# PRACTICAL ORIENTED ANALYSIS ON THE SIGNAL PROCESSING USING FFT ALGORITHM

*Prof. K. Phani Srinivas and Dr. P. S. Aithal*

## ABSTRACT

*The idea of the research is mainly to understand the application of DFT, windowing, zero padding, use the thereafter knowledge to calculate spectrum of the whole .wav file from which the fundamental frequencies and corresponding harmonics are grouped together. The spectrum of the whole file is calculated using "FFT". From these frequencies one can find out the note played in music. Discrete Fourier Transform (DFT) is one of the specific forms of Fourier analysis. The DFT requires an input function that is discrete and whose non-zero values have a finite duration. In particular, the DFT is widely employed in signal processing and related fields to analyze the frequencies contained in a sampled signal, to solve partial differential equations, and to perform other operations such as convolutions. The DFT can be computed efficiently in practice using Fast Fourier Transform (FFT) algorithms.*

Keywords- FFT, DFT, WAV file, Signal Processing

***Reference to this paper should be made as follows:***

***Prof. K. Phani Srinivas and Dr. P. S. Aithal. '****PRACTICAL ORIENTED ANALYSIS ON THE SIGNAL PROCESSING USING FFT ALGORITHM, International Journal of Electronics Engineering and Applications, Volume 8, Issue II, July-Dec. 2020, pp 01-10.*

***Biographical notes:***

***K.PHANI SRINIVAS*** *working as a Director for the Research and Development and He Had Five Years of Industrial Experience as a team Leader in the research areas of Embedded Systems and Tele-Communications and also He is Having 13 Years of Experience in Academics, Research and Administrative reports. He received several research awards like Best Engineer Award, Best Teacher Award and Best Research Paper Award. Also He is acting as an Editor/Reviewer for so many top international Journals.*

***Prof. Dr. P. Sreeramana Aithal*** *has 29 years experience in Teaching & Research and 18 years experience in Administration.Dr. P. S. Aithal has secured theFIRST RANKin TOP 12,000 Business Management Authors in the Global Ranking of Elsevier'sSSRN (USA) for maximum number of Research papers publications during 2017 &2018. He hasworked as Principal at Srinivas Institute of Management Studies*

Published by: International Associations of Professionals and Technical Teachers (IAPATT)

## I. INTRODUCTION

Digital source data typically consists of a linear array of 'samples' of some signal collected at uniformly spaced time intervals. Examples include (uncompressed) audio such as .WAV files, and digitized radio frequency signals. Digital image processing is the whole branch of DSP which deals with two-dimensional (image) datasets, but is not something I intend to dwell on here.

A large part of Digital Signal Processing (DSP) is concerned with frequency domain processing; this page introduces basic fourier techniques, concepts of signals, modulation and side bands, and will demonstrate the methods used for detection and filtering of specific frequency signals in a dataset (such as DTMF tones in an audio file).

Multiplying by a windowing function suppresses glitches and so avoids the broadening of the frequency spectrum caused by glitches. Any waveform may be considered as the sum a set of components, which are each sinusoidal waveforms. A fourier analysis will decompose any sound into a fixed set of such components. However, real sounds, particularly those produced by resonating bodies, such as musical instruments, vocal chords, and so forth, tend to show prominent peaks in the power spectrum including that these sources produce certain components predominantly. These components often form harmonic series.

The discrete Fourier transform is the Fourier transform of a digital signal. A digital signal has a fixed resolution (the sampling period) and a limited extent on the time axis (duration). As a result the output frequency spectrum also has a limited frequency resolution (caused by the limited duration of the input) and limited frequency range caused by the limited temporal resolution of the input. The frequency resolution is the reciprocal of the input duration and the frequency range is half the sampling frequency. Thus, frequency spectra produced by the dft have just over half as many values (N/2+1) as the input waveform. The extra value is for the amplitude at zero frequency and its (arbitrary) phase. This value represents the DC-offset of the waveform combined with its overall power. The discrete Fourier transform may be executed with less computation by using a more efficient algorithm called the Fast Fourier Transform.

