4-30-2013

Fourier and Walsh Transform Models for JPEG Compression

Nathan Bush

Louisiana State University

Honors Thesis

Fourier and Walsh Transform Models for JPEG Compression

Author:
Nathan Bush

Advisor:
Dr. Richard Oberlin

April 30, 2013

Abstract. This paper discusses discrete Fourier and Walsh transforms of one and two dimensions. The paper then describes how to optimize both the Fourier and Walsh transforms to get fast transforms capable of performing matrix operations quickly and efficiently. Using fast two-dimensional transforms, the objective is to remove high frequency data from an image to simulate a JPEG compression algorithm.

1. Introduction

JPEG compression and image processing require algorithms that remove data while preserving an image still recognizable to the human eye. The algorithms remove data at certain frequencies to accomplish this task. While the given function for an image does not have a domain consisting of frequencies, there are transforms capable of converting a function's time domain into a frequency domain.

Both the discrete Fourier transform and Walsh transform are capable of converting a function into a frequency domain function. Accordingly, in this paper, we define both transforms in one and two dimensions and find a method of reducing the number of operations required for their calculation. We then define a function to remove data at certain frequencies to compress a given image. Following this compression, we restore the image using the inverse discrete Fourier transform and Walsh transforms.

2. Frequency Plots

Recall that a function is *periodic* with *period* p and *frequency* $\frac{1}{p}$ if $f(x+p) = f(x)$ for all x.

A *time-domain* plot illustrates the values of a function of time with time on the x-axis and the values of the function on the y-axis.

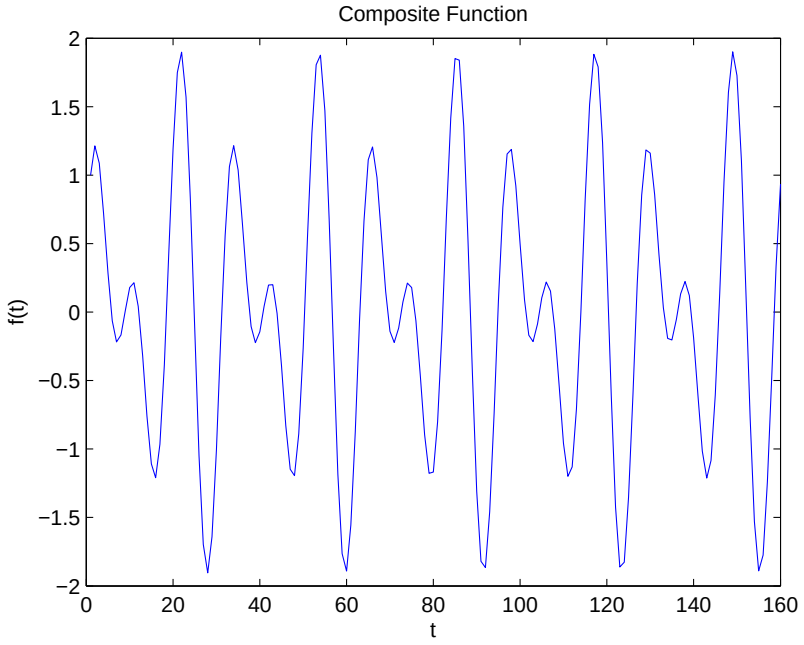
We note that in this context, the time domain refers to the initial domain rather than a physical quantity of time, since we are analyzing images and not sound waves.

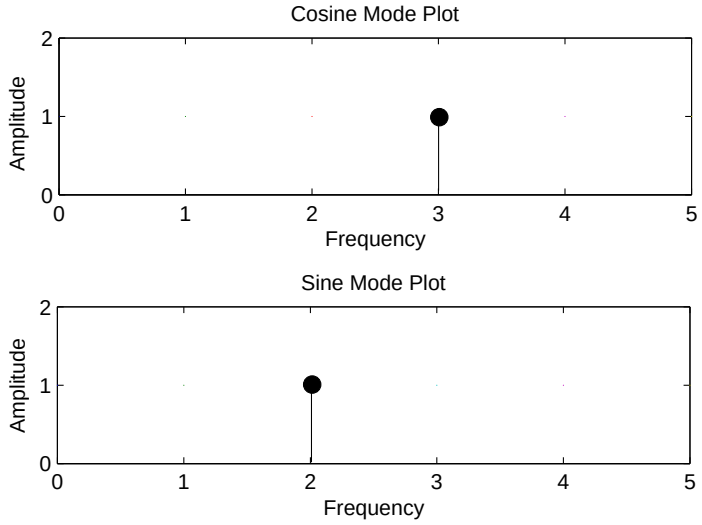
A *frequency-domain* plot or *stem* plot illustrates the amplitude of the individual addends of a composite periodic function.

One type of frequency-domain plotting uses two plots, one called the *sine mode* plot and the other called the *cosine mode* plot. The sine

mode plot and cosine mode plot for a composite periodic function are frequency-domain plots of the sine components and cosine components respectively.

Example 2.1. Below, we plot the sine and cosine modes of the function $f(t) = cos(2\pi t3) + sin(2\pi t2)$.





Although in the first plot, f(t) is periodic and has its own period and frequency, note that it is not possible to determine the stem plot from only the frequency of the composite function. In this sense, it is possible to identify the original function f(t) only by considering both the plot of the composite f(t) and the stem plot.

2.1. **Complex Exponential Representation.** It is often convenient to represent the trigonometric functions in terms of complex exponentials by using Euler's formula.

Lemma 2.2. *If* $i = \sqrt{-1}$, $cos\,\theta = \frac{e^{i\theta}+e^{-i\theta}}{2}$ *and* $sin\,\theta = \frac{e^{i\theta}-e^{-i\theta}}{2i}$.

Proof. By Euler's formula,

$$\begin{aligned}\frac{e^{i\theta}+e^{-i\theta}}{2} &= \frac{cos\,\theta + isin\,\theta + cos(-\theta) + isin(-\theta)}{2}\\ &= \frac{cos\,\theta + cos\,\theta + isin\,\theta - isin\,\theta}{2}\\ &= \frac{2cos\,\theta}{2}\\ &= cos\,\theta.\end{aligned}$$

Similarly,

$$\begin{aligned}\frac{e^{i\theta}-e^{-i\theta}}{2i} &= \frac{cos\,\theta + isin\,\theta - (cos(-\theta)+isin(-\theta))}{2i}\\ &= \frac{cos\,\theta - cos\,\theta + isin\,\theta + isin(\theta)}{2i}\\ &= \frac{2isin\,\theta}{2i}\\ &= sin\,\theta.\end{aligned}$$

□

The following proposition and proof are similar to those found in [3].

