The FFT Algorithm*

Robert Nowak

Abstract

The FFT, an efficient way to compute the DFT, is introduced and derived throughout this module.

Definition 1: FFT
(Fast Fourier Transform) An efficient **computational algorithm** for computing the DFT[1].

1 The Fast Fourier Transform FFT

DFT can be **expensive** to compute **directly**

$$\forall k, 0 \leq k \leq N-1 : \left(X[k] = \sum_{n=0}^{N-1} \left(x[n] e^{-\left(i2\pi \frac{k}{N} n\right)} \right) \right)$$

For each k, we must execute:

- N complex multiplies
- $N-1$ complex adds

The total cost of direct computation of an N-point DFT is

- N^2 complex multiplies
- $N(N-1)$ complex adds

How many adds and mults of **real** numbers are required?
This " $O\left(N^2\right)$ " computation rapidly gets out of hand, as N gets large:

N	1	10	100	1000	10^6
N^2	1	100	10,000	10^6	10^{12}

Table 1

*Version 2.6: Jul 22, 2005 2:30 pm GMT-5
†http://creativecommons.org/licenses/by/1.0
[1]"Discrete Fourier Transform (DFT)" <http://cnx.org/content/m10249/latest/>

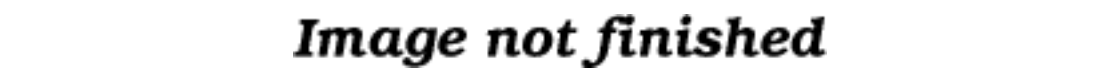

Figure 1

The FFT provides us with a much more **efficient** way of computing the DFT. The FFT requires only " $O\left(N\log N\right)$" computations to compute the N-point DFT.

N	10	100	1000	10^6
N^2	100	10,000	10^6	10^{12}
$N\log_{10}N$	10	200	3000	6×10^6

Table 2

How long is $10^{12}\mu$sec? More than 10 days! How long is $6\times 10^6\mu$sec?

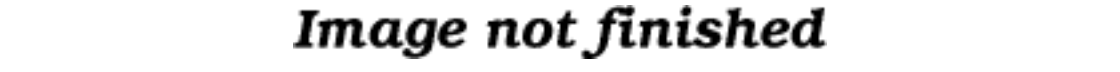

Figure 2

The FFT and digital computers revolutionized DSP (1960 - 1980).

2 How does the FFT work?

- The FFT exploits the **symmetries** of the complex exponentials ${W_N}^{kn}=e^{-\left(i\frac{2\pi}{N}kn\right)}$
- ${W_N}^{kn}$ are called "**twiddle factors**"

Rule 1: Complex Conjugate Symmetry

$$ {W_N}^{k(N-n)}={W_N}^{-(kn)}=\overline{{W_N}^{kn}} $$

$$ e^{-\left(i2\pi\frac{k}{N}(N-n)\right)}=e^{i2\pi\frac{k}{N}n}=\overline{e^{-\left(i2\pi\frac{k}{N}n\right)}} $$

Rule 2: Periodicity in n and k

$$ {W_N}^{kn}={W_N}^{k(N+n)}={W_N}^{(k+N)n} $$

$$ e^{-\left(i\frac{2\pi}{N}kn\right)}=e^{-\left(i\frac{2\pi}{N}k(N+n)\right)}=e^{-\left(i\frac{2\pi}{N}(k+N)n\right)} $$

$$ W_N=e^{-\left(i\frac{2\pi}{N}\right)} $$

3 Decimation in Time FFT

- Just one of **many** different FFT algorithms
- The **idea** is to build a DFT out of smaller and smaller DFTs by decomposing $x[n]$ into smaller and smaller subsequences.
- Assume $N = 2^m$ (a power of 2)

3.1 Derivation

N is **even**, so we can complete $X[k]$ by separating $x[n]$ into **two** subsequences each of length $\frac{N}{2}$.