## 2. DISCRETE FOURIER TRANSFORM (DFT):

The Discrete Fourier Transform (DFT) is one of the most important tools in Digital Signal Processing. First, the DFT can calculate a signal's frequency spectrum. This is a direct examination of information encoded in the frequency, phase, and amplitude of the component sinusoids. For example, human speech and hearing use signals with this type of encoding. Second, the DFT can find a system's frequency response from the system's impulse response, and vice versa. This allows systems to be analyzed in the frequency domain, just as convolution allows systems to be analyzed in the time domain. Third, the DFT can be used as an intermediate step in more elaborate signal processing techniques. The classic example of this is FFT convolution, an algorithm for convolving signals that is hundreds of times faster than conventional methods.

The sequence of N complex numbers$x_0, ..., x_{N-1}$is transformed into the sequence of N complex numbers $X_0, ..., X_{N-1}$by the DFT according to the formula:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N} kn} \qquad k = 0, \ldots, N-1$$

The inverse discrete Fourier transform (IDFT) is given by [2]:

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k e^{\frac{2\pi i}{N} kn} \qquad n = 0, \ldots, N-1.$$

1. The normalization factor multiplying the DFT and IDFT (here1 and 1/*N*) and the signs of the exponents are merely conventions.
2. A normalization of for both the DFT and IDFT makes the transforms unitary, which has some theoretical advantages.
3. The convention of a negative sign in the exponent is often convenient because it means that $X_k$ is the amplitude of a "positive frequency" $2\pi k/ N$. Equivalently, the DFT is often thought of as a matched filter: when looking for a frequency of +1, one correlates the incoming signal with a frequency of −1.

Since the DTFT is also a continuous Fourier transform (of a comb function), the Fourier series also applies to it. Thus, when *s*[*n*] is periodic, with period **N**, $S_T(f)$ is another Dirac comb function, modulated by the coefficients of a Fourier series. And the integral formula for the coefficients simplifies to**:**

$$S[k] = \sum_{n=0}^{N-1} s[n] \cdot e^{-i2\pi \frac{k}{N} n}$$ for all integer values of *k*.

Since the DTFT is periodic, so is S[k]. And it has the same period (N) as the input function. This transform is also called DFT, particularly when only one period of the output sequence is computed from one period of the input sequence.

When s[n] is not periodic, but its non-zero portion has finite duration (N), ST($f$) is continuous and finite-valued. But a discrete subset of its values is sufficient to reconstruct/represent the (finite) portion of s[n] that was analyzed. The same discrete set is obtained by treating N as if it is the period of a periodic function and computing the Fourier series coefficients / DFT. The inverse transform of S[k] does not produce the finite-length sequence, s[n], when evaluated for all values of n. (It takes the inverse of ST($f$) to do that.) The inverse DFT can only reproduce the entire timedomain if the input happens to be periodic (forever). Therefore it is often said that the DFT is a transform for Fourier analysis of finite-domain, discrete-time functions. An alternative viewpoint is that the periodicity is the time-domain consequence of approximating the continuous-domain function, ST($f$), with the discrete subset, S[k]. N can be larger than the actual non-zero portion of s[n]. The larger it is, the better the approximation (also known as zero-padding).

The DFT can be computed using a fast Fourier transform (FFT) algorithm, which makes it a practical and important transformation on computers. See Discrete Fourier transform for much more information, including:

- the inverse transform
- transform properties
- applications
- tabulated transforms of specific functions

Applications of DFT:
The DFT has seen wide usage across a large number of fields:

Spectral analysis,

Data compression,

Partial differential equations,

Multiplication of large integers,

Outline of DFT polynomial multiplication algorithm.

