APPLICATIONS OF FOURIER ANALYSIS IN THE DIGITAL AGE

JACOB DILLES ♦ G00513892
OCTOBER 2011

ECE 421 – Technical Paper Writing Assignment – Final
(Word Count = 3995 satisfies 3500 requirement)

ABSTRACT

The author presents a brief overview of Fourier analysis, including its history and a review of four commonly used modern transformation pair notations. Several implementations found in digital signal processing applications are explored, including algorithmic considerations and common optimizations. The latest tools available for frequency-domain analysis on both high performance CSIC and real-time embedded systems are studied.

TABLE OF CONTENTS

HONOR CODE STATEMENT

On my honor, I have neither received nor given dishonorable aid on this report.

……………………………………………………

Jacob Dilles

Note: Some content in this report was prepared by the author in the Spring 2011 semester for Project 3 the ECE 320 class taught by Dr. Janos Gertler at George Mason University. It was nominated as a student writing sample during the ABET accreditation process for "demonstration of writing ability" and "mastery of course material."

A BRIEF OVERVIEW OF FOURIER ANALYSIS

INTRODUCTION

The early 1800's brought exciting progress in modern science, conveniently earmarked by the adoption of the original metric system prototypes and decimal multiples in 1799[1]. Dalton's publication in 1801 formed the basis for modern atomic theory. Volta discovered the wet electrochemical cell (1800), causing a flurry of experiments by Davy in electrochemistry (1806), Ørsted in electromagnetism (1819), Ampere in electrodynamics (1821), Seebeck in thermo-electricity (1821), Poisson in mathematical physics (circa 1823), culminating with Georg Ohm's now ubiquitous law of proportionality between voltage, current and resistance published in 1827 provided basis for the theoretical study of electricity. [1]

It was in this atmosphere of avid experimentation and scientific progress that the French mathematician and physicist Joseph Fourier was studying the flow of heat in a metal plate. Specifically he endeavored to solve the partial differential equation for which no general solution was known at the time:

$$\frac{\partial u}{dt} - \alpha\nabla^2 u = 0$$

where α is a constant and u is a function of three spatial variables and time. Experimentation had found specific solutions for simple heat sources: the trivial constant gradient case, and the case of sinusoidal oscillation. Fourier's idea was to model arbitrarily complex heat sources as a linear combination of superimposed monotonic sinusoids, and published an initial thesis in 1807. This linear combination is know today as the Fourier Series (**FS**). [2]

In a later memoir (1811) Fourier hypothesized that *any* function of a variable can be expanded in a series of integer multiple frequency sinusoids with linear coefficients expressed in terms of that variable. While his results were informal at best, and in the general case incorrect, his investigations began an entirely new field of mathematics which also bears his name (Fourier Analysis). Attempts to prove or disprove his theories have led to developments in the fields of convergence, function spaces, and abstract harmonic analysis.

Herein we present a cursory study of the family of domain transformations derived from Fourier's work, and examine their relevance, applications, and implementations in the modern age of digital computing.

1 Although technically, it was not until 1861 when a BAAS committee of prominent scientists (including Lord **Kelvin**, James Clerk ***Maxwell***, and John Prescott ***Joule***!) introduced the coherent MKS system which, was after being expanded to electrical units in 1873, was used almost exclusively by the scientific community for over a century (modern SI system was developed in the 1960s)

NOTATION

The symbolic convention used in this report was chosen for clarity and ease of comparison by inspection. In general, we expect the *continuous* signal $x(t)$ and the *discrete* signal $x[n]$ to be (potentially) complex valued functions of the real variable time, that is, specified in the *time domain*[2]. The notation $X(j\omega) = \text{CTFT}\{x(t)\}$ and $X(e^{j\omega}) = \text{DTFT}\{x[n]\}$ [3], are used to denote complex-valued functions of real variable ω which describe frequency spectrum calculated via the *Continuous Time Fourier Transform* (**CTFT**) and *Discrete Time Fourier Transform* (DTFT) of their respective time domain functions. The Fourier Series (**FS**) coefficients, often denoted as A_k, will be written here instead as $X_{(k)}$ and $X_{[k]}$ for the periodic cases of *continuous* (**FSC**) and *discrete* (**DFT**) time, respectively, to emphasize symmetry between the cases.