Proposition 2.3. *If* $f(t)=\sum_{k=1}^{n} A_k cos(2\pi f_k t)+B_k sin(2\pi f_k t)$*, where* $A_0=0$*, then f(t) can be written as a linear combination of complex exponential terms.*

Proof. By Lemma 2.4, if $X_{\pm}k=\frac{A_k \mp iB_k}{2}$, $X_0=0$, and $f_{-k}=-f_k$,

$$\begin{aligned}f(t) &= \sum_{k=1}^{n} A_k cos(2\pi f_k t)+B_k sin(2\pi f_k t)\\ &= \sum_{k=1}^{n} A_k \frac{e^{i2\pi f_k t}+e^{-i2\pi f_k t}}{2} + B_k \frac{e^{i2\pi f_k t}-e^{-i2\pi f_k t}}{2i}\\ &= \sum_{k=1}^{n} \frac{A_k - iB_k}{2}\, e^{i2\pi f_k t} + \frac{A_k + iB_k}{2}\, e^{-i2\pi f_k t}\\ &= \sum_{k=1}^{n} X_k\, e^{i2\pi f_k t} + X_{-k}\, e^{i2\pi f_{-k} t}\\ &= X_0 + \sum_{k=1}^{n} X_k\, e^{i2\pi f_k t} + \sum_{k=1}^{n} X_{-k}\, e^{i2\pi f_{-k} t}\\ &= X_0 + \sum_{k=1}^{n} X_k\, e^{i2\pi f_k t} + \sum_{k=-n}^{-1} X_k\, e^{i2\pi f_k t}\\ &= \sum_{k=-n}^{n} X_k\, e^{i2\pi f_k t}\end{aligned}$$

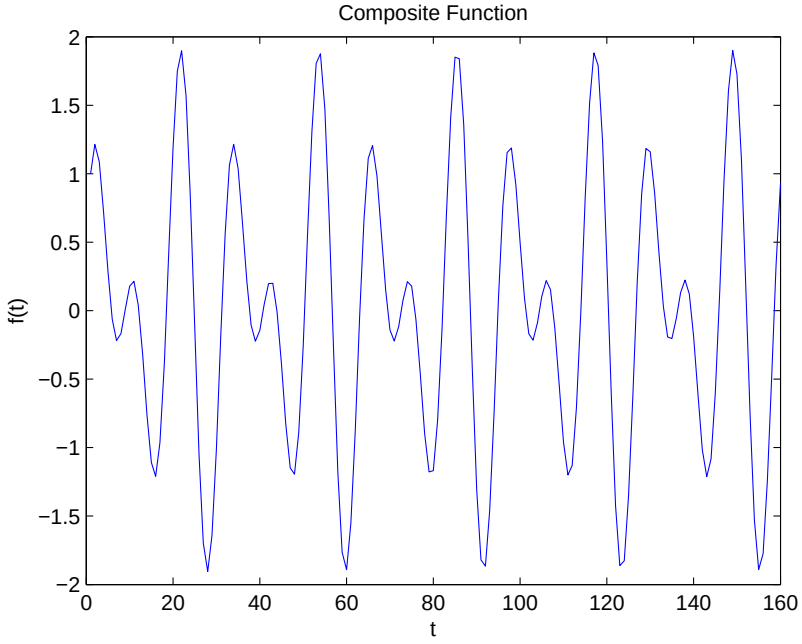
□

We note for future reference that these X_k's are the same as those in the DFT definition in the next section.

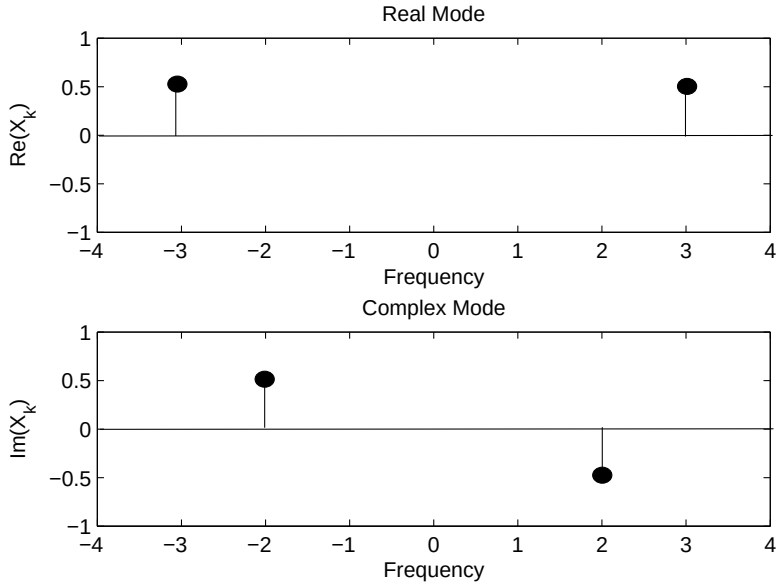
Another type of frequency-domain plot is called *complex exponential mode* frequency plot, where the plot is separated into a *real mode* plot

and an *imaginary mode* plot. $Re(x_k)$ and $Im(X_k)$ from the above proof correspond to the amplitudes at frequency f_k on the real mode and imaginary mode plots respectively.

Example 2.4. Below, we plot the complex exponential modes of the function $f(t) = cos(2\pi t3) + sin(2\pi t2)$.



Below is the complex exponential mode plot of f(t).



This is important in the context of the DFT, because when the transform changes a time domain function into a frequency domain function, these frequencies are the frequncies of the individual sine and cosine functions that sum to get f(t) and are not pertaining to the fequency of the whole function f(t).

3. One-Dimensional Transforms

We start by discussing two methods for writing periodic functions on one-dimensional domains (continuous and discrete) as a series of complex exponential functions of various frequencies.

3.1. **The Fourier transform.** The "character" functions used in the Fourier transform of a function of period p are based on the trigonometric functions $\sin(t)$ and $\cos(t)$ (later we will show that it can be convenient to rewrite these in terms of complex exponentials). Specifically for each positive integer k, consider the functions $\sin(2\pi kt/p)$ and $\cos(2\pi kt/p)$ which have "frequency" k on the domain $[-p/2, p/2]$ (i.e. they complete k cycles on that time interval).

The "Fourier-coefficients"

$$A_k := \frac{2}{p}\int_{-p/2}^{p/2} f(t)cos(\frac{2\pi kt}{p})dt, k = 0, 1, 2, \ldots$$

$$B_k := \frac{2}{p}\int_{-p/2}^{p/2} f(t)sin(\frac{2\pi kt}{p})dt, k = 1, 2, 3, \ldots$$

measure how closely f corresponds to the pure tone functions. One can then attempt to reconstruct f as a superpositioning of its components by using a "Fourier-series."

$$S[f](t) := \frac{A_0}{2} + \sum_{k=1}^{\infty} A_k cos(\frac{2\pi kt}{p}) + B_k sin(\frac{2\pi kt}{p}). \tag{3.1}$$

It is natural to ask whether the series defining $S[f]$ converges to f (or indeed, does it even converge at all?). This question is beyond the scope of our present investigation, but we will at least mention one of the first steps, given many years ago by Dirichlet, towards an answer:

Theorem 3.1. *If f(t) is a real-valued function defined on* $\mathbb{R}$ *and it satisfies the Dirichlet conditions:*
i. f(t) is bounded on any bounded closed subinterval [a,b];
ii. f(t) had only a finite number of maxima and minima on any interval [a,b];
iii. f(t) has on [a,b] at most a finite number of dicontinuities, each of which is a jump discontinuity;
iv. f(t) is periodic with period p;
then for every t at which f is continuous, f(t) equals its Fourier series, and its Fourier series is convergent at t.