## 3. FAST FOURIER TRANSFORM:

A fast Fourier transform (FFT) is an efficient algorithm to compute the discrete Fourier transform (DFT) and it's inverse. There are many distinct FFT algorithms involving a wide range of mathematics, from simple complex-number arithmetic to group theory and number theory; this article gives an overview of the available techniques and some of their general properties, while the specific algorithms are described in subsidiary articles linked below. A DFT decomposes a sequence of values into components of different frequencies. This operation is useful in many fields (see discrete Fourier transform for properties and applications of the transform) but computing it directly from the definition is often too slow to be practical. An FFT is a way to compute the same result more quickly: computing a DFT of N points in the naive way, using the definition, takes O(N2) arithmetical operations, while an FFT can compute the same result in only O(N log N) operations. The difference in speed can be substantial, especially for long data sets where N may be in the thousands or millions—in practice, the computation time can be reduced by several orders of magnitude in such cases, and the improvement is roughly proportional to N/log (N). This huge improvement made many DFT-based algorithms practical; FFTs are of great importance to a wide variety of applications, from digital signal processing and solving partial differential equations to algorithms for quick multiplication of large integers.

The most well known FFT algorithms depend upon the factorization of N, but (contrary to popular misconception) there are FFTs with O (N log N) complexity for all N, even for prime N. Many FFT algorithms only depend on the fact that  is an Nth primitive root of unity, and thus can be applied to analogous transforms over any finite field, such as number-theoretic transforms. Since the inverse DFT is the same as the DFT, but with the opposite sign in the exponent and a 1/N factor, any FFT algorithm can easily be adapted for it.

An FFT computes the DFT and produces exactly the same result as evaluating the DFT definition directly; the only difference is that an FFT is much faster. (In the presence of round-off error, many

FFT algorithms are also much more accurate than evaluating the DFT definition directly, as discussed below.)
Let x0, ...., xN-1 be complex numbers. The DFT is defined by the formula

$$X_k = \sum_{n=0}^{N-1} x_n e^{-i 2\pi k \frac{n}{N}} \qquad k = 0, \ldots, N-1.$$

Evaluating this definition directly requires $O(N^2)$ operations: there are $N$ outputs $X_k$, and each output requires a sum of $N$ terms. An FFT is any method to compute the same results in O($N$ log $N$) operations. More precisely, all known FFT algorithms require $\Theta(N \log N)$ operations (technically, $O$ only denotes an upper bound), although there is no proof that better complexity is impossible.

To illustrate the savings of an FFT, consider the count of complex multiplications and additions. Evaluating the DFT's sums directly involves $N^2$ complex multiplications and $N(N-1)$ complex additions [of which $O(N)$ operations can be saved by eliminating trivial operations such as multiplications by 1]. The well-known radix-2 Cooley–Tukey algorithm, for $N$ a power of 2, can compute the same result with only $(N/2)\log_2 N$ complex multiplies (again, ignoring simplifications of multiplications by 1 and similar) and $N \log_2 N$ complex additions. In practice, actual performance on modern computers is usually dominated by factors other than arithmetic and is a complicated subject (see, e.g., Frigo & Johnson, 2005)

## 4.1. WHY A DFT IS USUALLY CALLED AN FFT IN PRACTICE?

Practical implementations of the DFT are usually based on one of the Cooley-Tukey ``Fast Fourier Transform" (FFT) algorithms For this reason, the matlab DFT function is called `fft', and the actual algorithm used depends primarily on the transform length .8.2 The fastest FFT algorithms generally occur when is a power of 2. In practical audio signal processing, we routinely zero-pad our FFT input buffers to the next power of 2 in length (thereby interpolating our spectra somewhat) in order to enjoy the power-of-2 speed advantage. Finer spectral sampling is a typically welcome side benefit of increasing to the next power of 2. Appendix A provides a short overview of some of the better known FFT algorithms, and some pointers to literature and online resources

## 4.2. FFT ALGORITHMS:

Cooley–Tukey algorithm

Prime-factor FFT algorithm,

Bruun's FFT algorithm, Rader's

FFT algorithm,

Bluestein’s FFT algorithm**.**

## 4.3. COOLEY–TUKEY ALGORITHM:

By far the most common FFT is the Cooley–Tukey algorithm. This is a divide and conquer algorithm that recursively breaks down a DFT of any composite size $N = N_1N_2$ into many smaller DFTs of sizes $N_1$ and $N_2$, along with O($N$) multiplications by complex roots of unity traditionally called twiddle factors (after Gentleman and Sande).