A BRIEF REVIEW OF FOURIER TRANSFORMATIONS

We begin by dividing the set of well-behaved, complex-valued functions of a real variable into the following categorizations[4]:

1. A periodic, continuous-time function $x(t)$ which has **FS** coefficients $X_{(k)}$
2. An aperiodic, continuous-time function $x(t)$ which has **CTFT** $X(j\omega)$
3. An aperiodic, discrete-time function $x[n]$ which has **DTFT** $X(e^{j\omega})$
4. A periodic, discrete-time function $x[n]$ which has **DFT** $X_{[k]}$

In the sections that follow, we will examine these cases individually, compare their symmetries, and discuss their applications in the field of signal processing.

2 This gives a more tangible definition to the already abstract concept of a "frequency spectrum." One should note that these signals could just as easily be a complex-valued function of any other real variable(s), for example, position in the case when calculating the far-field diffraction pattern of an (EM) aperture.

3 The notation of X(jω) for the CTFT is chosen primarily to emphasis that ω is a real value denoting the magnitude of a complex frequency. The notation of X(exp(jω)) helps to distinguish between the CTFT and DTFT, and draws attention to the periodic nature of a sampled function. Notation aside, both are only functions of the variable ω, and for the purposes of substitution (as in f(a) = 2*a → f(2*a) = 2*(2*a)) the other arguments are ignored.

4 The study of ill-behaved functions, convergence issues, and other such pesky nuances is another fascinating subject, but unfortunately beyond the scope of this report.

CASE 1 – PERIODIC CONTINUOUS FUNCTIONS

Even after two hundred years of refinement, Fourier would still recognize his work in this area. With respect to a continuous-time well-behaved[5] unbounded *periodic* function $x(t)$ with period $T = \frac{2\pi}{\omega_0}$, we may write the Fourier Series (**FS**) coefficients $X_{(k)}$ as:

$$X_{(k)} = \frac{1}{T}\int_T x(t)e^{-j\omega_0 kt}dt \qquad (1.1)$$

where $k \in \mathbb{Z}$. Thus an unbounded *continuous* periodic signal in the time domain has an unbounded *discrete* frequency spectrum, the components of which define the inverse transform, called the **FS** expansion, that may be used to reconstruct the original function:

$$\hat{x}(t) = \sum_{k=-\infty}^{\infty} X_{(k)}e^{j\omega_0 kt} \qquad (1.2)$$

where $x(t) \equiv \hat{x}(t)$ [3]. The implication of this result is that a periodic signal of arbitrary complexity may be decomposed into the superposition of simple sinusoidal frequency components. In other words, the frequency spectrum of a time-periodic function is a discrete series of harmonics in the frequency domain.

For many functions of interest, a subset of **FS** coefficients go to zero, and this relationship has facilitated many advancements in the fields of mathematics. It is, however, quite difficult to apply algorithmically, as would be necessitated by system in the real world, due to the infinite number of coefficients required for reconstructing a signal. It is worth noting that the approximation:

$$\tilde{x}(t) = \sum_{k=-N}^{N} X_{(k)}e^{j\omega_0 kt} \quad \Rightarrow \quad \hat{x}(t) = \lim_{N\to\infty}\tilde{x}(t) \qquad (1.3)$$

can, in some cases, converge quickly; and in all cases produces a bounded, calculable error. Shortly, we shall study the relationship between this approximation and the effects of *windowing* the non-periodic signal $x(t)$ before taking its Fourier transform.

5 See note 4 under "Notation". Well-behaved periodic functions are continuously differentiable, with *absolutely summable* **FS** coefficients.

