

DECIMATION-IN-FREQUENCY (DIF) RADIX-2 FFT

Version 1.6: Sep 17, 2006 8:47 am GMT-5

Douglas L. Jones

This work is produced by The Connexions Project and licensed under the
Creative Commons Attribution License *

Abstract

The radix-2 algorithms are the simplest FFT algorithms. The decimation-in-frequency (DIF) radix-2 FFT partitions the DFT computation into even-indexed and odd-indexed outputs, which can each be computed by shorter-length DFTs of different combinations of input samples. Recursive application of this decomposition to the shorter-length DFTs results in the full radix-2 decimation-in-frequency FFT.

The radix-2 decimation-in-frequency and decimation-in-time¹ fast Fourier transforms (FFTs) are the simplest FFT algorithms². Like all FFTs, they compute the discrete Fourier transform (DFT)³

$$\begin{aligned} X(k) &= \sum_{n=0}^{N-1} \left(x(n) e^{-i \frac{2\pi nk}{N}} \right) \\ &= \sum_{n=0}^{N-1} \left(x(n) W_N^{nk} \right) \end{aligned} \quad (1)$$

where for notational convenience $W_N^k = e^{-i \frac{2\pi k}{N}}$. FFT algorithms gain their speed by reusing the results of smaller, intermediate computations to compute multiple DFT frequency outputs.

1 Decimation in frequency

The radix-2 decimation-in-frequency algorithm rearranges the discrete Fourier transform (DFT) equation (1) into two parts: computation of the even-numbered discrete-frequency indices $X(k)$ for $k = [0, 2, 4, \dots, N-2]$ (or $X(2r)$ as in (2)) and computation of the odd-numbered indices $k = [1, 3, 5, \dots, N-1]$ (or $X(2r+1)$ as in (3))

$$\begin{aligned} X(2r) &= \sum_{n=0}^{N-1} \left(x(n) W_N^{2rn} \right) \\ &= \sum_{n=0}^{\frac{N}{2}-1} \left(x(n) W_N^{2rn} \right) + \sum_{n=0}^{\frac{N}{2}-1} \left(x\left(n + \frac{N}{2}\right) W_N^{2r\left(n + \frac{N}{2}\right)} \right) \\ &= \sum_{n=0}^{\frac{N}{2}-1} \left(x(n) W_N^{2rn} \right) + \sum_{n=0}^{\frac{N}{2}-1} \left(x\left(n + \frac{N}{2}\right) W_N^{2rn} \right) \\ &= \sum_{n=0}^{\frac{N}{2}-1} \left(\left(x(n) + x\left(n + \frac{N}{2}\right) \right) W_N^{rn} \right) \\ &= \text{DFT}_{\frac{N}{2}} \left[x(n) + x\left(n + \frac{N}{2}\right) \right] \end{aligned} \quad (2)$$

*<http://creativecommons.org/licenses/by/1.0>

¹"Decimation-in-time (DIT) Radix-2 FFT" <<http://cnx.org/content/m12016/latest/>>

²"Overview of Fast Fourier Transform (FFT) Algorithms" <<http://cnx.org/content/m12026/latest/>>

³"DFT Definition and Properties" <<http://cnx.org/content/m12019/latest/>>

$$\begin{aligned}
 X(2r+1) &= \sum_{n=0}^{N-1} \left(x(n) W_N^{(2r+1)n} \right) \\
 &= \sum_{n=0}^{\frac{N}{2}-1} \left(\left(x(n) + W_N^{\frac{N}{2}} x\left(n + \frac{N}{2}\right) \right) W_N^{(2r+1)n} \right) \\
 &= \sum_{n=0}^{\frac{N}{2}-1} \left(\left((x(n) - x\left(n + \frac{N}{2}\right)) W_N^n \right) W_N^{r\frac{N}{2}} \right) \\
 &= \text{DFT}_{\frac{N}{2}} \left[(x(n) - x\left(n + \frac{N}{2}\right)) W_N^n \right]
 \end{aligned} \tag{3}$$

The mathematical simplifications in (2) and (3) reveal that both the even-indexed and odd-indexed frequency outputs $X(k)$ can each be computed by a length- $\frac{N}{2}$ DFT. The inputs to these DFTs are sums or differences of the first and second halves of the input signal, respectively, where the input to the short DFT producing the odd-indexed frequencies is multiplied by a so-called **twiddle factor** term $W_N^k = e^{-j\frac{2\pi k}{N}}$. This is called a **decimation in frequency** because the frequency samples are computed separately in alternating groups, and a **radix-2** algorithm because there are two groups. Figure 1 graphically illustrates this form of the DFT computation. This conversion of the full DFT into a series of shorter DFTs with a simple preprocessing step gives the decimation-in-frequency FFT its computational savings.