This method (and the general idea of an FFT) was popularized by a publication of J. W. Cooley and J. W. Tukey, but it was later discovered (Heideman & Burrus) that those two authors had independently re-invented an algorithm known to Carl Friedrich Gauss (and subsequently rediscovered several times in limited forms).

The most well-known use of the Cooley–Tukey algorithm is to divide the transform into two pieces of size $N / 2$ at each step, and is therefore limited to power-of-two sizes, but any factorization can be used in general (as was known to both Gauss and Cooley/Tukey). These are called the **radix2** and **mixed-radix** cases, respectively (and other variants such as the split-radix FFT have their own names as well). Although the basic idea is recursive, most traditional implementations rearrange the algorithm to avoid explicit recursion. Also, because the Cooley–Tukey algorithm breaks the DFT into smaller DFTs, it can be combined arbitrarily with any other algorithm for the DFT. More generally, Cooley–Tukey algorithms recursively re-express a DFT of a composite size $N = N_1N_2$ as:

1. Perform $N_1$ DFTs of size $N_2$.
2. Multiply by complex roots of unity called twiddle factors.
3. Perform $N_2$ DFTs of size $N_1$.

Typically, either $N_1$ or $N_2$ is a small factor (not necessarily prime), called the radix (which can differ between stages of the recursion). If $N_1$ is the radix, it is called decimation in time (DIT) algorithm, whereas if $N_2$ is the radix, it is decimation in frequency (DIF, also called the SandeTukey algorithm).

## 4.4. The radix-2 DIT case:

A Radix-2 decimation-in-time (DIT) FFT is the simplest and most common form of the Cooley–Tukey algorithm, although highly optimized Cooley–Tukey implementations typically use other forms of the algorithm as described below. Radix-2 DIT divides a DFT of size N into two interleaved DFTs (hence the name "radix-2") of size N/2 with each recursive stage.

The discrete Fourier transform (DFT) is defined by the formula:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N} nk},$$

where k is an integer ranging from 0 to N − 1.

$x_{2m}$ $x_0, x_2, \ldots, x_{N-2}$

$x_{2m+1}$ $x_1, x_3, \ldots, x_{N-1}$

Radix-2 DIT first computes the DFTs of the even-indexed inputs ( ) and of the odd-indexed inputs ( ), and then combines those two results to produce the DFT of the whole sequence. This idea can then be performed recursively to reduce the overall runtime to O(N log N). This simplified form assumes that N is a power of two; since the number of sample points N can usually be chosen freely by the application, this is often not an important restriction.

The Radix-2 DIT algorithm rearranges the DFT of the function $x_n$ into two parts: a sum over the even-numbered indices n = 2m and a sum over the odd-numbered indices n = 2m + 1:

$$X_k = \sum_{m=0}^{N/2-1} x_{2m} e^{-\frac{2\pi i}{N}(2m)k} + \sum_{m=0}^{N/2-1} x_{2m+1} e^{-\frac{2\pi i}{N}(2m+1)k}.$$

