THE FAST FOURIER TRANSFORM (FFT)

1. DIRECT COMPUTATION
2. RADIX-2 FFT
3. DECIMATION-IN-TIME FFT
4. FLOWGRAPHS
5. BIT REVERSAL PERMUTATION
6. COMPLEXITY
7. DECIMATION-IN-FREQUENCY FFT

THE FFT

A fast Fourier transform (FFT) is any fast algorithm for computing the DFT. The development of FFT algorithms had a tremendous impact on computational aspects of signal processing and applied science. The DFT of an N-point signal

$$\{x[n], 0 \leq n \leq N-1\}$$

is defined as

$$X[k] = \sum_{n=0}^{N-1} x[n]\, W_N^{-kn}, \qquad 0 \leq k \leq N-1$$

where

$$W_N = e^{j\frac{2\pi}{N}} = \cos\left(\frac{2\pi}{N}\right) + j\, \sin\left(\frac{2\pi}{N}\right)$$

is the principal N-th root of unity.

DIRECT DFT COMPUTATION

Direct computation of $X[k]$ for $0 \leq k \leq N-1$ requires

$(N-1)^2$ complex multiplications

$N(N-1)$ complex additions

RADIX-2 FFT

The radix-2 FFT algorithms are used for data vectors of lengths $N = 2^K$. They proceed by dividing the DFT into two DFTs of length $N/2$ each, and iterating. There are several types of radix-2 FFT algorithms, the most common being the *decimation-in-time* (DIT) and the *decimation-in-frequency* (DIF). This terminology will become clear in the next sections.

Preliminaries

The development of the FFT will call on two properties of W_N. The first property is:

$$\boxed{W_N^2 = W_{N/2}}$$

which is derived as

$$\begin{aligned} W_N^2 &= e^{-j\frac{2\pi}{N}\cdot 2} \\ &= e^{-j\frac{2\pi}{N/2}} \\ &= W_{N/2}. \end{aligned}$$

More generally, we have

$$W_N^{2nk} = W_{N/2}^{nk}.$$

The second property is:

$$\boxed{W_N^{k+\frac{N}{2}} = -W_N^k}$$

which is derived as

$$\begin{aligned} W_N^{k+\frac{N}{2}} &= e^{j\frac{2\pi}{N}(k+\frac{N}{2})} \\ &= e^{j\frac{2\pi}{N}k} \cdot e^{j\frac{2\pi}{N}(\frac{N}{2})} \\ &= e^{j\frac{2\pi}{N}k} \cdot e^{j\pi} \\ &= -e^{j\frac{2\pi}{N}k} \\ &= -W_N^k \end{aligned}$$

Consider an N-point signal $x[n]$ of even length. The derivation of the DIT radix-2 FFT begins by splitting the sum into two parts — one part for the even-indexed values $x[2n]$ and one part for the odd-indexed values $x[2n+1]$. Define two $N/2$-point signals $x_1[n]$ and $x_2[n]$ as

$$x_0[n] = x[2n]$$
$$x_1[n] = x[2n+1]$$

for $0 \leq n \leq N/2-1$. The DFT of the N-point signal $x[n]$ can be written as

$$X[k] = \sum_{\substack{n=0 \\ n\ even}}^{N-1} x[n]\, W_N^{-nk} + \sum_{\substack{n=0 \\ n\ odd}}^{N-1} x[n]\, W_N^{-nk}$$

which can be written as

$$\begin{aligned} X[k] &= \sum_{n=0}^{N/2-1} x[2n]\, W_N^{-2nk} + \sum_{n=0}^{N/2-1} x[2n+1]\, W_N^{-(2n+1)k} \\ &= \sum_{n=0}^{N/2-1} x_0[n]\, W_N^{-2nk} + \sum_{n=0}^{N/2-1} x_1[n]\, W_N^{-(2n+1)k} \\ &= \sum_{n=0}^{N/2-1} x_0[n]\, W_N^{-2nk} + W_N^{-k} \cdot \sum_{n=0}^{N/2-1} x_1[n]\, W_N^{-2nk} \\ &= \sum_{n=0}^{N/2-1} x_0[n]\, W_{N/2}^{-nk} + W_N^{-k} \cdot \sum_{n=0}^{N/2-1} x_1[n]\, W_{N/2}^{-nk} \end{aligned}$$