In the discrete case below, there is no question of convergence since the sum is finite, and the fact that $f = S[f]$ essentially follows from the fact that the complex exponential functions form an orthogonal basis of the n-dimensional function space.

3.2. **The Discrete Fourier Transform.** In the previous section we considered functions f of period p defined on $\mathbb{R}$; these were effectively represented by their restriction to one period p, i.e. the interval $[0, p]$. For computational purposes it is not usually feasible to work with such functions since they encode an infinite amount of information. In this section we will thus consider functions defined on a finite subset of $[0, p]$ (and henceforth, for convenience, let us assume that the period $p = 1$.) These functions represent periodic functions whose domains are discrete subsets of $\mathbb{R}$.

Definition 3.2. The $N \times N$ *Fourier matrix* is denoted ${}^{N}_{F}\Omega$. For the entry in the kth row and lth column of this matrix, ${}^{N}_{F}\Omega_{k,l} := e^{\frac{i(k-1)(l-1)2\pi}{N}}$.

Lemma 3.3. *For any geometric sequence,* $\{ra^i\}_{i=0}^{k-1}$ *for* $a, r \in \mathbb{C}, a \neq 0, 1, \sum_{i=0}^{k-1} ra^i = \frac{r(1-a^k)}{1-a}$.

Proof. First observe that

$$\begin{aligned}\sum_{i=0}^{k-1} ra^i - \sum_{i=0}^{k} ra^{i+1} &= r + ra + \cdots + ra^{k-1} - (ra + ra^2 + \cdots + ra^k) \\ &= r + 0 + \cdots + 0 - ra^k \\ &= r - ra^k \,.\end{aligned}$$

But since

$$\sum_{i=0}^{k-1} ra^i - \sum_{i=0}^{k} ra^{i+1} = \sum_{i=0}^{k-1} ra^i - a\sum_{i=0}^{k-1} ra^i = (1-a)\sum_{i=0}^{k-1} ra^i = r - ra^k,$$

$$\sum_{i=0}^{k-1} ra^i = \frac{r(1-a^k)}{1-a},$$

which concludes our proof. □

Theorem 3.4. *Considering the $m \times m$ Fourier matrix, the collection of row s $\eta_0, ..., \eta_{m-1}$ is an orthonormal basis.*

Proof. Considering the $m \times m$ Fourier matrix,

$$\begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & e^{i\frac{2\pi}{m}1} & e^{i\frac{2\pi}{m}2} & \cdots & e^{i\frac{2\pi}{m}(m-1)} \\ 1 & e^{i\frac{2\pi}{m}2} & e^{i\frac{2\pi}{m}4} & \cdots & e^{i\frac{2\pi}{m}2(m-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & e^{i\frac{2\pi}{m}(m-1)} & e^{i\frac{2\pi}{m}2(m-1)} & \cdots & e^{i\frac{2\pi}{m}(m-1)(m-1)} \end{bmatrix}$$

if $\eta_0, ..., \eta_{m-1}$ denote the m rows of this matrix, then for $p \neq v$, by Lemma 3.3,

$$\begin{aligned}\langle \eta_p | \eta_v \rangle &= \sum_{j=0}^{m-1} e^{\frac{i2\pi}{m}j(p+v)} \\ &= \frac{1 - e^{\frac{i2\pi}{m}m(p+v)}}{1 - e^{\frac{i2\pi}{m}(p+v)}} \\ &= \frac{1 - e^{i2\pi(p+v)}}{1 - e^{\frac{i2\pi}{m}(p+v)}} \\ &= \frac{1 - (\cos(2\pi(p+v)) + i\sin(2\pi(p+v)))}{1 - e^{\frac{i2\pi}{m}(p+v)}} \\ &= \frac{1 - (1+0)}{1 - e^{\frac{i2\pi}{m}(p+v)}} \\ &= 0.\end{aligned}$$

□

Because $\eta_0, ..., \eta_{m-1}$ is an orthonormal basis, $x = \sum_{j=0}^{m-1} \eta_j \langle x|\eta_j\rangle_m$, where if x_k and η_{j_k} represent the kth coordinates of a vector x and the row vector η_j respectively, the inner product $\langle x|\eta_j\rangle_m = \frac{1}{m}\sum_{k=0}^{m-1} x_k \cdot \eta_{j_k}$. This leads to the following definition.

Definition 3.5. If X_r denotes the rth coordinate in the DFT of a vector f, and x_k denotes the kth coordinate of f, it is possible to express each coordinate of the DFT as:

$$X_r = \frac{1}{\sqrt{n}} \sum_{k=0}^{n-1} x_k e^{-irk2\pi/n},\ r = 0, 1, ..., n-1.$$

We can define the inverse discrete transform using the DFT and the derivation of the DFT from the Fourier matrix.

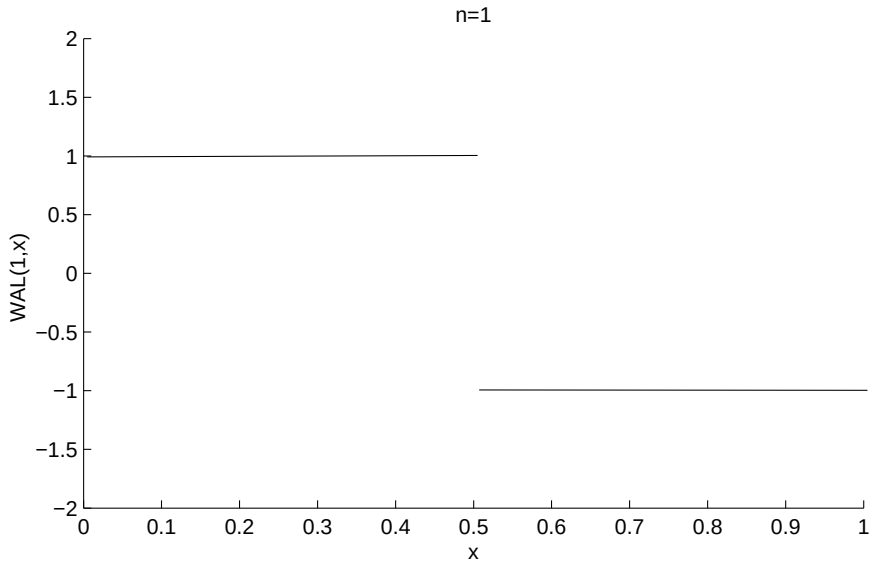
Definition 3.6. If we say $X_r^{-1} = x_r$ denotes the rth coordinate in the inverse DFT, denoted IDFT, of a function f, and x_k denotes the kth coordinate of f, it is possible to express each coordinate of the IDFT as:

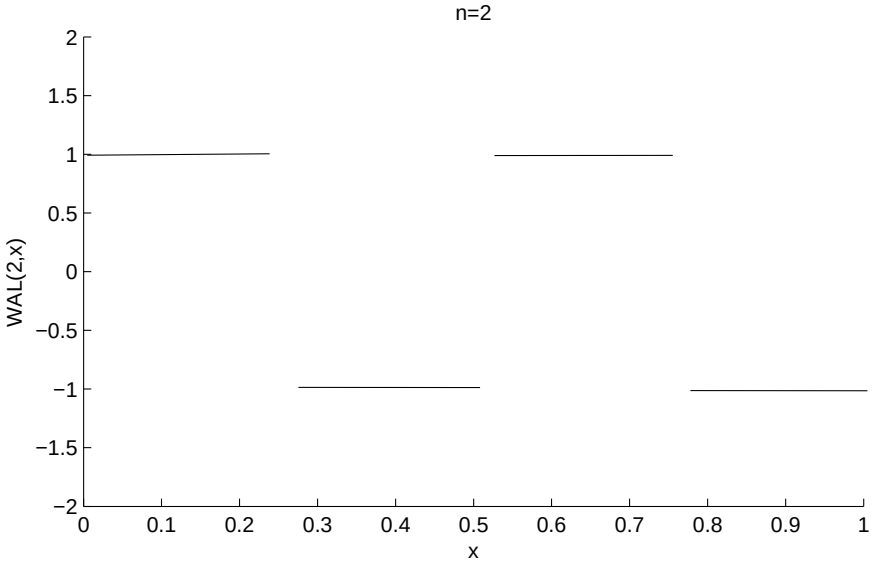
$$X_r^{-1} = \frac{1}{\sqrt{n}} \sum_{k=0}^{n-1} X_k e^{irk2\pi/n},\ r = 0, 1, ..., n-1.$$

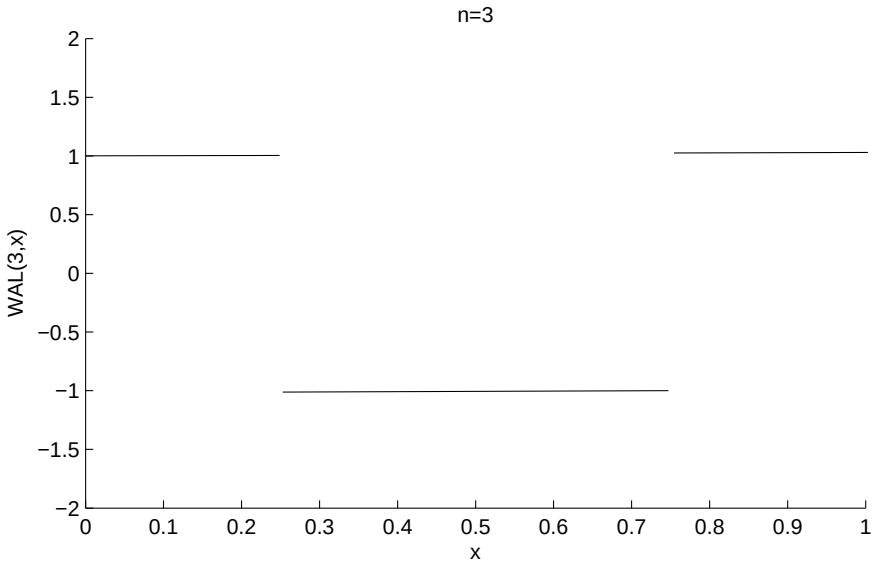
4. The Walsh Transform

Definition 4.1. The Walsh function, denoted WAL(n, x), where $n \in \mathbb{N}$ and $x \in [0, 1]$, is defined by WAL$(n, x) = \prod_{k \in \mathbb{Z}} (-1)^{n_k x_{-1-k}}$, where n_i denotes the coefficient of 2^i in the binary expansion of n.

Example 4.2. We plot the Walsh function for n= 1,2, and 3 to illustrate the function.


n=1
WAL(1,x)
x
2
1.5
1
0.5
0
−0.5
−1
−1.5
−2
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1


n=2
WAL(2,x)
x
2
1.5
1
0.5
0
−0.5
−1
−1.5
−2
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1



This example helps illustrate the property that $WAL(a,x)WAL(b,x) = WAL(a \oplus b, x)$, where $\oplus$ denotes addition in base 2 in which one does not "carry a 1" as in base 10 addition.

To illustrate $\oplus$ addition, we use the following example.

Example 4.3. $111_2 \oplus 11_2 = 100_2$.

4.1. **Discrete Walsh Transform.** Using the Walsh function in the place of the DFT's complex exponential function, it is possible to define a discrete Walsh transform in the same way as the DFT.

Definition 4.4. If X_r denotes the rth coordinate in the discrete Walsh transform (DWT) of a function f, and x_k denotes the kth coordinate of f, we can express each coordinate of the DWT as:

$$X_r = \frac{1}{\sqrt{n}} \sum_{k=0}^{n-1} x_k WAL(k, \frac{r}{n}),\ r = 0, 1, ..., n-1.$$

One useful property of the DWT is that the inverse of the DWT is the DWT itself.

5. Fast algorithms for discrete transforms

Although discrete transforms are sufficient for studying frequency domain functions, their computation time is slow when working with a large amount of data. Therefore, there is motivation to create an algorithm to compute the transform in less time than the discrete method.

5.1. **Fast Fourier Transform.** Referring to the X_r from the DFT,

$$\begin{aligned} X_r &= \frac{1}{\sqrt{n}} \sum_{k=0}^{n-1} x_k e^{-irk2\pi/n},\ r = 0, 1, ..., n-1 \\ &= \frac{1}{\sqrt{n}} \sum_{k=0}^{\frac{n}{2}-1} x_{2k} e^{-ir2k2\pi/n} + \frac{1}{\sqrt{n}} \sum_{k=0}^{\frac{n}{2}-1} x_{2k+1} e^{-ir(2k+1)2\pi/n} \\ &= \frac{1}{\sqrt{n}} \sum_{k=0}^{\frac{n}{2}-1} x_{2k} e^{-ir2k2\pi/n} + \frac{e^{ir2\pi/n}}{\sqrt{n}} \sum_{k=0}^{\frac{n}{2}-1} x_{2k+1} e^{-ir2k2\pi/n}. \end{aligned}$$

In this way, DFT of a function breaks down into a sum of the DFT of the terms of the function with even indices and the DFT of the terms of the function with odd indices.

This recursive separation of the DFT into two smaller sums is called the ***fast Fourier transform***, denoted FFT.