One can factor a common multiplier $e^{-\frac{2\pi i}{N}k}$ out of the second sum, as shown in the equation below. It is then clear that the two sums are the DFT of the even-indexed part $x_{2m}$ and the DFT of odd-indexed part $x_{2m+1}$ of the function $x_n$. Denote the DFT of the Even-indexed inputs $x_{2m}$ by $E_k$ and the DFT of the Odd-indexed inputs $x_{2m+1}$ by $O_k$ and we obtain:

$$X_k = \underbrace{\sum_{m=0}^{N/2-1} x_{2m} e^{-\frac{2\pi i}{N/2}mk}}_{\text{DFT of even-indexed part of } x_m} + e^{-\frac{2\pi i}{N}k} \underbrace{\sum_{m=0}^{N/2-1} x_{2m+1} e^{-\frac{2\pi i}{N/2}mk}}_{\text{DFT of odd-indexed part of } x_m} = E_k + e^{-\frac{2\pi i}{N}k} O_k.$$

However, these smaller DFTs have a length of N/2, so we need compute only N/2 outputs: thanks to the periodicity properties of the DFT, the outputs for $N/2 \leq k < N$ from a DFT of length N/2 are identical to the outputs for $0 \leq k < N/2$. That is, $E_{k+N/2} = E_k$ and $O_{k+N/2} = O_k$. The phase factor exp[ − 2πik / N] (called a twiddle factor) obeys the relation: exp[ − 2πi(k + N / 2) / N] = $e^{-\pi i}$exp[ − 2πik / N] = − exp[ − 2πik / N], flipping the sign of the $O_{k+N/2}$ terms. Thus, the whole DFT can be calculated as follows:

$$X_k = \begin{cases} E_k + e^{-\frac{2\pi i}{N}k} O_k & \text{if } k < N/2 \\ E_{k-N/2} - e^{-\frac{2\pi i}{N}(k-N/2)} O_{k-N/2} & \text{if } k \geq N/2. \end{cases}$$

This result, expressing the DFT of length N recursively in terms of two DFTs of size N/2, is the core of the radix-2 DIT fast Fourier transform. The algorithm gains its speed by re-using the results of intermediate computations to compute multiple DFT outputs. Note that final outputs are obtained by a +/− combination of $E_k$ and $O_k$exp( − 2πik / N), which is simply a size-2 DFT; when this is generalized to larger radices below, the size-2 DFT is replaced by a larger DFT (which itself can be evaluated with an FFT). Theses processes is an example of the general technique of divide and conquer algorithms; in many traditional implementations, however, the explicit recursion is

avoided, and instead one traverses the computational tree in breadth-first fashion. The above reexpression of a size-N DFT as two size-N/2 DFTs is sometimes called the Danielson–Lanczos lemma, since the identity was noted by those two authors. They applied their lemma in a "backwards" recursive fashion, repeatedly doubling the DFT size until the transform spectrum converged (although they apparently didn't realize the line arithmetic asymptotic complexity they had achieved). The Danielson–Lanczos work predated widespread availability of computers and required hand calculation (possibly with mechanical aids such as adding machines); they reported a computation time of 140 minutes for a size-64 DFT operating on real inputs to 3–5 significant digits. Cooley and Tukey's 1965 paper reported a running time of 0.02 minutes for a size-2048 complex DFT on an IBM 7094 (probably in 36-bit single precision, ~8 digits).[3] Rescaling the time by the number of operations, this corresponds roughly to a speedup factor of around 800,000. (To put the time for the hand calculation in perspective, 140 minutes for size 64 corresponds to an average of at most 16 seconds per floating-point operation, around 20% of which are multiplications).

