Analysis of the Cooley–Tukey Fast Fourier Transform

Laura Kocubinski

Boston University Metropolitan College
Department of Computer Science

Spring 2018

In the field of digital signal processing, the discrete Fourier transform (DFT) is a powerful mathematical tool that reveals the frequency content (or spectrum) of a discrete-time signal. As such, the DFT first gained popularity in the mid-twentieth century with rise of digital computing. As the DFT rose in popularity so did the need to calculate it efficiently. However, until the 1965, the DFT could only be computed directly in time $O(N^2)$. In that year, James W. Cooley and John W. Tukey published an algorithm for the computation of the DFT, when N is a composite number, that required only $O(N \log_2 N)$ operations! This reduced the number of operations required to calculate the DFT significantly. This algorithm was the first of its kind and catalyzed the development of additional efficient DFT algorithms, which are now known collectively as the Fast Fourier Transform (FFT). Today the FFT is used in a variety of contexts in addition to digital signal processing, such as MP3 encoding, polynomial multiplication, and more.

In the following paper we will explore the Cooley-Tukey FFT algorithm in depth.

Table of Contents

1. Background

1.1 Historical Context

In 1965, J.W. Cooley and John Tukey published an efficient algorithm for the calculation of the discrete Fourier transform (DFT) [1]. Previously, the DFT could only be computed directly with $O(N^2)$ arithmetic operations, but the Cooley-Tukey algorithm required only $O(N \log_2 N)$.

Carl Friedrich Gauss invented a similar algorithm in 1805, over a century before Cooley and Tukey. It is believed, however, that Cooley and Tukey independently rediscovered this algorithm. For Cooley and Tukey, the dawn of digital computing made it the right time for an efficient DFT algorithm to gain traction, and as such, Cooley and Tukey are largely credited for the development of this algorithm [2].

1.2 The Continuous-Time Fourier Transform (CFT)

The continuous-time Fourier transform describes a continuous-time signal $x(t)$ as a linear combination (integral) of sinusoidal signals at different frequencies.

$$X(j\omega) = \int_{-\infty}^{\infty} x(t) e^{-j\omega t} dt \quad (1.0)$$

This may be intuitive in recognizing Euler's formula in Eq. 1.0.

$$e^{j\theta} = cos\theta + jsin\theta \quad (2.0)$$

where e is the base of the natural logarithm, j is the imaginary unit, and cos and sin are trigonometric functions sine and cosine, respectively [3].

1.3 The Discrete-Time Fourier Transform (DFT)

The discrete-time Fourier transform (DFT) is nothing more than the Fourier transform of a discrete sequence of equally spaced samples $x[n]$. The DFT of a finite-length sequence of length N is

$$\mathrm{X(k)} = \sum_{n=0}^{N-1} x[n]\, W_N^{kn}\,, \quad k = 0, 1, \ldots, N-1 \text{ (3.0)}$$

where $W_N = e^{-j2\pi/N}$ [4].

The output sequence X(k) is the DFT of the input sequence $x[n]$.

2. Computation of the DFT

2.1 Direct Computation

A straightforward calculation of Eq. (3.0) would require N^2 operations: N complex multiplications and $(N-1)$ complex additions. Thus, a direct computation of the DFT requires $O(N^2)$ operations.

FFT algorithms can reduce the number of operations by decomposing the computation into successively smaller DFTs as well as exploiting both the symmetry and periodicity properties of the complex exponential W_N^{kn}.

Since $W_N^{kn} = \cos(2\pi kn/N) - j\sin(2\pi kn/N)$, the periodicity and complex exponential properties are a result of the underlying sine and cosine functions:

$$W_N^{k(N-n)} = W_N^{-kn} = \left(W_N^{kn}\right)^* \text{ (4.0) (complex conjugate symmetry)}$$

$$W_N^{kn} = W_N^{k(n+N)} = W_N^{(k+N)n} \text{ (5.0) (periodicity in } n \text{ and } k)$$

A complex n^{th} root of unity is a complex number such that $W^n = 1$. There are n complex n^{th} roots of unity: $e^{2\pi jk/n}$ for $k = 0, 1, \ldots, n-1$. The principal n^{th} root of unity is $W_n = e^{2\pi j/n}$.

2.2 Derivation of Radix-2 Decimation-in-Time FFT

The radix-2 decimation-in-time (DIT) FFT is perhaps the simplest and most popular form of the Cooley-Tukey algorithm. This algorithm employs a classic divide-and-conquer strategy [5].