Similarly,

$$\begin{aligned} X_r^{-1} &= \frac{1}{n} \sum_{k=0}^{n-1} X_k e^{irk2\pi/n},\ r = 0, 1, ..., n-1 \\ &= \frac{1}{\sqrt{n}} \sum_{k=0}^{\frac{n}{2}-1} X_{2k} e^{ir2k2\pi/n} + \frac{1}{\sqrt{n}} \sum_{k=0}^{\frac{n}{2}-1} X_{2k+1} e^{ir(2k+1)2\pi/n} \\ &= \frac{1}{\sqrt{n}} \sum_{k=0}^{\frac{n}{2}-1} X_{2k} e^{ir2k2\pi/n} + \frac{e^{ir2\pi/n}}{\sqrt{n}} \sum_{k=0}^{\frac{n}{2}-1} X_{2k+1} e^{ir2k2\pi/n}. \end{aligned}$$

This recursive separation of the IDFT into two smaller sums is called the ***inverse fast Fourier transform***, denoted IFFT.

5.2. **Fast Walsh Transform.** Referring to the X_r from the DWT,

$$\begin{aligned} X_r &= \frac{1}{\sqrt{n}} \sum_{k=0}^{n-1} x_k WAL(k, \frac{r}{n}),\ r = 0, 1, ..., n-1 \\ &= \frac{1}{\sqrt{n}} \sum_{k=0}^{\frac{n}{2}-1} x_{2k} WAL(2k, \frac{r}{n}) + \frac{1}{\sqrt{n}} \sum_{k=0}^{\frac{n}{2}-1} x_{2k+1} WAL(2k \oplus 1, \frac{r}{n}) \\ &= \frac{1}{\sqrt{n}} \sum_{k=0}^{\frac{n}{2}-1} x_{2k} WAL(2k, \frac{r}{n}) + \frac{WAL(1, \frac{r}{n})}{\sqrt{n}} \sum_{k=0}^{\frac{n}{2}-1} x_{2k+1} WAL(2k, \frac{r}{n}). \end{aligned}$$

In this way, the DWT of a function breaks down into a sum of the DWT of the terms of the function with even indices and the DWT of the terms of the function with odd indices.

This recursive separation of the discrete Walsh transform into two smaller sums is called the ***fast Walsh transform***, denoted FWT.

We note that since the inverse of the DWT is itself, the inverse of FWT is FWT, meaning there is no need for a separate inverse definition.

5.3. **Transform Efficiency.** To understand why the fast transforms are more efficient than the discrete transforms, it is necessary to consider the difference in the number of arithmetic operations required to perform each transform.

If N is length of the vector to which we apply the transform, there are N^2 arithmetic operations required to complete the discrete transforms.

On the other hand, there are only $5NLog_2(N)$ required operations for the fast transforms.

If $N = 2^r$ for some $r \in \mathbb{N}$, $5NLog_2(N) = 2^r 5Log_2(2^r) = 5r2^r$.

By preservation of order, since $2^r > 5r$ when $r \geq 5$, $N^2 = 2^r 2^r > 2^r 5r = 5NLog_2(N)$ when $r \geq 5$, meaning the FFT requires less arithmetic operations when $r \geq 5$ and moreover when N is large.

One can find a proof of the number of algorithms for both transforms in [2].

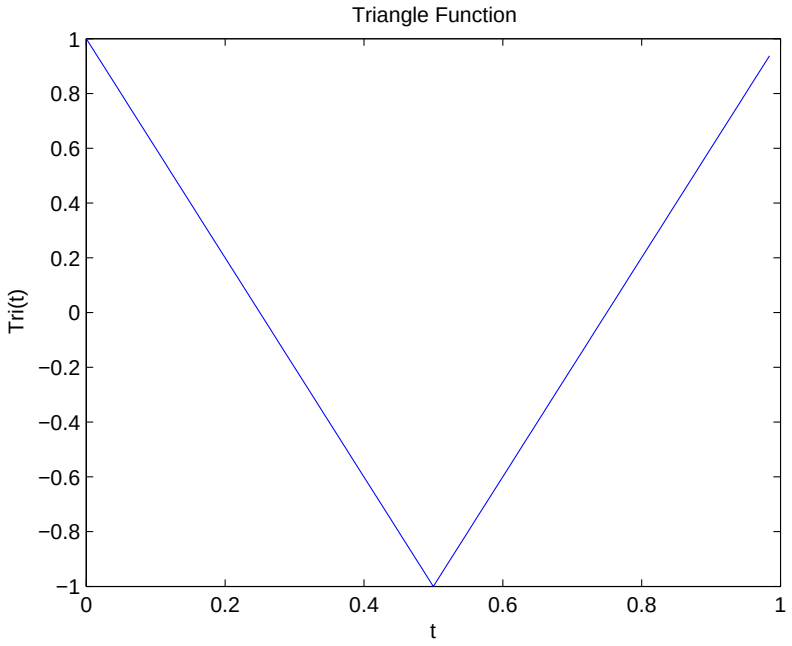
6. One-Dimensional Transform Examples

The first plot to consider is that of:

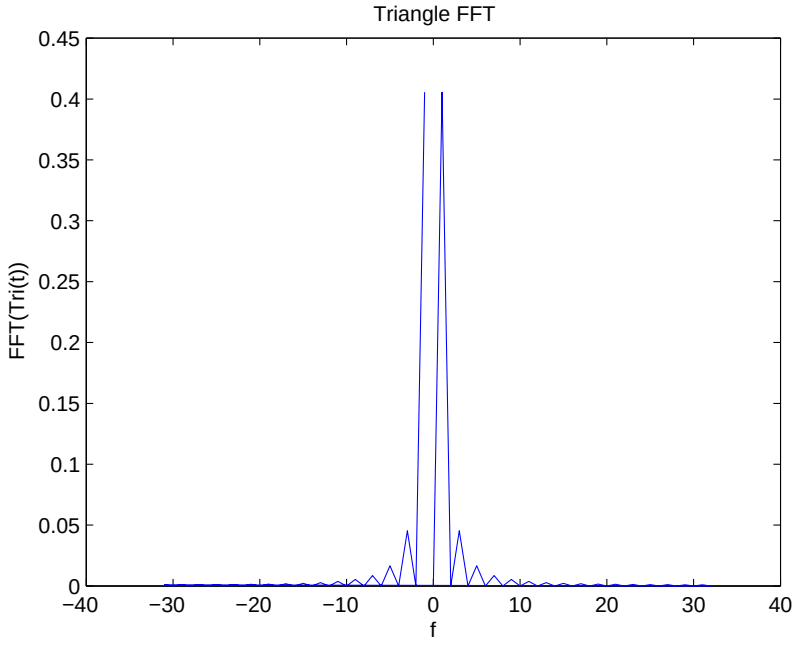
$$f(t) = \begin{cases} -4t + 1 & : t \in [0, \frac{1}{2}] \\ -4(1-t) + 1 & : t \in (\frac{1}{2}, 1] \end{cases}$$

on [0,1].

In each plot, we evaluate f(t) on the 256 points $\{0, \frac{1}{256}, \frac{2}{256}, ..., \frac{255}{256}\}$.