CASE 2 - APERIODIC CONTINUOUS FUNCTIONS

A continuous-time unbounded *aperiodic* function $x(t)$ has a continuous unbounded frequency spectrum $X(j\omega)$ obtained via the Continuous Time Fourier Transform (**CTFT**). Conceptually, the **CTFT** may be thought of the limit of (1.1) in the case where the period $T \to \infty$ [4]. In this case, the fundamental frequency $\omega_0 \to 0$, and the frequency spectrum becomes a continuous function of an (arbitrary) variable we call $\omega \in \Re$ (in contrast with $k \in \mathrm{Z}$):

$$X(j\omega) = \int_{-\infty}^{\infty} x(t) e^{-j\omega t} dt \qquad (1.4)$$

In the same way the $X_{(k)}$ may be used to reconstruct a periodic $x(t)$, the spectrum $X(j\omega)$ obtained from a **CTFT** may be used to exactly reconstruct the original aperiodic $x(t)$:

$$x(t) = \int_{-\infty}^{\infty} X(j\omega) e^{j\omega t} d\omega \qquad (1.5)$$

We may recognize (1.4) and (1.5) as the "classic" Fourier Transform-Inverse pair. This beautifully symmetric relationship also has many applications in pure mathematics, but again is not particularly useful in the computational arts due to the difficulty in creating a symbolic function from sampled data and in the solution of definite integrals over infinite time.

CASE 3 – APERIODIC DISCRETE FUNCTIONS

When the continuous unbounded aperiodic function $x(t)$ is *sampled* at intervals T such that the sampling frequency $f_s = \frac{1}{T}$ in Hertz, (or $\omega_s = 2\pi f_s = \frac{2\pi}{T}$ in radians per second) values of $x(t)$ modulate the Dirac comb:

$$x[n] = \sum_{n=-\infty}^{\infty} x(nT)\delta(t - nT) \qquad (1.6)$$

Taking the **FT** of (1.6), we can find the frequency spectrum $X(e^{j\omega})$ in units of $2\pi f / f_s$ Hertz:

$$X(e^{j\omega}) = \sum_{n=-\infty}^{\infty} x(nT) e^{-j\omega nT} \qquad (1.7)$$

Returning to (1.6), any particular value of $x\left[n_0\right]$ is equal to $x\left(n_0 T\right)$, we may substitute into (1.7), from which the **DTFT** is defined. This represents the transform of the sampled signal into the frequency domain[5]:

$$X\left(e^{j\omega}\right)=\sum_{n=-\infty}^{\infty} x\left[n\right]e^{-j\omega n} \qquad (1.8)$$

It is crucial to note that (1.7) is periodic with period T, and that the frequency spectrum of $X\left(e^{j\omega}\right)$ is continuous with $\omega \in \Re$. The inverse **DTFT** is defined:

$$x\left[n\right]=\frac{1}{2\pi}\int_{2\pi} X\left(e^{j\omega}\right)e^{j\omega n}d\omega \qquad (1.9)$$

To the astute reader, equation (1.9) will look strikingly similar to (1.1) in the case where $\omega_0 = 1$. In this sense, the samples of a continuous signal are in fact the $X_{(k)}$ coefficients of a **FS** expansion of their own frequency spectrum (normalized to 2π).

There is similar duality between (1.8) and (1.2), from which one may casually (though correctly) conclude that the frequency spectrum of a sampled signal repeats every ω_s. Although the input signal is now slightly more compatible with computational methods, the spectrum $X\left(e^{j\omega}\right)$ is unbounded and continuous[6], and overlap of the repeated signals (as well as the need to process an infinite number of samples) prohibits practical usage of the **DTFT** in signal processing.