### 4.5. Other FFT Algorithms:

There are other FFT algorithms distinct from Cooley–Tukey. For $N = N_1N_2$ with co prime $N_1$ and $N_2$, one can use the Prime-Factor (Good-Thomas) algorithm (PFA), based on the Chinese Remainder Theorem, to factorize the DFT similarly to Cooley–Tukey but without the twiddle factors. The Rader-Brenner algorithm (1976) is a Cooley–Tukey-like factorization but with purely imaginary twiddle factors, reducing multiplications at the cost of increased additions and reduced numerical stability; it was later superseded by the split-radix variant of Cooley–Tukey (which achieves the same multiplication count but with fewer additions and without sacrificing accuracy). Algorithms that recursively factorize the DFT into smaller operations other than DFTs include the Bruun and QFT algorithms. (The Rader-Brenner and QFT algorithms were proposed for power-oftwo sizes, but it is possible that they could be adapted to general composite n. Bruun's algorithm applies to arbitrary even composite sizes.) Bruun's algorithm, in particular, is based on interpreting the FFT as a recursive factorization of the polynomial $z^N - 1$, here into real-coefficient polynomials of the form $z^M - 1$ and $z^{2M} + az^M + 1$.

Another polynomial viewpoint is exploited by the Winograd algorithm, which factorizes $z^N - 1$ into cyclotomic polynomials—these often have coefficients of 1, 0, or −1, and therefore require few (if any) multiplications, so Winograd can be used to obtain minimal-multiplication FFTs and is often used to find efficient algorithms for small factors. Indeed, Winograd showed that the DFT can be computed with only O(N) irrational multiplications, leading to a proven achievable lower bound on the number of multiplications for power-of-two sizes; unfortunately, this comes at the cost of many more additions, a tradeoff no longer favorable on modern processors with hardware multipliers. In particular, Winograd also makes use of the PFA as well as an algorithm by Rader for FFTs of prime sizes.

Rader's algorithm, exploiting the existence of a generator for the multiplicative group modulo prime N, expresses a DFT of prime size n as a cyclic convolution of (composite) size N − 1, which can then be computed by a pair of ordinary FFTs via the convolution theorem (although Winograd uses other convolution methods). Another prime-size FFT is due to L. I. Bluestein, and is

sometimes called the chirp-z algorithm; it also re-expresses a DFT as a convolution, but this time of the same size (which can be zero-padded to a power of two and evaluated by radix-2 Cooley–Tukey FFTs, for example), via the identity $nk = -(k-n)^2/2 + n^2/2 + k^2/2$.

### 4.6. Real Time Applications:

1. EEG Display.
2. SIGVIEW spectrum Analyzer.
3. TDS oscilloscope.
4. Musical Applications.
5. Audio Applications.

## 6. CONCLUSION

Practical oriented research based on various innovative DSP algorithms based on FFT analyzed with real time applications..

**REFERENCES**

[1] Wei Chu; Champagne, B.; "Further studies of a FFT-based auditory spectrum with application in audio classification", proceedings of the ICSP 9th Int. Conf. on Signal Processing, Beijing., Oct 2008 , pp. 2729-2733.

[2] Wei chu; Champagne,B., "A NOISE-ROBUST FFT-BASED AUDITORY SPECTRUM WITH APPLICATION IN AUDIO CLASSIFICATION", .IEEE Transactions on Audio, Speech and Language Processing, vol. 16, Jan. 2008, pp. 137-150.

[3] Adaptive algorithms for acoustic echo cancellation in speech processing International Journal of Research and Reviews in Applied Sciences 2011/4,Radhika Chinaboina K Phani Srinivas Volume: 7,Issue 1

[4] DESIGN AND ANALYSIS OF SPEECH PROCESSING USING KALMAN FILTERING. 2011/2/28, Journal of Theoretical & Applied Information Technology VINEELA MURIKIPUDI, K PHANI SRINIVAS,

[5] A survey on various watermarking methods for gis vector data International Journal of Computer and Electronics Research.2013 Publication, Volume: 2, Issue: 3

[6] Anoop Joyti Sahoo, and Rajesh Kumar Tiwari "A Novel Approach for Hiding Secret data in Program Files" International Journal of Information and Computer Security. Volume 8 Issue 1, March 2016,

[7] Abu Salim, Sachin Tripathi and Rajesh Kumar Tiwari "A secure and timestamp-based communication scheme for cloud environment" Published in International Journal of Electronic Security and Digital Forensics, Volume 6, Issue 4, 319-332.