where we used the first identity above. Recognizing that the $\frac{N}{2}$-pont

DFT of $x_0[n]$ and $x_1[n]$ are given by

$$X_0[k] = \mathsf{DFT}_{\frac{N}{2}}\{x_0[n]\} = \sum_{n=0}^{N/2-1} x_0[n]\, W_{N/2}^{-nk}$$

$$X_1[k] = \mathsf{DFT}_{\frac{N}{2}}\{x_1[n]\} = \sum_{n=0}^{N/2-1} x_1[n]\, W_{N/2}^{-nk}$$

we obtain the equation

$$\boxed{X[k] = X_0[k] + W_N^{-k} \cdot X_1[k].}$$

The signal $x_0[n]$ is an $N/2$-point signal, so its DFT is also an $N/2$-point signal. When k is taken outside the range $0 \leq k \leq N/2 - 1$ the DFT coefficients are periodic with a period of $N/2$:

$$X_0[k] = X_0[\langle k \rangle_{N/2}] \qquad \text{or} \qquad X_0[k] = X_0[k + N/2],$$

and likewise for $X_1[k]$

$$X_1[k] = X_1[\langle k \rangle_{N/2}] \qquad \text{or} \qquad X_1[k] = X_1[k + N/2].$$

For example, if $N = 8$, then

$$\begin{aligned}
X[0] &= X_0[0] + 1 \cdot X_1[0] \\
X[1] &= X_0[1] + W_8^{-1} \cdot X_1[1] \\
X[2] &= X_0[2] + W_8^{-2} \cdot X_1[2] \\
X[3] &= X_0[3] + W_8^{-3} \cdot X_1[3] \\
X[4] &= X_0[0] + W_8^{-4} \cdot X_1[0] \\
X[5] &= X_0[1] + W_8^{-5} \cdot X_1[1] \\
X[6] &= X_0[2] + W_8^{-6} \cdot X_1[2] \\
X[7] &= X_0[3] + W_8^{-7} \cdot X_1[3].
\end{aligned}$$

Using $W_N^{k+\frac{N}{2}} = -W_N^k$ derived earlier, we obtain for this example

$$\begin{aligned}
X[0] &= X_0[0] + 1 \cdot X_1[0] \\
X[1] &= X_0[1] + W_8^{-1} \cdot X_1[1] \\
X[2] &= X_0[2] + W_8^{-2} \cdot X_1[2] \\
X[3] &= X_0[3] + W_8^{-3} \cdot X_1[3] \\
X[4] &= X_0[0] - 1 \cdot X_1[0] \\
X[5] &= X_0[1] - W_8^{-1} \cdot X_1[1] \\
X[6] &= X_0[2] - W_8^{-2} \cdot X_1[2] \\
X[7] &= X_0[3] - W_8^{-3} \cdot X_1[3].
\end{aligned}$$

With this modification, the DFT coefficients $X_2[k]$ need only be multiplied by W_8^{-k} for $1 \leq k \leq N/2-1$, incurring $N/2-1$ multiplications.

In general, one has

$$X[k] = X_0[k] + W_N^{-k} \cdot X_1[k] \qquad \text{for } 0 \leq k \leq \frac{N}{2} - 1 \quad (1)$$

$$X[k + \frac{N}{2}] = X_0[k] - W_N^{-k} \cdot X_1[k] \qquad \text{for } 0 \leq k \leq \frac{N}{2} - 1 \quad (2)$$

The multipliers W_N^k are known as *twiddle factors*.

The Matlab code fragment illustrates this relation for $N = 8$:

```
>> x = [82 44 62 79 92 74 18 41]';
>>
>> % "decimate" x(n) in time:
>> x0 = x(1:2:8);
>> x1 = x(2:2:8);
>>
>> % Take the half-length DFTs:
>> X0 = fft(x0);
>> X1 = fft(x1);
>>
>> % define constants
>> i = sqrt(-1);
>> N = 8;
>> W = exp(2*pi/N*i);
>> k = [0:N/2-1]';
>>
```