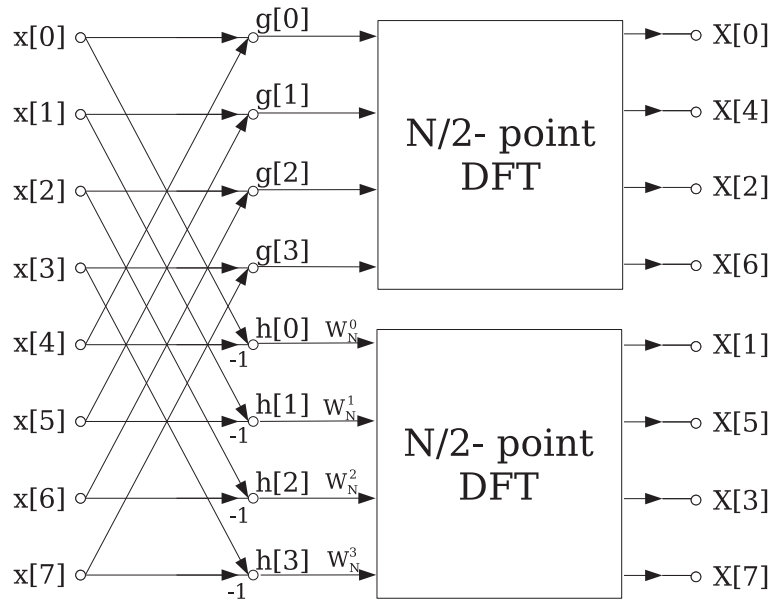


Figure 1: Decimation in frequency of a length- N DFT into two length- $\frac{N}{2}$ DFTs preceded by a preprocessing stage.

Whereas direct computation of all N DFT frequencies according to the DFT equation⁴ would require N^2 complex multiplies and $N^2 - N$ complex additions (for complex-valued data), by breaking the computation into two short-length DFTs with some preliminary combining of the data, as illustrated in Figure 1, the computational cost is now

New Operation Counts

⁴"DFT Definition and Properties" <<http://cnx.org/content/m12019/latest/>>

- $2\left(\frac{N}{2}\right)^2 + N = \frac{N^2}{2} + \frac{N}{2}$ complex multiplies
- $2\frac{N}{2}\left(\frac{N}{2} - 1\right) + N = \frac{N^2}{2}$ complex additions

This simple manipulation has reduced the total computational cost of the DFT by almost a factor of two!

The initial combining operations for both short-length DFTs involve parallel groups of two time samples, $x(n)$ and $x\left(n + \frac{N}{2}\right)$. One of these so-called **butterfly** operations is illustrated in Figure 2. There are $\frac{N}{2}$ butterflies per **stage**, each requiring a complex addition and subtraction followed by one **twiddle-factor** multiplication by $W_N^n = e^{-j\left(\frac{2\pi n}{N}\right)}$ on the lower output branch.

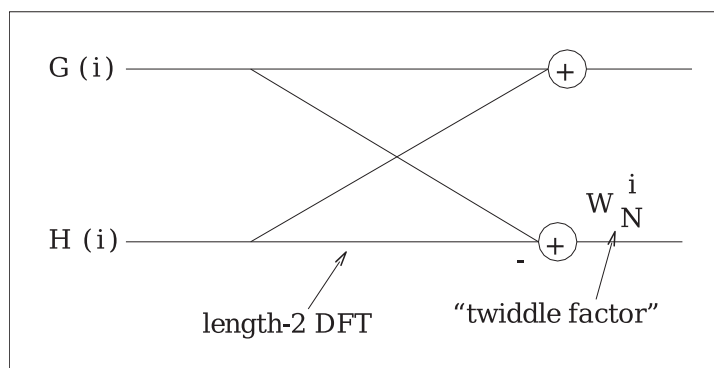


Figure 2: DIF butterfly: twiddle factor after length-2 DFT

It is worthwhile to note that the initial add/subtract part of the DIF butterfly is actually a length-2 DFT! The theory of multi-dimensional index maps⁵ shows that this must be the case, and that FFTs of any factorable length may consist of successive stages of shorter-length FFTs with twiddle-factor multiplications in between. It is also worth noting that this butterfly differs from the decimation-in-time radix-2 butterfly⁶ in that the twiddle factor multiplication occurs *after* the combining.

2 Radix-2 decimation-in-frequency algorithm

The same radix-2 decimation in frequency can be applied recursively to the two length- $\frac{N}{2}$ DFT⁷s to save additional computation. When successively applied until the shorter and shorter DFTs reach length-2, the result is the radix-2 decimation-in-frequency FFT algorithm (Figure 3).