We note that the FFT plot below has several spikes in magnitude, meaning there are cosine functions of several frequencies required to model f(t).

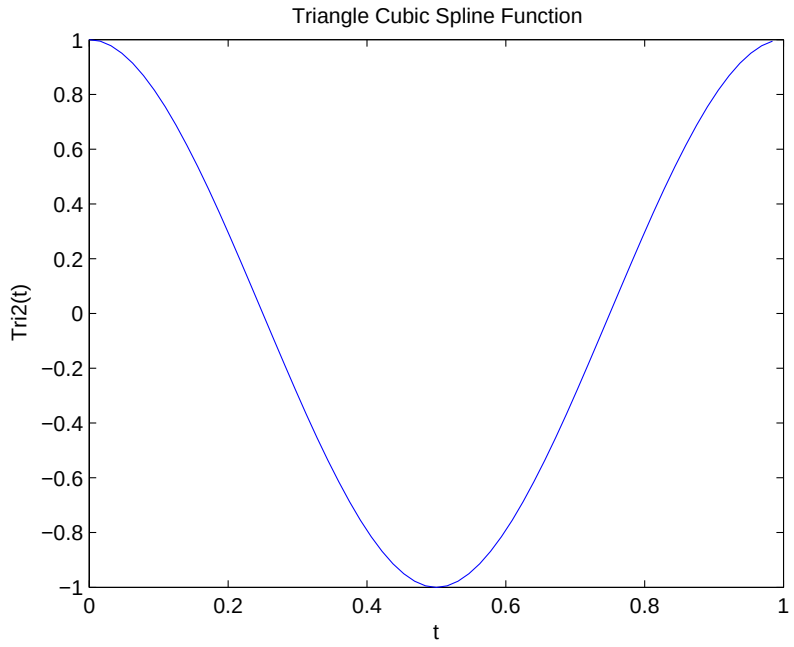


Below is the plot of:

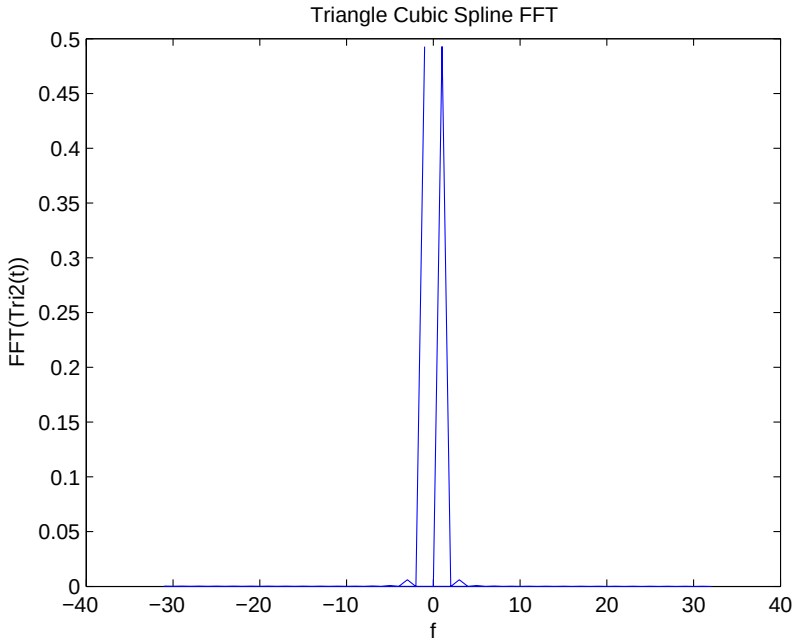
$$s(t) = \begin{cases} 1 - 24t^2 + 32t^3 & : t \in [0, \frac{1}{2}] \\ 1 - 24(1-t) + 32(1-t)^3 & : t \in (\frac{1}{2}, 1] \end{cases}$$

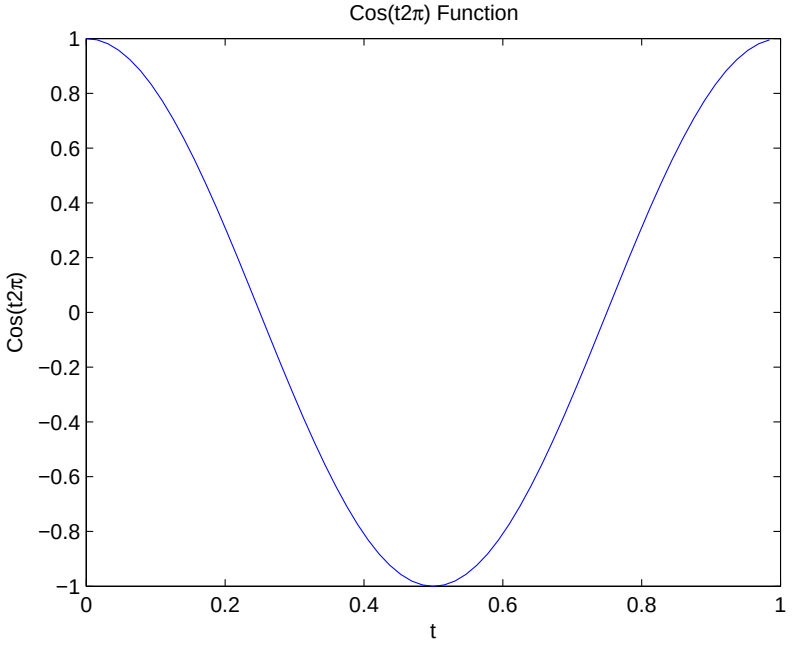
on [0,1].

In fact, $s(t)$ is called a *cubic spline* of $f(t)$, and its purpose is to approximate trigonometric functions more closely than $f(t)$.

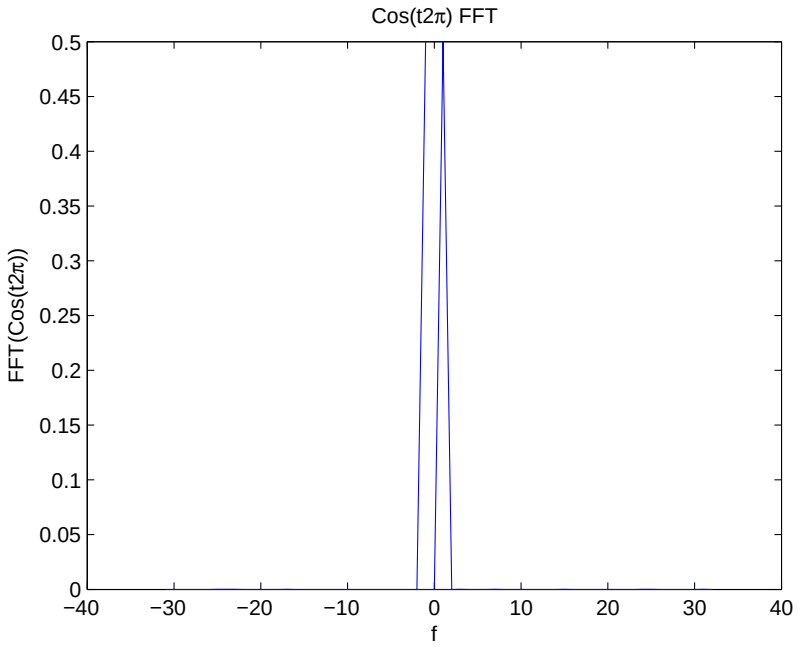


Because the cubic spline of f(t) approximates $cos(2\pi t)$ more closely than f(t), the FFT plot shows fewer spikes in magnitude since there are fewer required trigonometric functions to model s(t).





