**Theory:**

* Direct implementation of DFT is not efficient.
* In direct computation, total number of complex additions are N(N − 1) and complex multiplications are N2.
* FFT is an algorithm to compute DFT in an efficient manner.
* Two common algorithms are

1. Decimation-in-time FFT
2. Decimation-in-frequency FFT
3. **Radix-2 DIT-FFT:**

* Radix 2 algorithm means N can be expressed as a power of 2, i.e., N = 2M
* Input sequence is broken in to two sequence of length N/2.
  + xe(n)=x(2n); n=0,1,….,(N/2)-1 🡪 even samples
  + xo(n)=x(2n+1); n=0,1,….,(N/2)-1 🡪 odd samples

Or,

So,

Or,

Where

Hence,

Or,

As we know that

Therefore,

So, three steps are used.

* Compute N/2-point DFT Xe[k] of xe[n] = even samples of x[n].
* Compute N/2-point DFT Xo[k] of xo[n] = odd samples of x[n].
* Compute G[k] = , k = 0, . . . , N/2 – 1
* Combine: X[k] = Xe[k] + G [k]

X[k + N/2] = Xe[k] − G [k] , k = 0, . . . , N/2 – 1

Butterfly:

G[k]

xo[n]

xe[n]

X[k+N/2]

X[k]

**-1**

So basically key idea is: splitting into two sequences (even/odd) each of half the length, take DFT of each, then combine, saves a factor of 2 in complex multiplies.

**Result:**

**We get following graph using while computing FFT using DIT:**

`

**2. Radix-2 DIF-FFT:**

* + Output sequence X(k) is divided into smaller and smaller subsequences.
  + Input sequence x(n) partitioned into two sequences each of length N/2
* x1(n) – first N/2 samples of x(n)
* x2(n) – last N/2 samples of x(n)
* The N - point DFT of x(n) can be written as:
* When k is even,
* This is N/2-point DFT of the N/2-point sequence obtained by adding the first half and the last half of the input sequence.
* When k is odd,
* This is the N/2- point DFT of the sequence obtained by subtracting the second half of the input sequence from the first half and multiplying the resulting sequence by
* Where
* So the even and odd samples of the DFT can be obtained from the N/2-point DFTs of f(n) and g(n) respectively:
* .
* .

=

**Result:**

**We get following graph using while computing FFT using DIF:**

****

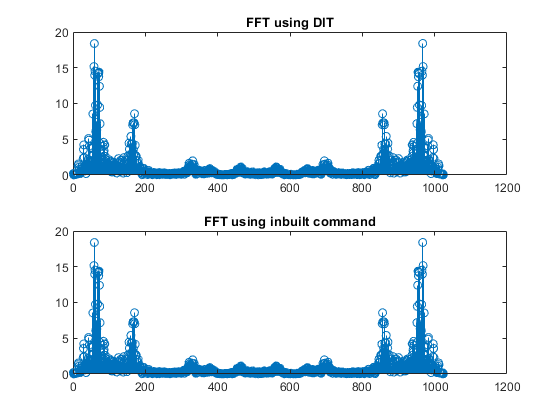
**Conclusion:**

* For DIT, the input is bit reversed while the output is natural order. For DIF, the input is in natural order while the output is bit reversed. 2.
* The DIF butterfly is slightly different from the DIT.
* In DIF, the complex complex multiplication takes place after the add-subtract operation.
* Both algorithms require N log2N operations to compute the DFT. Both algorithms can be done in-place and both need to perform bit reversal at some place during the computation.