CASE 4 – PERIODIC DISCRETE FUNCTIONS

When the discrete-time signal $x[n]$ is itself periodic with period N, is of finite duration of length N, or is *windowed* over length N[7], it is possible to completely describe the frequency spectrum $X_{[k]}$ with *only* N discrete frequency components (i.e. $x[n]$ and $X[k]$ have the same number of elements) using the **DFT**:

$$X_{[k]}=\sum_{n=0}^{N-1} x_n e^{-j\frac{2\pi}{N}kn} \qquad (1.10)$$

This is remarkable – a complete, finite frequency spectrum may be computed! Coupled with the inverse transform:

6 If x[n] is unbounded, but zero everywhere except for a contiguous portion of *finite duration*, it reduces to the **DFT**

7 These cases are indistinguishable after the **DFT**

$$x[n] = \frac{1}{N}\sum_{k=0}^{N-1} X_{[k]} e^{j\frac{2\pi}{N}kn} \tag{1.11}$$

It is physically possible to construct a machine which can transform a finite duration, discrete input signal exactly[8] into the frequency domain with a finite amount of memory, manipulate that data in a finite (real) time, then transform it back into the time-domain in a format comparable with the input data. Further, the work of Nyquist and Shannon has shown that, so long as the highest frequency component $\omega_{\text{MAX}}\left(x(t)\right)$ of the signal $x(t)$ sampled as $x[n]$ is strictly less than $2\omega_s$, there is sufficient information in the frequency spectrum $X_{[k]}$ to reconstruct the original signal $x(t)$ exactly [3] [4]. In practice, this allows the realization of filters, control systems, compression algorithms, and real-time spectral analysis in both software and hardware digital electronic systems.

SUMMERY OF THE FOURIER FAMILY OF TRANSFORMATION EQUATIONS

The following table aims to highlight the symmetries between the cases we have discussed, the duality of the transform-inverse pairs, and the general elegance of Fourier analysis:

Time Domain Characterization	**Time → Frequency**	**Frequency → Time**
(1) Aperiodic Continuous and Unbounded	$X(j\omega) = \int_{-\infty}^{\infty} x(t)e^{-j\omega t}dt$	$x(t) = \int_{-\infty}^{\infty} X(j\omega)e^{j\omega t}d\omega$
(2) Periodic or Bounded	$X_{(k)} = \frac{1}{T}\int_{T} x(t)e^{-j\omega_0 kt}dt$	$x(t) = \sum_{k=-\infty}^{\infty} X_{(k)}e^{j\omega_0 kt}$
(3) Unbounded and Aperiodic	$X\left(e^{j\omega}\right) = \sum_{n=-\infty}^{\infty} x[n]e^{-j\omega n}$	$x[n] = \frac{1}{2\pi}\int_{2\pi} X\left(e^{j\omega}\right)e^{j\omega n}d\omega$
(4) Bounded or periodic	$X_{[k]} = \sum_{n=0}^{N-1} x_n e^{-j\frac{2\pi}{N}kn}$	$x[n] = \frac{1}{N}\sum_{k=0}^{N-1} X_{[k]}e^{j\frac{2\pi}{N}kn}$

8 Exact in the ideally-sampled, symbolically manipulated sense. Obviously compromises to exactness are made in pursuit of practicality

The remainder of this report will examine modern applications of case (4), the **DFT/IDFT**, which is the only pair with bounded summation/integration in both transformations.

APPLICATIONS OF THE DFT IN THE DIGITAL AGE

Illustration 1: The "spectrum analyzer" feature, common in digital media player software, uses the DFT to compute frequency bin magnitudes.

There are a vast array of applications which use the DFT, some of which include:

- Control and automation: digital proportional-integral-differential (PID) control, nondestructive testing, manufacturing process monitoring
- Multimedia compression algorithms: MP3 audio, JPEG images, MPEG video
- Defense industry: phased-array radar, jam-resistant communication channels, broadband spectral imagery analysis
- Communications: digital filtering, direct conversion transceivers, frequency domain multiplexing (FDM) coding
- Medicine: ultrasound (acoustic phased array), magnetic resonance imaging (MRI), optical coherence tomography (OCT)
- Experimental science: physics; astronomic, quantum, high energy, condensed matter, nuclear

There are many, many more, with applications ranging from the everyday to the top secret. The majority of applications which use the DFT, including those listed above, require a fast algorithm for computation. A brief overview of the standard implementation methods are given here. We shall examine a few examples below, in illustrating how the DFT is implemented algorithmically.