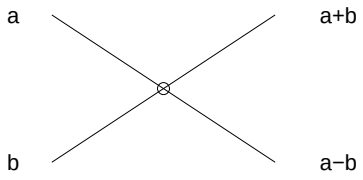
```
>> X0+W.^(-k).*X1

ans =

         492
     -58.083 -     49.657i
          94 +           2i
      38.083 +     38.343i

>>
>> X0-W.^(-k).*X1

ans =

          16
      38.083 -     38.343i
          94 -           2i
     -58.083 +     49.657i


>> % verify that this gives the DFT of x(n):
>> fft(x)

ans =

         492
     -58.083 -     49.657i
```

```
    94 +          2i
38.083 +     38.343i
    16
38.083 -     38.343i
    94 -          2i
-58.083 +     49.657i
```

If $N/2$ can be further divided by 2, then this same procedure can be used to calculated the $N/2$-point DFTs. It is useful to illustrate the radix-2 FFT algorithm with a flowgraph, as developed here.

The expression above shows how an N-point DFT can be computed using two $N/2$-point DFTs. After taking the two $N/2$-point DFTs it only remains to multiply the result of the second DFT with the terms W_N^k and to combine the results by adding and subtracting.

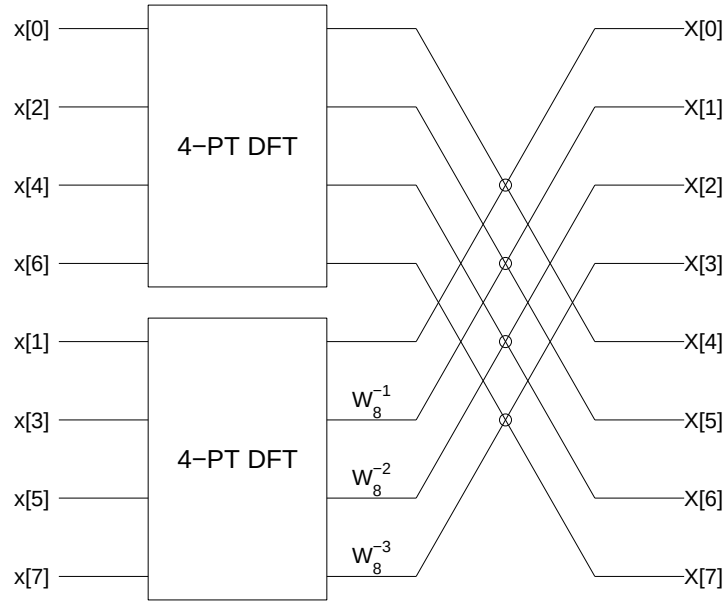
The flowgraph for the sum and difference operation is called the *butterfly*. This unit will be used as a shorthand notation for the sum and difference, to simplify the flowgraphs for the FFT.



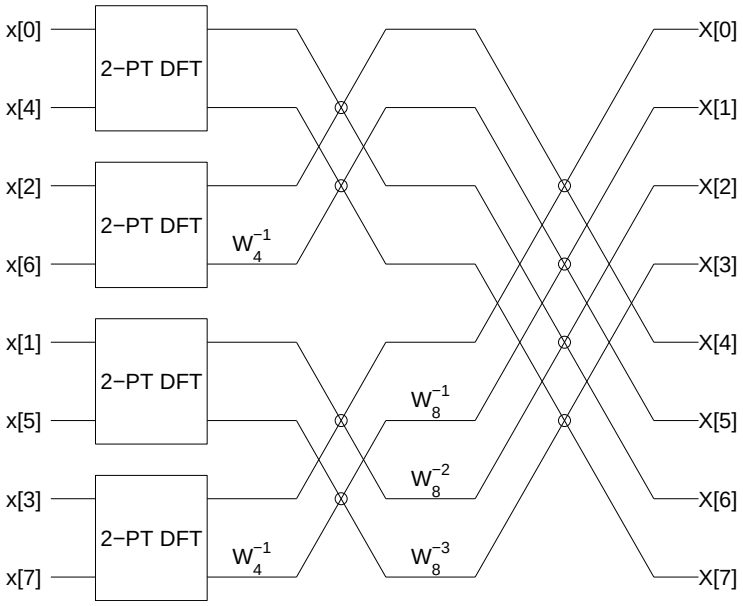
The decimation-in-time FFT for an 8-point DFT consists of

1. 2 4-point DFT computations,
2. the twiddle factors,
3. butterflies

We order the input so the even-indexed terms come first.