**APPENDIX**

# FFT Using DIT

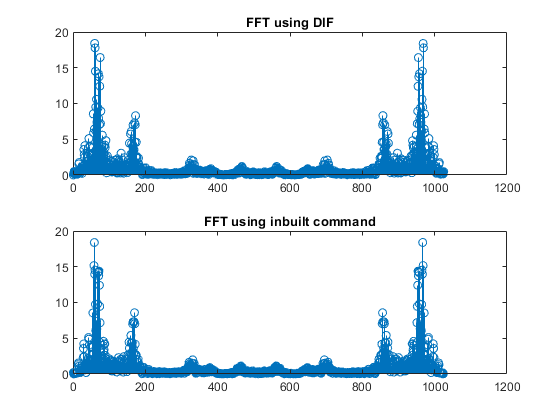
clear all  
close all  
clc  
warning off  
fs=1000;  
[readmusic,fs]=audioread('1610958375116.wav');  
N=1024;  
  
%Taking first 1024 samples  
for z = 0:1:(N-1)  
 a(z+1)=readmusic(z+1);  
end  
  
%finding even and odd parts  
for zz = 0:1:(N/2)-1  
 a\_even(zz+1) = a(2\*zz+1);  
 a\_odd(zz+1) = a((2\*zz)+2);  
end  
  
%For twiddle factor type 1 length=(N/2)  
for k= 0:1:(N/2)-1  
 for n= 0:1:(N/2)-1  
 p=exp(-sqrt(-1)\*pi\*2\*2\*k\*n/N);  
 twiddle\_fact\_matrix(k+1, n+1)=p;  
 end  
end  
%N/2-point DFT of even samples  
fft\_ev = twiddle\_fact\_matrix\*a\_even';  
%N/2-point DFT of even samples  
fft\_odd = twiddle\_fact\_matrix\*a\_odd';  
  
%Combine  
for k=0:1:(N/2)-1;  
 fft\_simm(k+1) = fft\_ev(k+1) + (exp(-sqrt(-1)\*2\*pi\*k/N)\*fft\_odd(k+1));  
 fft\_simm((N/2)+k+1) = fft\_ev(k+1) - (exp(-sqrt(-1)\*2\*pi\*k/N)\*fft\_odd(k+1));  
end  
  
subplot(211);  
stem(abs(fft\_simm));  
title('FFT using DIT')  
fft\_inbuilt=fft(a);  
subplot(212);  
stem(abs(fft\_inbuilt));  
title('FFT using inbuilt command')



[*Published with MATLAB® R2020b*](https://www.mathworks.com/products/matlab)

# FFT Using DIF

clear all  
close all  
clc  
warning off  
fs=1000;  
[readmusic,fs]=audioread('1610958375116.wav');  
N=1024;  
  
%Taking first 1024 samples  
for z = 0:1:(N-1)  
 a(z+1)=readmusic(z+1);  
end  
  
%Input sequence x(n) partitioned into two sequences each of length N/2  
% 0 to 511 as x1[n] and 512 to 1023 as x2[n]  
for zz = 0:1:(N/2)-1  
 x1(zz+1) = a(zz+1);  
 x2(zz+1) = a((N/2)+zz+1);  
end  
  
  
%For twiddle factor type 1 length=(N/2)  
for k= 0:1:(N/2)-1  
 for n= 0:1:(N/2)-1  
 p=exp(-sqrt(-1)\*pi\*2\*2\*k\*n/N);  
 twiddle\_fact\_matrix(k+1, n+1)=p;  
 end  
end  
  
f=x1+x2;  
g=x1-x2;   
  
%N/2-point even samples DFT  
fft\_even\_2k = twiddle\_fact\_matrix\*f';  
%N/2-point odd samples DFT  
fft\_odd\_2kplus1 = twiddle\_fact\_matrix\*g';  
  
%Combine  
for k=1:1:(N/2)  
 fft\_simm(2\*k) = fft\_even\_2k(k);  
 fft\_simm((2\*k)+1) = fft\_odd\_2kplus1(k)\*exp(-sqrt(-1)\*2\*pi\*k/N);  
end  
  
subplot(211);  
stem(abs(fft\_simm));  
title('FFT using DIF')  
fft\_inbuilt=fft(a);  
subplot(212);  
stem(abs(fft\_inbuilt));  
title('FFT using inbuilt command')



[*Published with MATLAB® R2020b*](https://www.mathworks.com/products/matlab)