This algorithm assumes that N is an exact power of 2

$$N = 2^m, \qquad m = 0, 1, 2, \ldots$$

There exist FFT algorithms for which N is not a power of 2 but those are beyond the scope of this paper.

The radix-2 DIT algorithm divides the sequence $x[n]$ into two $N/2$-point subsequences of even-numbered samples $g[n] = x[2N]$ and odd-numbered samples $h[N] = x[2n+1]$[1].

$$X[k] = \sum_{x\ even} x[n]W_N^{kn} + \sum_{x\ odd} x[n]W_N^{kn}$$

$$X[k] = \sum_{n=0}^{N/2-1} x[2n]\, W_{N/2}^{kn} + W_N^k \sum_{n=0}^{N/2-1} x[2n+1]W_{N/2}^{kn}, \quad k = 0,1, \ldots, N-1 \text{ (6.0)}$$

This is the divide step. The DFT is then calculated for $N/2$ even-indexed inputs and $N/2$ odd-indexed inputs. This is the conquer step. The results of the sub problems are then combined to calculate the size N DFT (combine). This is repeated recursively for p stages.

The radix-2 decimation-in-time decomposition may be visualized in Figure 1. Here an N-point DFT computation is decomposed into $N/2$-point DFT computations ($N = 8$).

[1] In discussing the FFT, point and sample are often used to reference a sequence value.

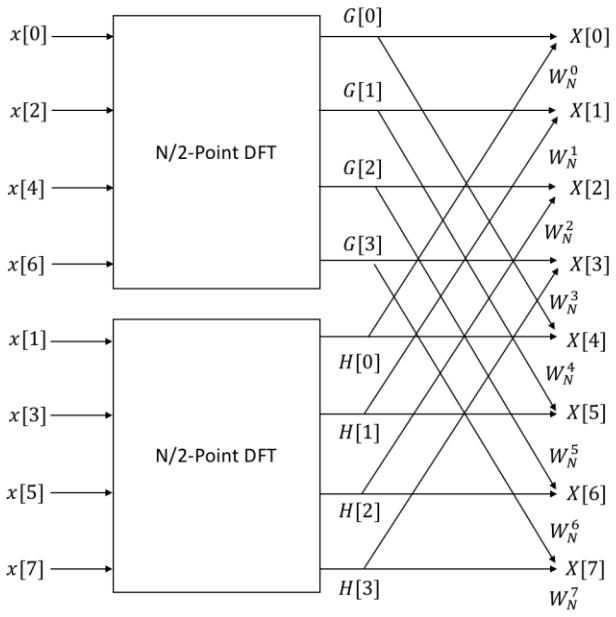
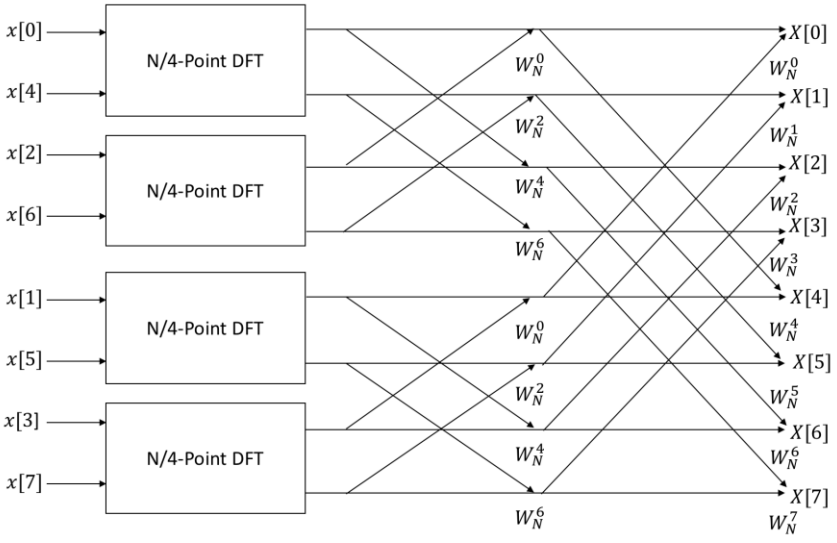


Figure 1 *Flow graph of DIT of an N-point DFT computation into N/2-point DFT computations (N=8).*

It is straightforward to calculate that this will yield $\frac{N^2}{2} + N$ computations.

Following the divide-and-conquer strategy, the subsequences would divide again to create four $N/4$-point DFTs. However, this algorithm gains its speed by reusing the results of intermediate computations.