Although the plot of $cos(2\pi t)$ is similar to that of s(t), there are still small spikes in the FFT plot of s(t) that are not present in the FFT plot of $cos(2\pi t)$, as $cos(2\pi t)$ is a trigonometric function.



7. Two-Dimensional Transforms

Definition 7.1. If $x_{l,m}$ denotes the entry of an $N \times N$ matrix in the row $l+1$ and column $m+1$ and $X_{i,k}$ denotes the entry of the two-dimensional discrete Fourier transform in row $j+1$ and column $k+1$,

$$X_{j,k} = \sum_{l=0}^{N-1} \sum_{m=0}^{N-1} x_{l,m} e^{-2\pi i(j)(k)/N} e^{-2\pi i(l)(m)/N}.$$

The fast two-dimensional transforms require the one-dimensional transforms. Moreover, we complete the fast two-dimensional transforms by applying either the FFT or FWT to both the rows and the columns of the given matrix.

Definition 7.2. If R_i is the ith row of some $2^N \times 2^N$ matrix which we call M, define $R_r^{'} = FFT(R_r)$, and say $M^{'}$ is the matrix with rows $R_0^{'}, ..., R_{N-1}^{'}$. Then, if C_i is the ith row of $(M^{'})^T$, define $C_r^{'} = FFT(C_r)$, and say $M^{''}$ is the matrix with rows $C_0^{'}, ..., C_{N-1}^{'}$. We define the *two-dimensional fast Fourier transform*, denoted MFFT, by $(M^{''})^T$.

The two-dimensional Walsh transforms are almost identical to the two-dimensional Fourier transforms, with the only difference being the change to the Walsh function from a complex exponential function.

Definition 7.3. If $x_{l,m}$ denotes the entry of an $N \times N$ matrix in row l+1 and column m+1 and $X_{i,k}$ denotes the entry of the two-dimensional

discrete Walsh transform in the row j+1 and column k+1,

$$X_{j,k} = \sum_{l=0}^{N-1} \sum_{m=0}^{N-1} x_{l,m} WAL(j, \frac{k}{N}) WAL(l, \frac{m}{N}).$$

Definition 7.4. If R_i is the ith row of some $2^N \times 2^N$ matrix which we call M, define $R_r^{'} = FWT(R_r)$, and say $M^{'}$ is the matrix with rows $R_0^{'}, ..., R_{N-1}^{'}$. Then, if C_i is the ith row of $(M^{'})^T$, define $C_r^{'} = FWT(C_r)$, and say $M^{''}$ is the matrix with rows $C_0^{'}, ..., C_{N-1}^{'}$. We define the *two-dimensional fast Walsh transform*, denoted MFWT, by $(M^{''})^T$.

Because the inverse of the FWT is FWT, the inverse of MFWT is MFWT by the definition of MFWT. On the other hand, the MFFT requires the following inverse definition:

Definition 7.5. If R_i is the ith row of some $2^N \times 2^N$ matrix which we call M, define $R_r^{'} = IFFT(R_r)$, and say $M^{'}$ is the matrix with rows $R_0^{'}, ..., R_{N-1}^{'}$. Then, if C_i is the ith row of $(M^{'})^T$, define $C_r^{'} = IFFT(C_r)$, and say $M^{''}$ is the matrix with rows $C_0^{'}, ..., C_{N-1}^{'}$. We define the *inverse two-dimensional fast Fourier transform*, denoted IMFFT, by $(M^{''})^T$.

8. Model for JPEG Compression

8.1. **JPEG Background.** The images we will work with will consist of grayscale matrices containing entries, called pixels, between 0 and 255, meaning there are $Log_2(2^8) = 8$ bits per pixel.

If the entry is 0, it corresponds to the pixel color black, and if the entry is 255, its pixel color is white, with the intermediate values representing corresponding shades of gray.

JPEG compression refers to a certain method for removing data from an image. JPEG compression is effective, because it should remove data while maintaining an image whose appearance to the human eye is as close as possible to the original image.

Using Matlab, we obtain the grayscale matrix of an image and apply compression functions similar to the basic idea behind JPEG compression.

Example 8.1. The image below is the original 64×64 image.

FIGURE 1. 64×64 Image

FIGURE 2. 24:40 FFT Compression

The compression algorithm used in Figure 2 performs the MFFT on the JPEG matrix and then removes rows 20-40 and columns 24-40 in the matrix by replacing the pixels by 0 in these rows and columns. The above image is the IMFFT of the resulting matrix.

FIGURE 3. 16:48 FFT Compression

The compression algorithm used in Figure 3 performs the MFFT on the grayscale matrix and then removes rows 16-48 and columns 16-48 in the matrix by replacing the pixels by 0 in these rows and columns. The above image is the IMFFT of the resulting matrix. Figure 3 appears blurrier than Figure 2, because the compression algorithm used for Figure 3 removes more data from the original image matrix than the algorithm used in Figure 2 removes.

FIGURE 4. 24:40 FWAL Compression

The compression algorithm used in Figure 4 performs the MFWT on the grayscale matrix and then removes rows 24-40 and columns 24-40 in the matrix by replacing the pixels by 0 in these rows and columns. The above image is the MFWT of the resulting matrix.

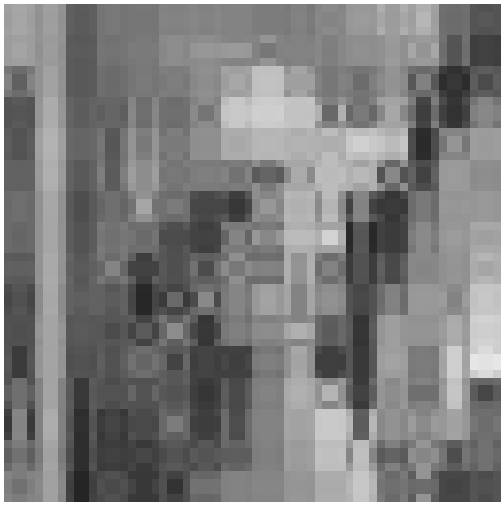


FIGURE 5. 16:48 FWAL Compression

The compression algorithm used in Figure 5 performs the MFWT on the grayscale matrix and then removes rows 16-48 and columns 16-48 in the matrix by replacing the pixels by 0 in these rows and columns. The above image is the IMFWT of the resulting matrix. Figure 5 appears blurrier than Figure 4, because the compression algorithm used for Figure 5 removes more data from the original image matrix than the algorithm used in Figure 4 removes.