⁵"Multidimensional Index Maps" <<http://cnx.org/content/m12025/latest/>>

⁶"Decimation-in-time (DIT) Radix-2 FFT", Figure 2 <<http://cnx.org/content/m12016/latest/#fig2>>

⁷"DFT Definition and Properties" <<http://cnx.org/content/m12019/latest/>>

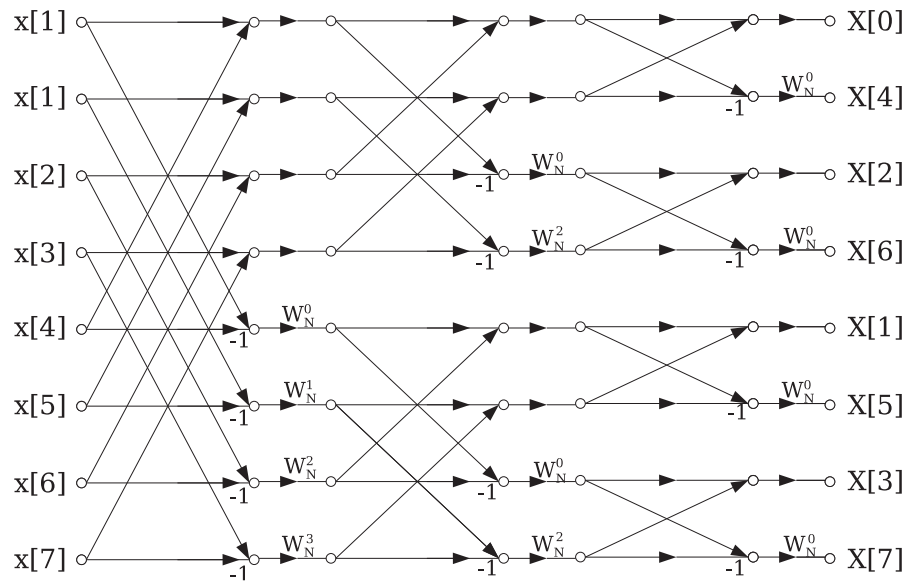


Figure 3: Radix-2 decimation-in-frequency FFT for a length-8 signal

The full radix-2 decimation-in-frequency decomposition illustrated in Figure 3 requires $M = \log_2 N$ stages, each with $\frac{N}{2}$ butterflies per stage. Each butterfly requires 1 complex multiply and 2 adds per butterfly. The total cost of the algorithm is thus

Computational cost of radix-2 DIF FFT

- $\frac{N}{2} \log_2 N$ complex multiplies
- $N \log_2 N$ complex adds

This is a remarkable savings over direct computation of the DFT. For example, a length-1024 DFT would require 1048576 complex multiplications and 1047552 complex additions with direct computation, but only 5120 complex multiplications and 10240 complex additions using the radix-2 FFT, a savings by a factor of 100 or more. The relative savings increase with longer FFT lengths, and are less for shorter lengths. Modest additional reductions in computation can be achieved by noting that certain twiddle factors, namely W_N^0 , $W_N^{\frac{N}{2}}$, $W_N^{\frac{N}{4}}$, $W_N^{\frac{3N}{4}}$, require no multiplications, or fewer real multiplies than other ones. By implementing special butterflies for these twiddle factors as discussed in FFT algorithm and programming tricks⁸, the computational cost of the radix-2 decimation-in-frequency FFT can be reduced to

- $2N \log_2 N - 7N + 12$ real multiplies
- $3N \log_2 N - 3N + 4$ real additions

The decimation-in-frequency FFT is a flow-graph reversal of the decimation-in-time⁹ FFT: it has the same twiddle factors (in reverse pattern) and the same operation counts.

NOTE: In a decimation-in-frequency radix-2 FFT as illustrated in Figure 3, the output is in **bit-reversed** order (hence "decimation-in-frequency"). That is, if the frequency-sample index n is

⁸"Efficient FFT Algorithm and Programming Tricks" <<http://cnx.org/content/m12021/latest/>>

⁹"Decimation-in-time (DIT) Radix-2 FFT" <<http://cnx.org/content/m12016/latest/>>

written as a binary number, the order is that binary number reversed. The bit-reversal process is illustrated here¹⁰.

It is important to note that, if the input data are in order before beginning the FFT computations, the outputs of each butterfly throughout the computation can be placed in the same memory locations from which the inputs were fetched, resulting in an **in-place algorithm** that requires no extra memory to perform the FFT. Most FFT implementations are in-place, and overwrite the input data with the intermediate values and finally the output.

¹⁰"Decimation-in-time (DIT) Radix-2 FFT", Example 1: N=8 <<http://cnx.org/content/m12016/latest/#ex1>>