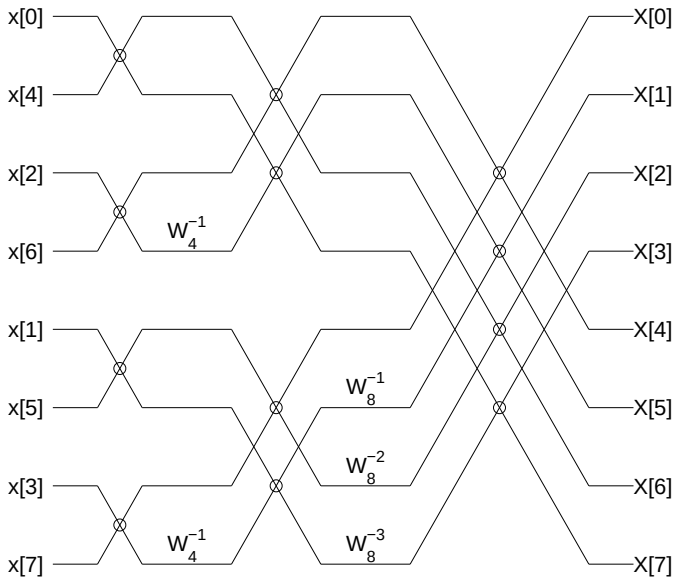


The decomposition of the N-point DFT into two $\frac{N}{2}$-point DFTs can be repeated (provided N is divisible by 4).



FULL FLOWGRAPH

The 2-point DFT is simply a butterfly (sum/difference), so the final FFT has 3 stages.



Remarks

1. This FFT algorithm can be implemented *in-place*.
2. The number of stages is $\log_2 N$. Because each stages has a complexity of order N, the overall complexity is of order $N \log_2 N$.
3. In the DIT flowgraph the signal $x(n)$ is *bit-reversed*.

BIT REVERSAL PERMUTATION

Notice that the input for the full DIT radix-2 FFT flowgraph is permuted.

Before the in-place implementation of the DIT FFT algorithm can be done, it is necessarily to first shuffle the the sequence $x(n)$ according to this permutation.

The required permutation corresponds to reversing the binary representation of the index.

n	binary	bit-rev	n'
0	000	000	0
1	001	100	4
2	010	010	2
3	011	110	6
4	100	001	1
5	101	101	5
6	110	011	3
7	111	111	7

To determine the arithmetic complexity of this algorithm for computing the DFT, let $A_c(N)$ and $M_c(N)$ denote respectively the number of complex additions and multiplications for computing the DFT of an N-point complex sequence $x[n]$. Let N be a power of 2, $N = 2^K$. Then, according to the procedure above, one has

$$A_c(N) = 2\,A_c(N/2) + N$$
$$M_c(N) = 2\,M_c(N/2) + \frac{N}{2} - 1$$

as N complex additions and $\frac{N}{2} - 1$ complex multiplications are required to put the two $N/2$-point DFTs together. Note that a 2-point DFT is simply a sum and difference:

$$X[0] = x[0] + x[1]$$
$$X[1] = x[0] - x[1].$$

Hence, the starting conditions are $A_c(2) = 2$ and $M_c(2) = 0$. Then solving the recursive equation yields

$$A_c(N) = N \log_2 N \quad \text{complex additions.}$$

Similarly, one has a formula for complex multiplications:

$$M_c(N) = \frac{N}{2} \log_2 N - N + 1 \qquad \text{complex multiplications.}$$

A single complex multiplication can be performed with 4 real multiplications and 2 real additions. A single complex addition can be performed with 2 real additions. Therefore,

$$M_r(N) = 4 \cdot M_c(N)$$
$$A_r(N) = 2 \cdot M_c(N) + 2 \cdot A_c(N)$$

which gives

$$M_r(N) = 2\,N \log_2 N - 4N + 4$$
$$A_r(N) = 3N \log_2 N - 2N + 2$$

COMPLEXITY (2)

In fact, this number can be reduced by a more careful examination of the multipliers W_N^k (the twiddle factors). In particular, the numbers 1, -1, j, and $-j$ will be among the twiddle factors W_N^k, when k is a multiple of $N/4$ — and so those multiplications need not be performed. Taking this into account one can reduce the number of multiplications a bit more.