IMPLEMENTATION OF THE DFT

The DFT (1.10) is a algebraic formula which a digital computer can not directly understand. Before we may employ such a device to evaluate this formula (or any other), it must be translated into a sequence of machine-language operations that instruct a processor to manipulate data. This task is often addressed by writing a "high-level" program, and letting a compiler take care of the rest. In order to effectively describe the implications of this, the following section assumes the reader has basic knowledge of computer programming (examples are given in MATLAB syntax) and computational time complexity.

THE STRAIGHTFORWARD CALCULATION

The standard notation for the computational-oriented DFT is:

$$X_{[k]} = \sum_{n=0}^{N-1} x_n \omega_N^{kn} \qquad (2.1)$$

with $0 \le k < N$. Contrasted with (1.10), there are two differences. First, the term $\omega_N = e^{-j\frac{2\pi}{N}}$ is pre-computed before the summation and included as a constant to eliminate redundant calculations. Second, the iterative variables k and n are exponents to the term ω_N, rather than included inside. This is done because multiplication of integer values, and raising of a number to an integer power (especially a power of two), is far less expensive than carrying out the equivalent operations in floating point, as would be required in the original formulation.

The most straightforward way compute the (2.1) algorithmically would be to compute the summation over N elements for each k. In MATLAB, one could write:

```
function [ xk ] = slowfft( xn )
nfft = length(xn);
wn = (-2j * pi/N);
xk = xn*0;
for k = 1:nfft
    for n = 1:nfft
        xk(n) = xk(n) + x(n) *exp(wn * k * n );
    end
end
```

Which clearly runs in $O(NN) = O(N^2)$ time. For most of the applications outlined above, quadratic time performance is either undesirable or simply inadequate. Fortunately, a better algorithm exists.

THE COOLEY AND TUKEY FAST FOURIER TRANSFORM

The Fast Fourier Transform (FFT) algorithm, introduced by James Cooley and John Tukey in 1965[9], can compute the same result for (2.1) in $O(N\log_2 N)$ operations using no more memory than the trivial solution

9 Technically, Gauss had used this method for computing the power series as early as 1805, however the T&C article revived the technique and is considered responsible for its widespread use in modern computing.

[6]. It works because many of the $x_n e^{-j\frac{2\pi}{N}kn}$ terms are used repeatedly. The strategy is to factor N into $N = N_1 N_2$, in which equation (2.1) can be re-written

$$X_{[k_1+k_2N_1]} = \left(\sum_{n_2=0}^{N_2-1} \left(\omega_{N_1}^{n_2k_1}\right) \left[\sum_{nL1}^{N_1-1} \left(\omega_{N_1}^{n_1k_1}\right) \left(x_{[n_1N_2+n_2]}\right) \right] \right) \left(\omega_{N_2}^{n_2k_2}\right) \qquad (2.2)$$

The algorithm computes a DFT over N_1 elements (the inner sum), multiplies that by a $\omega_N^{n_2k_1}$ term (the *twiddle factor*), and finally computes a DFT over N_2 elements (the outer sum), usually the smaller of the two, known as the *butterfly*. This process is continued in a recursive[10] manner. The N_2 factor is often fixed, common values being 2, 3, 5, and 7, and is referred to as the *radix* of the algorithm. [7]

While it would be within the realm of a second year CS assignment to implement (2.2), such an exercise has no practical application since highly optimized FFT libraries are freely available.