9. Future Considerations

Although the Fourier transform and Walsh transform were sufficient for compressing the image used in this paper, the fast Haar transform requires even fewer arithmetic operations than both the FFT and FWT. Therefore, studying and programming a Haar transform and the Haar function is a worthwhile future project.

Using the same transforms discussed in this paper, analysis and compression of sound files using Matlab is another possibility for future research.

After taking graduate courses in measure theory, research on the continuous transforms and sampling theory involving these tranforms

could provide sufficient material for a presentation at a graduate seminar.

Although the JPEG2000 wavelet transform resembles the FWT, the wavelet functions used for this program more closely resemble the Haar functions. A project aiming at programming a compression algorithm more similar to the one used for JPEG2000 than the algorithms using the FFT and DWT would be even more helpful for understanding JPEG compression.

Appendix

9.1. **Matlab Codes.** The following Matlab functions were used for the calculations required for this paper:

DFT

```
function [output_args] = DFT( input_args )
output_args=1:length(input_args)
for j=1:length(input_args)
    output_args(j)=0
for k=1:length(input_args)
output_args(j)
=exp((-2*pi*i*(j-1)*(k-1)/length(input_args)))*input_args(k)+output_args(j)
end
end
```

Odds Function

```
function [ output_args ] = Odds( input_args )
indexes=2*[1:ceil(length(input_args)/2)]-1
output_args=input_args(indexes)
end
```

Evens Function

```
function [ output_args ] = Evens( input_args )
indexes=2*[1:floor(length(input_args)/2)]
output_args=input_args(indexes)
end
```

FFT

```
function [output_args] = FFT( input_args )
if length(input_args)==1
    output_args=input_args(1)
```

```
    return
end
evenout=FFT(Evens(input_args))
oddout=FFT(Odds(input_args))
for j=1:length(input_args) output_args(j)
=oddout((mod(j-1,length(input_args)/2)+1))
+exp(-2*(pi)*i*(j-1)/length(input_args))
*evenout((mod(j-1,length(input_args)/2)+1))
end
end
```

Normalized FFT

```
function [output_args] = NFFT( input_args )
output_args=1/length(input_args)*FFT(input_args)
end
```

IFFT

```
function [output_args] = FIT( input_args )
output_args=1/length(input_args)*conj(FFT(conj(input_args)))
end
```

Retain Function

```
function [output_args] = Ret( input1_args,input2_args )
newx=mod(input1_args,2^(input2_args+1))
if (newx >= 2^(input2_args)) output_args=1
else output_args=0
end
```

Walsh Function

```
function [output_args] = Wal( input1_args,input2_args )
output_args=1:length(input2_args)
for k=1:length(output_args)
output_args(k)=1
for j=0:ceil(log2(input1_args))
output_args(k)
=(-1)^(Ret(input1_args,j)*Ret(input2_args(k),-1-j))*output_args(k)
end
end
end
```

DWT

```
function [output_args] = Dwal( input_args )
output_args=1:length(input_args)
for j=1:length(input_args)
    output_args(j)=0
for k=1:length(input_args)
output_args(j)
=Wal((j-1),(k-1)/length(input_args))*(input_args(k))+output_args(j)
end
end
end
```

FWT

```
function [output_args] = Fwal( input_args )
if length(input_args)==1
    output_args=input_args(1)
    return
end
evenout=Fwal(Evens(input_args))
oddout=Fwal(Odds(input_args))
for j=1:length(input_args)
output_args(j)=
(oddout((mod(j-1,length(input_args)/2)+1))
+Wal(j-1,1/length(input_args))*evenout((mod(j-1,length(input_args)/2)+1)))
end
end
```

Normalized FWT

```
function [output_args] = NFwal( input_args )
output_args=1/length(input_args)^(1/2)*Fwal(input_args)
end
```

Two-Dimensional DFT

```
function [output_args] = MDFT( input_args )
output_args=input_args
for l=1:length(input_args(:,1))
for j=1:length(input_args(1,:))
    output_args(l,j)=0
for s=1:length(input_args(:,1))
for k=1:length(input_args(1,:))
output_args(l,j)
=exp((-2*pi*i*(j-1)*(k-1)/length(input_args(1,:))))
*exp((-2*pi*i*(l-1)*(s-1)/length(input_args(:,1))))
```

```
*input_args(s,k)+output_args(l,j)
end
end
end
end
```

MFFT

```
function [output_args] = MFFT( input_args )
output_args=input_args
for l=1:length(input_args(:,1))
   output_args(l,:)=FFT(input_args(l,:))
end
for j=1:length(input_args(1,:))
output_args(:,j)=FFT(output_args(:,j))
end
end
```

IMFFT

```
function [output_args] = IMFFT( input_args )
output_args=input_args
for l=1:length(input_args(:,1))
   output_args(l,:)=FIT(input_args(l,:))
end
for j=1:length(input_args(1,:))
output_args(:,j)=FIT(output_args(:,j))
end
end
```

Two-Dimensional DWT

```
function [output_args] = DMwal( input_args )
output_args=input_args
for l=1:length(input_args(:,1))
for j=1:length(input_args(1,:))
    output_args(l,j)=0
for s=1:length(input_args(:,1))
for k=1:length(input_args(1,:))
output_args(l,j)
=Wal((j-1),(k-1)/length(input_args))
*Wal((l-1),(s-1)/length(input_args))*input_args(s,k)+output_args(l,j)
end
end
end
```

```
end
```

FMWT

```
function [output_args] = FMwal( input_args )
output_args=input_args
for l=1:length(input_args(:,1))
   output_args(l,:)=NFwal(input_args(l,:))
end
for j=1:length(input_args(1,:))
output_args(:,j)=NFwal(output_args(:,j))
end
end
```

Triangle Cubic Spline Function

```
function [output_args] = Tri( input_args )
output_args=1:length(input_args)
for j=1:length(input_args)
newinput=mod(input_args(j),1)
if (newinput<(1/2))
output_args(j)=-24*(newinput)^2+32*(newinput)^3+1
else
output_args(j)=-24*(1-newinput)^2+32*(1-newinput)^3+1
end
end
end
```

Triangle Function

```
function [output_args] = Tri2( input_args )
output_args=1:length(input_args)
for j=1:length(input_args)
newinput=mod(input_args(j),1)
if (newinput<(1/2))
output_args(j)=-4*(newinput)+1
else
output_args(j)=-4*(1-newinput)+1
end
end
end
```

References

[1] Beauchamp, K.G. *Walsh Functions and Their Applications.* Academic Press, 1975.
[2] Chu, Eleanor and Alan George. *Inside the FFT Black Box.* CRC Press, 2000.
[3] Chu, Eleanor. *Discrete and Continuous Fourier Transforms.* CRC Press, 2008.
[4] Nievergelt, Yves. *Wavelets Made Easy.* Birkhauser Boston, c/o Springer-New York, New York, NY, 2001.