This will yield $\frac{N^2}{4} + 2N$ computations. This pattern will continue until there are 1-point DFTs (base case). The DFT will yield p stages where $p = \log_2 N$. It is easy to see that following this pattern we have $\frac{N^2}{2^p} + pN$ computations. We can characterize this FFT algorithm with the recurrence:

$$T(n) = 2T\left(\frac{n}{2}\right) + cn \quad (7.0)$$

By using the master theorem,

$$T(n) = O(n \log_2 n)$$

3. Implementation

3.1 Recursive FFT Pseudocode

Below is the pseudocode of a recursive FFT algorithm [5]. The algorithm itself is quite elegant. The input sequence $x[n]$ contains N elements, where N is a power of 2.

```
fft_rec(x)
    n = length of x // n is a power of 2
    if n = 1 // base case of recursion
        return a
    W_n = e^(2πj/n) // W_n is the principal n^th root of unity
    W = 1
    x^[0] = [x_0, x_2, ..., x_(n-2)]
    x^[1] = [x_1, x_3, ..., x_(n-1)]
    y^[0] = Recursive-FFT(x^[0])
    y^[1] = Recursive-FFT(x^[1])
    for k = 0 to n/2 – 1
        y_k = y_k^[0] + W y_k^[1]
        y_(k+(n/2)) = y_k^[0] − W y_k^[1]
        W = W · W_n
    return y
```

3.2 Recursive FFT in Matlab

The recursive FFT algorithm is outlined above is implemented below in Matlab code.

```
function y = fft_rec(x)
n = length(x)
if n == 1
        y = x
else
        w_n = cos(2*pi/n) + i*sin(2*pi/n)
        w = 1
        a_top = x(2:2:n)
        a_bot = x(1:2:(n-1))
        y_top = my_fft1(a_top)
        y_bot = my_fft1(a_bot)
        m = n/2
        for k = 1:m
                y(k) =  y_top(k) + w*y_bot(k)
                y(k+m) = y_top(k) - w*y_bot(k)
                w = w*w_n
        end
end
end
```

Of course it follows that the FFT algorithm may be implemented iteratively as well [6].

4. Errors in Computation

The accuracy of the FFT and the error associated with the computation of the FFT is an extensive field of study unto itself. As such, an in-depth study of the error associated with the FFT is beyond the scope of this paper.

Nonetheless, it is important to acknowledge there is error associated with the computation of the FFT. One possible source of error, for example, may be the approximation of the coefficients (also known as twiddle factors). This approximation error is inherent in the limitations of computation hardware (i.e., the finite-length of registers). Another source of error in the computation of the FFT may be the round-off error associated with the addition and

multiplication. How the round-off error is compounded may be dependent on a number of variables, such as how many stages are involved in the FFT, etc.

The sources of error may be broad and nuanced. For example, how the numbers are represented (as either fixed-point or floating-point) may contribute to errors and possible overflow. Additionally, FFT error may be related to the exact algorithm used (e.g., iterative or recursive) [7].

5. Conclusion

The FFT is perhaps one of the most important and popular algorithms in the modern world. Its applications are far-reaching, ranging from engineering to mathematics, and beyond. As such, the FFT has rightfully generated a lot of study and interest from a broad set of communities. This paper has only scratched the surface with what the FFT is and has to offer.

Works Cited

1. Cooley, James W., and John W. Tukey. "An Algorithm for Machine Calculation of Complex Fourier Series". *Mathematics of Computation,* Vol. 19, No. 90, 1965, 297-301.

2. Heideman, Michael T., et al. "Gauss and the History of the Fast Fourier Transform." *IEEE ASSP Magazine,* Vol. 1, Issue 4, 1984, 14-21.

3. Oppenheim, Alan V., et al. *Signals & Systems, Second Edition*. Prentice Hall, 1997.

4. Oppenheim, Alan V., and Ronald W. Schafer. *Discrete-Time Signal Processing, Third Edition*. Prentice Hall, 2010.

5. Cormen, Thomas H., et al. *Introduction to Algorithms, Third Edition*. The MIT Press, 2009.

6. Wörner, Stefan. "Fast Fourier Transform." Numerical Analysis Seminar, Swiss Federal Institute of Technology Zurich.

7. Schatzman, James C. "Accuracy of the Discrete Fourier Transform and the Fast Fourier Transform." *SIAM Journal on Scientific Computing,* Vol. 17, No. 5, 1996, 1150-1166.