THE FFTW LIBRARY

FFTW, an acronym for the "Fastest Fourier Transform in the West", is a freely available, open source FFT library[11] which is so widely used that it has become the *de facto* implementation of choice – even the MATLAB `fft()` function is backed by FFTW. Written in ANSI C, it is highly portable and can compile on any system with a c99 compiler (including many embedded systems). It is also highly optimized, using "dynamic programing" to heuristically select the fastest algorithm variant, or *codelet*, for the current platform, data type, and N. It automatically enables processor dependent special instructions (eg. SSE and 3DNOW!), as well as hand-coded assembly for popular architectures. The code to invoke FFTW in MATLAB is quite trivial:

```
xk = fft(xn);
xn = ifft(xk);
```

The syntax for the native C interface is also nearly as terse. For all but the most demanding real-time applications, the FFTW library can provide sufficient performance with arbitrary length DFTs. There are exceptions, however, in the case of large data sets (high f_s), high numerical precision and high repetition rates (transforms/second) required for realtime frequency domain digital signal processing when something faster is required.

10 It is often implemented procedurally for optimization of fixed sizes of N, but it is still logically recursive.
11 See http://www.fftw.org/

THE NVIDIA CUFFT LIBRARY

Rendering, the process of converting vector shapes to the discrete pixels required for display on a computer monitor, was traditionally performed by application code running on a computer's CPU. In the early 1990's, the consumer expectation of rendering performance was pushing the bounds of current CPU performance. The emerging computer gaming market drove manufactures of video adapters, which at the time were simple screen buffers with line-scanning D/A circuitry, to add “Graphics Acceleration” features to their cards.

The early models - humble FPGAs which could only draw lines and shade rectangles – were soon replaced by more powerful graphics processing unit, or GPU. These GPUs were extremely specialized, on custom silicon, developed specifically for chore of rapidly performing vector coordinate transformations on large numbers of wireframe polygons (specified in 3 dimensional coordinates by the CPU), painting each polygon with a set of colors textures (requiring fast local memory access), and calculating effects like transparency, particulates (smoke, fog), flames, and water specular, within the “scene,” and rendering everything to a screen buffer to be displayed to the user.

The modern GPU must render hundreds of thousands of polygons into color values at millions of pixels per frame at least 24 frames per second. The only way to achieve such performance is with parallel processing. As such, GPUs are equipped with dozens, if not hundreds[12] of identical cores. These cores are not nearly as complete as a CISC CPU, but rather are specialized ALUs with hardware floating point vector functions (such as four-cycle multiply-accumulate and square root). The architecture is also unique in that the cores share a program counter, so each clock cycle all cores perform the same operation on a different memory address. The GPU is so well suited to the task that it might as well have been designed from the start for computing the FFT.

Recently (2007) the graphics card manufacturer NVIDIA released an open API called CUDA[13] to allow developers to compile and execute arbitrary code on their GPUs. With the initial release, they included a rough FFT implementation in the library CUFFT. The library has been continuously refined, and as of the fourth version (February 2011) it supports 1, 2, and 3 dimensional real and complex Fourier transforms of up to 128 million elements in double floating point (64 bit) precision.

Using the library is (almost) as simple as FFTW, and the API even includes a “FFTW compatible mode” where FFTW calls in existing code are executed on the GPU with only changes to header files. NVIDIA also offers a MATLAB plugin[14], which reportedly offers 1400% speedups in two dimensional transforms for an average GPU.

12 At the top of the chart as of publication is the Tesla M2070, with 448 cores and 6GB of ECC memory.
13 Quick overview located at http://www.nvidia.com/object/what_is_cuda_new.html
14 See http://developer.stage.nvidia.com/matlab-cuda

EMBEDDED DSP

It would be inappropriate to end a report on the DFT without touching on the wide range of embedded systems using digital signal processing (DSP) chipsets to run FFT algorithms. Embedded systems are computers that are integrated with other electrical or mechanical parts, designed to perform a specific function with real-time computing constraints. In contrast to the general purpose personal computer for which the methods we have discussed so far apply, embedded systems are typically designed to be low cost (1000 times less than a PC) and low power consumption (100,000 times less than a PC). The overwhelming majority are RSIC with very limited computing resources. When an embedded processor requires DSP functionality, a module with dedicated hardware is often designed specifically for the task, to be included on-die or in a separate package.