$$x[n] \rightarrow \begin{cases} \frac{N}{2} \text{ if } n = \text{even} \\ \frac{N}{2} \text{ if } n = \text{odd} \end{cases}$$

$$\forall k, 0 \leq k \leq N-1 : \left(X[k] = \sum_{n=0}^{N-1} \left(x[n] {W_N}^{kn} \right) \right)$$

$$X[k] = \sum \left(x[n] {W_N}^{kn} \right) + \sum \left(x[n] {W_N}^{kn} \right)$$

where $0 \leq r \leq \frac{N}{2} - 1$. So

$$\begin{array}{rcl} X[k] & = & \sum_{r=0}^{\frac{N}{2}-1} \left(x[2r] {W_N}^{2kr} \right) + \sum_{r=0}^{\frac{N}{2}-1} \left(x[2r+1] {W_N}^{(2r+1)k} \right) \\ & = & \sum_{r=0}^{\frac{N}{2}-1} \left(x[2r] \left({W_N}^2 \right)^{kr} \right) + {W_N}^k \sum_{r=0}^{\frac{N}{2}-1} \left(x[2r+1] \left({W_N}^2 \right)^{kr} \right) \end{array} \quad (1)$$

where ${W_N}^2 = e^{-\left(i\frac{2\pi}{N}2\right)} = e^{-\left(i\frac{2\pi}{\frac{N}{2}}\right)} = W_{\frac{N}{2}}$. So

$$X[k] = \sum_{r=0}^{\frac{N}{2}-1} \left(x[2r] {W_{\frac{N}{2}}}^{kr} \right) + {W_N}^k \sum_{r=0}^{\frac{N}{2}-1} \left(x[2r+1] {W_{\frac{N}{2}}}^{kr} \right)$$

where $\sum_{r=0}^{\frac{N}{2}-1} \left(x[2r] {W_{\frac{N}{2}}}^{kr} \right)$ is $\frac{N}{2}$-point DFT of even samples ($G[k]$) and $\sum_{r=0}^{\frac{N}{2}-1} \left(x[2r+1] {W_{\frac{N}{2}}}^{kr} \right)$ is $\frac{N}{2}$-point DFT of odd samples ($H[k]$).

$$\forall k, 0 \leq k \leq N-1 : \left(X[k] = G[k] + {W_N}^k H[k] \right)$$

Decomposition of an N-point DFT as a sum of 2 $\frac{N}{2}$-point DFTs.

Why would we want to do this? **Because it is more efficient!**

NOTE: Cost to compute an N-point DFT is approximately N^2 complex mults and adds.

But decomposition into 2 $\frac{N}{2}$-point DFTs + combination requires only

$$\left(\frac{N}{2}\right)^2 + \left(\frac{N}{2}\right)^2 + N = \frac{N^2}{2} + N$$

where the first part is the number of complex mults and adds for $\frac{N}{2}$-point DFT, $G[k]$. The second part is the number of complex mults and adds for $\frac{N}{2}$-point DFT, $H[k]$. The third part is the number of complex mults and adds for combination. And the total is $\frac{N^2}{2} + N$ complex mults and adds.

Example 1: Savings
For $N = 1000$,

$$N^2 = 10^6$$

$$\frac{N^2}{2} + N = \frac{10^6}{2} + 1000$$

Because 1000 is small compared to 500,000,

$$\frac{N^2}{2} + N \approx \frac{10^6}{2}$$

So why stop here?! Keep decomposing. Break each of the $\frac{N}{2}$-point DFTs into two $\frac{N}{4}$-point DFTs, *etc.*, ...
We can keep decomposing:

$$\frac{N}{2^1} = \left\{ \frac{N}{2}, \frac{N}{4}, \frac{N}{8}, \ldots, \frac{N}{2^{m-1}}, \frac{N}{2^m} \right\} = 1$$

where

$$m = \log_2 N = \text{ times}$$

Computational cost: N-pt DFT [U+F577] two $\frac{N}{2}$-pt DFTs. The cost is $N^2 \to 2\left(\frac{N}{2}\right)^2 + N$. So replacing each $\frac{N}{2}$-pt DFT with two $\frac{N}{4}$-pt DFTs will reduce cost to

$$2\left(2\left(\frac{N}{4}\right)^2 + \frac{N}{2}\right) + N = 4\left(\frac{N}{4}\right)^2 + 2N = \frac{N^2}{2^2} + 2N = \frac{N^2}{2^p} + pN$$

As we keep going $p = \{3, 4, \ldots, m\}$, where $m = \log_2 N$. We get the cost

$$\frac{N^2}{2^{\log_2 N}} + N\log_2 N = \frac{N^2}{N} + N\log_2 N = N + N\log_2 N$$

$N + N\log_2 N$ is the total number of complex adds and mults.
For large N, cost $\approx N\log_2 N$ or " $O\left(N\log_2 N\right)$", since $(N\log_2 N \gg N)$ for large N.

Image not finished

Figure 3: $N = 8$ point FFT. Summing nodes ${W_n}^k$ twiddle multiplication factors.

NOTE: Weird order of time samples

Image not finished

Figure 4: This is called "butterflies."