The advantage of the efficient algorithm for computing the DFT is a reduction from an arithmetic complexity of N^2 for direct calculation to a complexity of $N \log_2 N$. This is a fundamental improvement in the complexity, and historically it led to many new developments in signal processing that would not have otherwise been possible or practical. Due to its fundamental speed-up in calculating the DFT, the efficient algorithm for its computation is called the *Fast Fourier Transform* or FFT.

Consider an N-point signal $x[n]$ of even length. The derivation of the DIF radix-2 FFT begins by splitting the DFT coefficients $X[k]$ in to even- and odd- indexed values. The even values $X[2k]$ are given by:

$$\begin{aligned} X[2k] &= \sum_{n=0}^{N-1} x[n]\, W_N^{-2kn} \\ &= \sum_{n=0}^{N-1} x[n]\, W_{N/2}^{-kn}. \end{aligned}$$

Splitting this sum into the first $N/2$ and second $N/2$ terms gives

$$\begin{aligned} X[2k] &= \sum_{n=0}^{\frac{N}{2}-1} x[n]\, W_{N/2}^{-kn} + \sum_{n=\frac{N}{2}}^{N-1} x[n]\, W_{N/2}^{-kn} \\ &= \sum_{n=0}^{\frac{N}{2}-1} x[n]\, W_{N/2}^{-kn} + \sum_{n=0}^{\frac{N}{2}-1} x[n+\frac{N}{2}]\, W_{N/2}^{-k(n+\frac{N}{2})} \\ &= \sum_{n=0}^{\frac{N}{2}-1} x[n]\, W_{N/2}^{-kn} + \sum_{n=0}^{\frac{N}{2}-1} x[n+\frac{N}{2}]\, W_{N/2}^{-kn} \\ &= \sum_{n=0}^{\frac{N}{2}-1} \left(x[n] + x[n+\frac{N}{2}] \right) W_{N/2}^{-kn} \\ &= \mathrm{DFT}_{\frac{N}{2}} \left\{ x[n] + x[n+\frac{N}{2}] \right\}. \end{aligned}$$

That is, the even DFT values $X[2k]$ for $0 \leq 2\,k \leq N-1$ are given by the $\frac{N}{2}$-point DFT of the $\frac{N}{2}$-point signal $x[n] + x[n+N/2]$.

Similarly, the odd values $X[2k+1]$ are given by:

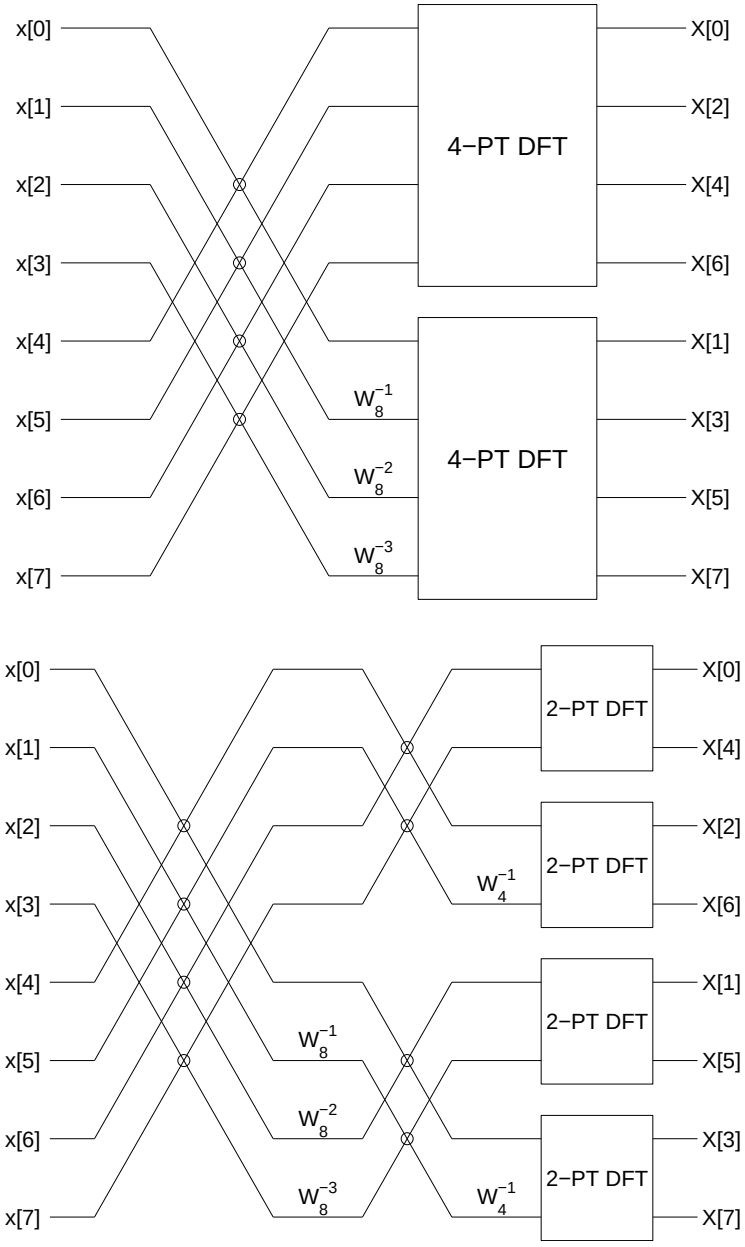
$$\begin{aligned} X[2k+1] &= \sum_{n=0}^{N-1} x[n]\, W_N^{-(2k+1)n} \\ &= \sum_{n=0}^{N-1} x[n]\, W_N^{-n}\, W_N^{-2kn} \\ &= \sum_{n=0}^{N-1} x[n]\, W_N^{-n}\, W_{N/2}^{-kn}. \end{aligned}$$