Among the simplest examples of an embedded DSP ASIC dedicated to performing (inverse) DFT[15]s is the humble MP3 decoder. Take the NXP UDA1380[16], which retails for about $1 USD[17]. During operation, it calculates a 24-bit $x[t]$ value from a block of N=1152 at a rate of 48kHz, while consuming only 24mW of power – a feat which would be nearly impossible to achieve in *any* general purpose ALU.

A more complex example is the "wifi" (IEEE 802.11A/G/N) transceiver chipset, which employs an orthogonal frequency division multiplexing (OFDM) scheme. A stream of data, or *symbols*, at the transmitting end are mapped into groups called *constellations* of four symbols each. When enough data has been buffered (64 constellations), each constellation is passed as an $X_{[k]}$ value to an IFFT which produces 64 values for $x[t]$. The real and imaginary components of $x[t]$ are modulated 90° out of phase (QAM) onto a carrier frequency, which is then amplified and sent off to an antenna. At the receiving end, the real and imaginary $x[t]$ components are demodulated off the carrier, and a FFT reconstructs the original $X_{[k]}$ constellations and their constituent symbols. The OFDM passband transmission has nearly flat distribution of frequency components (almost resembling white noise), offering strong immunity to severe channel conditions and narrow band interference, while causing very little interference itself. With high-linearity amplifiers, can achieve extremely high link spectral efficiencies. [8]

Aside from ASICs, there are many varieties of general purpose embedded processors with FFT capabilities. One of the first manufacturers to include an FFT block was Hyperstone with their E2 series in 1996. Other manufactures followed, and now DSP on-die coprocessors are available on MCUs from Microchip, TI, Atmel, and Freescale. There are also free and commercial DFT component-blocks available for hardware synthesis in FPGA designs.

15 Technically, MP3 uses the modified discrete cosine transform (MDCT), which is the result of limiting the DFT to real numbers and overlapping sample windows. The MDCT produces N/2 frequency values for N time values, however the algorithms are otherwise very similar, and mathematically the DCT is a subset of the DFT.
16 http://www.nxp.com/documents/data_sheet/UDA1380.pdf
17 Digikey.com part number UDA1380HN/N2,118-ND

CONCLUSION

We have studied the history behind, and the four modern exemplifications of, the Fourier transform and it's inverse, noting symmetries and dualities, and provided conceptual contexts for each case. We have demonstrated how only one pair, the Discrete Fourier Transform and it's inverse, is suitable for realization in digital signal processing applications, and listed a few cases where it has been employed. We have examined the algorithmic considerations for implementing the DFT in real systems, and explored the most common optimization, the Cooley and Tukey Fast Fourier Transform. We presented the latest tools available for frequency domain analysis on the personal computer, and touched on applications in embedded systems. In conclusion the DFT is a powerful tool, which allows digital computers to efficiently process time domain signals in the frequency domain. ◊ JSD

BIBLIOGRAPHY

[1] Maver, William Jr: "Electricity, it's History and Progress", The Encyclopedia Americana, 1978.
[2] Georgiĭ Pavlovich Tolstov: "Fourier series", Dover , 1962.
[3] Oppenheim, A.V., and R.W. Schafer: " Discrete-Time Signal Processing", Prentice-Hall, 1989.
[4] Stein, Elias; Shakarchi, Rami: "Fourier Analysis: An introduction", Princeton University Press, 2003.
[5] http://nptel.iitm.ac.in/courses/Webcourse-contents/IIT-KANPUR/Digi_Sign_Pro/pdf/final-chap-5.pdf
[6] Cooley, James W.;Tukey, John W., "An algorithm for the machine calculation of complex Fourier series", Math. Comp. v. 19, 1965
[7] Matteo Frigo and Steven G. Johnson, "The Design and Implementation of FFTW3", Proc. IEEE v. 93, no. 2, 2005
[8] http://www.digitaltvbooks.com/cofdm.pdf