat SigToNoise OutFile] = Compression(''%s'',%d,%d,%d,%f,%d);',strFileName,a(h),M(h),0,CRs(i),t);
                        eval(str);
                        str = sprintf('%sResults%s(1,%d,%d) = CompRat;',strType,strTempName,i,h);
                        eval(str)
                        str = sprintf('%sResults%s(2,%d,%d) = SigToNoise;',strType,strTempName,i,h);
                        eval(str)
                    end
                    % run the compression function and populate to appropriate array and
                    % row
                end
            end
        end
    end
end
```