Splitting this sum into the first $N/2$ and second $N/2$ terms gives

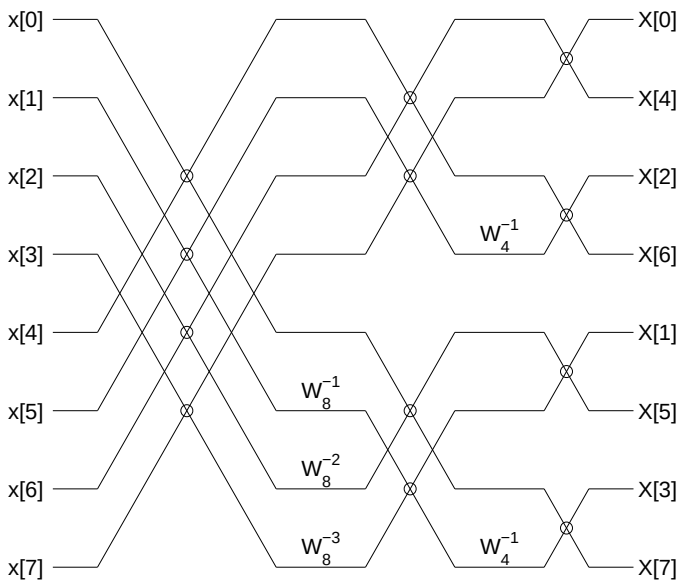
$$\begin{aligned} X[2k+1] &= \sum_{n=0}^{\frac{N}{2}-1} x[n]\, W_N^{-n}\, W_{N/2}^{-kn} + \sum_{n=\frac{N}{2}}^{N-1} x[n]\, W_N^{-n}\, W_{N/2}^{-kn} \\ &= \sum_{n=0}^{\frac{N}{2}-1} x[n]\, W_N^{-n}\, W_{N/2}^{-kn} + \sum_{n=0}^{\frac{N}{2}-1} x[n+\frac{N}{2}]\, W_N^{-(n+\frac{N}{2})}\, W_{N/2}^{-k(n+\frac{N}{2})} \\ &= \sum_{n=0}^{\frac{N}{2}-1} x[n]\, W_N^{-n}\, W_{N/2}^{-kn} - \sum_{n=0}^{\frac{N}{2}-1} x[n+\frac{N}{2}]\, W_N^{-n}\, W_{N/2}^{-kn} \\ &= \sum_{n=0}^{\frac{N}{2}-1} \left(x[n] - x[n+\frac{N}{2}] \right) W_N^{-n}\, W_{N/2}^{-kn} \\ &= \mathrm{DFT}_{\frac{N}{2}} \left\{ W_N^{-n} \left(x[n] - x[n+\frac{N}{2}] \right) \right\}. \end{aligned}$$

That is, the odd DFT values $X[2k+1]$ for $0 \le 2k+1 \le N-1$ are given by the $\frac{N}{2}$-point DFT of the $\frac{N}{2}$-point signal $W_N^{-n}\ (x[n] - x[n+N/2])$.

The DIF radix-2 FFT algorithm is illustrated by the



FULL DIF FLOWGRAPH



The DIT and DIF radix-2 FFT algorithms are very similar.

1. The DIT and DIF radix-2 FFT algorithms have the same complexity.

2. Both can be implemented *in-place*.

3. For the DIT FFT, $x[n]$ is in bit-reversed order, $X[k]$ is in normal order. For the DIF FFT, $X[k]$ is in bit-reversed order, $x[n]$ is in normal order.

The DIT FFT algorithm requires a bit-reversal *prior to* the in-place flowgraph, while the DIF FFT algorithm requires a bit-reversal *after* the in-place flowgraph.

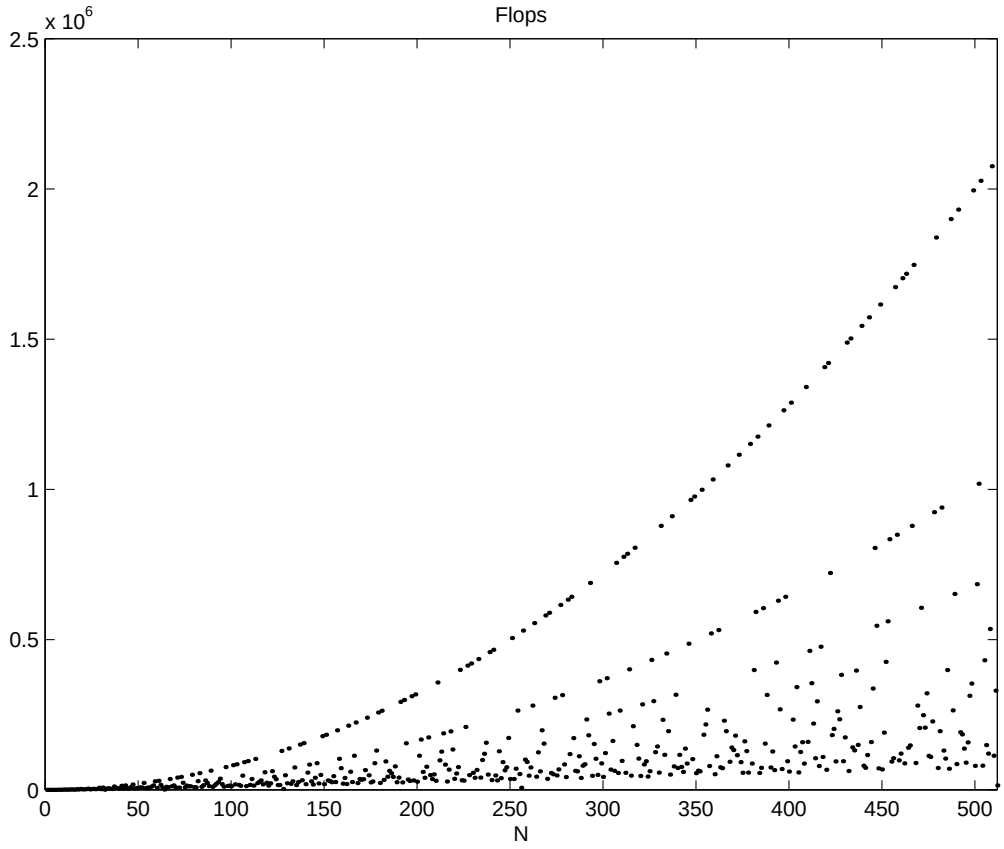
Other FFT algorithms

Other FFT algorithms include the Cooley-Tukey FFT which can be used whenever the length of the data vector can be factored as $N = N_1 \cdot N_2$.

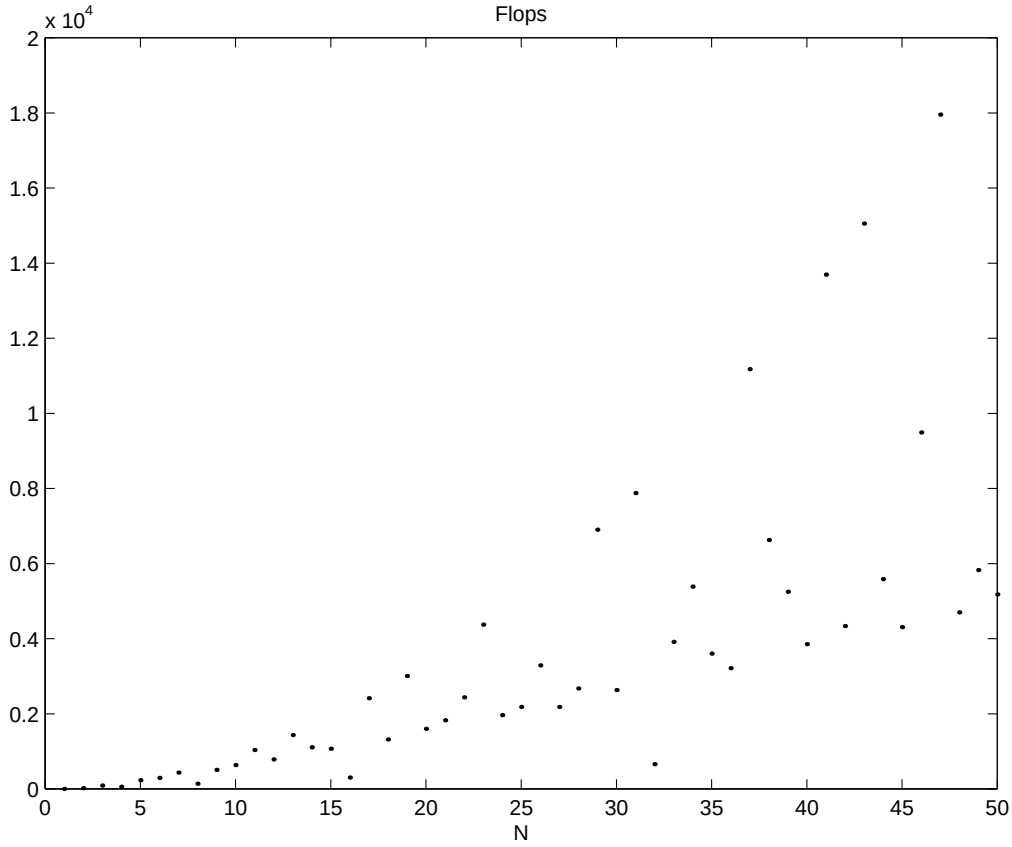
In addition, there is a class of FFT algorithms that apply only when the factors N_1 and N_2 are relatively prime (when their greatest common divisor is 1). Those algorithms, called *Prime Factor FFTs*, avoid the multiplications by twiddle factors. They require fewer multiplications, but are more complicated to program.

Operation Counts of the MATLAB `fft` Command

With early versions of MATLAB, we could measure the number of flops (floating point operations) of many MATLAB commands. (That ability is no longer available in current versions of MATLAB). Using an earlier version of MATLAB, we measured the flops used by the `fft` command for lengths $2 \leq N \leq 512$. We made a plot of the number of flops versus the length N using `plot(n,f,'.')` to prevent MATLAB from connecting the dots. The result is shown below:



Close up:



The graphs show the number of floating point operations (flops) incurred by the MATLAB `fft` command. This graph was done with version 5.2 of MATLAB. Version 6 will give different flop counts (improved flops counts for some values of N).

1. The lengths for which the `fft` command is slowest are the *prime* lengths. For these lengths, the `fft` command simply computes the DFT directly using no fast algorithm. For the direct computation of the DFT, we know the flops are proportional to N^2, and that is evident in the graph.

2. The second top line corresponds to lengths that are $N = 2 \cdot \text{prime}$. For these lengths the `fft` command decomposes

the DFT into two DFTs each of length $N/2$. Likewise for other composite lengths.

3. For powers of 2, the implementation of the `fft` command can use a pure radix-2 FFT algorithm, which is carefully programmed for maximum efficiency. The `fft` command is significantly more efficient for powers of 2 than for other lengths.

The following MATLAB code was used for this problem.

```
f = zeros(1,512);
for n = 1:512
   x = rand(1,n);
   flops(0);
   X = fft(x);
   f(n) = flops;
end

figure(1), clf
plot(1:512,f,'.')
xlabel('N')
title('Flops')
axis([0 512 0 2.5e6])
print -deps fftflops_a

figure(2), clf
plot(1:512,f,'.')
xlabel('N')
title('Flops')
axis([0 50 0 2.0e4])
print -deps fftflops_b
```