# Assignment - FFT IMPLEMENTATION

BUEREDDY VARUNI - EE18BTECH11005

Download all codes from

```
https://github.com/varunireddy/EE3025_IDP/tree/
    main/fft_implementation/codes
```

and latex-tikz codes from

```
https://github.com/varunireddy/EE3025_IDP/tree/
    main/fft_implementation
```

## 1 Problem

FFT implementation algorithm using recursive approach.

## 2 Method - Recursive approach

We compute DFT using the following equation.,

$$X(k) = \sum_{n=0}^{N-1} x[n] W_N^{kn} \tag{2.0.1}$$

$$\text{where.,} W_N = e^{\frac{-2\pi i}{N}} \tag{2.0.2}$$

$$X(k) = \sum_{n=even} x[n] W_N^{kn} + \sum_{n=odd} x[n] W_N^{kn} \tag{2.0.3}$$

Let n = 2r in first term and n = 2r+1 in the second term.

$$X(k) = \sum_{r=0}^{\frac{N}{2}-1} x[2r] W_N^{2kr} + \sum_{r=0}^{\frac{N}{2}-1} x[2r-1] W_N^{k(2r-1)} \tag{2.0.4}$$

$$\text{Let, } x[2r] = e[r],\ x[2r-1] = o[r] \tag{2.0.5}$$

$$\tag{2.0.6}$$

We know from the exponential property of complex numbers,

$$W_{\frac{N}{2}} = W_N^2 \tag{2.0.7}$$

The equation.2.0.4 can be rewritten as the following.,

$$X(k) = \sum_{r=0}^{\frac{N}{2}-1} e[r] W_{N/2}^{kr} + W_N^k \sum_{r=0}^{\frac{N}{2}-1} o[r] W_{N/2}^{kr} \tag{2.0.8}$$

We can observe that the first term is the N/2 point DFT of even indices of the signal x[n] and the second term is the N/2 point DFT of odd indices of the signal x[n]. Therefore, the equation ∀ k in [0,N/2) can be written as,

$$X(k) = E(k) + W_N^k O(k) \tag{2.0.9}$$

Where E(k) is the dft of the even indices and O(k) is the dft of the odd indices of x[n]. For ∀ k+N/2 in [N/2,N) the equation can be written as,

$$X(k+N/2) = E(k+N/2) + W_N^{k+N/2} O(k+N/2) \tag{2.0.10}$$

$$X(k+N/2) = E(k) - W_N^k O(k) \tag{2.0.11}$$

If $F_N$ is the N point DFT matrix and $F_{N/2}$ is the N/2 point DFT matrix.

$$X(k) = F_N x[n] \tag{2.0.12}$$

$$E(k) = F_{N/2} e[n] \tag{2.0.13}$$

$$O(k) = F_{N/2} o[n] \tag{2.0.14}$$

From the equations.2.0.9 and 2.0.11,
for k in [0,N/2)..,

$$F_N x[n] = F_{N/2} e[n] + F_{N/2} D_{N/2} o[n] \tag{2.0.15}$$

for k in [N/2,N)..,

$$F_N x[n] = F_{N/2} e[n] - F_{N/2} D_{N/2} o[n] \tag{2.0.16}$$

where $D_N$ is the diagonal matrix with diagonal values $[1, W_N^1, W_N^2, W_N^3 ..., W_N^{N-1}]$.
Combining the above two equations,

$$F_N x[n] = \begin{bmatrix} I_{N/2} & D_{N/2} \\ I_{N/2} & -D_{N/2} \end{bmatrix} \begin{bmatrix} F_{N/2} & 0 \\ 0 & F_{N/2} \end{bmatrix} \begin{bmatrix} e[n] \\ o[n] \end{bmatrix} \tag{2.0.17}$$

Let P be the transformation matrix., which transforms x[n] into [e[n] o[n]].

$$\begin{bmatrix} e[n] \\ o[n] \end{bmatrix} = [P]_{N\times N} x[n] \tag{2.0.18}$$

where.,

$$P = \begin{bmatrix} 1 & 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 1 & 0 & \dots & 0 \\ \vdots \\ 0 & 1 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & 1 & \dots & 0 \\ \vdots \\ 0 & 0 & 0 & 0 & \dots & 1 \end{bmatrix} \tag{2.0.19}$$

From equations.2.0.17 and 2.0.18 we get.,

$$F_N = \begin{bmatrix} I_{N/2} & D_{N/2} \\ I_{N/2} & -D_{N/2} \end{bmatrix} \begin{bmatrix} F_{N/2} & 0 \\ 0 & F_{N/2} \end{bmatrix} P \tag{2.0.20}$$

Thus, we can now compute $F_N$ from $F_{N/2}$, $F_{N/2}$ from $F_{N/4}$. This is the recursive approach. N=1 is the base case. When N = 1, FFT(x) = x.

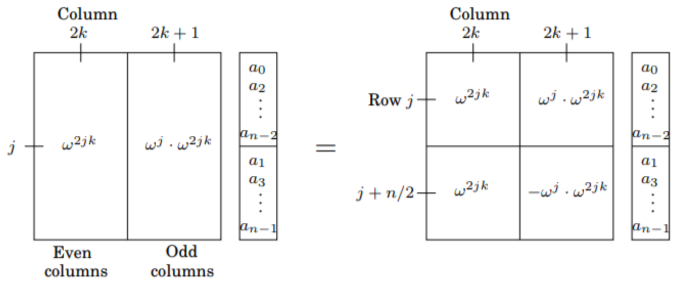
Column $2k$
$2k+1$
$j$
$\omega^{2jk}$
$\omega^j \cdot \omega^{2jk}$
$a_0$ $a_2$ $\vdots$ $a_{n-2}$
$a_1$ $a_3$ $\vdots$ $a_{n-1}$
Even columns
Odd columns
=
Column $2k$
$2k+1$
Row $j$
$j+n/2$
$\omega^{2jk}$
$\omega^j \cdot \omega^{2jk}$
$\omega^{2jk}$
$-\omega^j \cdot \omega^{2jk}$

Fig. 0: Recursive sub block approach

## 3 Pseudo Code

**Algorithm 1** fft(x)

```
Require: $N > 0$
Ensure: $N = 2^n$
  N ← length(x)
  if $N = 1$ then
    Return x
  else
    E ← fft(x[0],x[2]...,x[N-2])
    O ← fft(x[1],x[3]...,x[N-1])
    while $k \neq N/2 - 1$ do
      X[k] ← E[k] + $e^{2\pi jk/N}$ O[k]
      X[k+N/2] ← E[k] - $e^{2\pi jk/N}$ O[k]
    end while
    Return X
  end if
```

## 4 FFT Implementation in Python

Let x[n] = $\left\{ \underset{\uparrow}{1}, 2, 3, 4, 4, 3, 2, 1 \right\}$

$N = 8 = 2^3$. We need to compute the 8-point DFT using the above mentioned recursive algorithm in Python.

$$X = F_8 x[n] \tag{4.0.1}$$

$$F_8 = \begin{bmatrix} I_4 & D_4 \\ I_4 & -D_4 \end{bmatrix} \begin{bmatrix} F_4 & 0 \\ 0 & F_4 \end{bmatrix} P_8 \tag{4.0.2}$$

$$D_4 = diag[0, W_4^1, W_4^2, W_4^3] \tag{4.0.3}$$

$$P_8 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.0.4}$$

Our goal is to compute X, for which we need to find $F_8$, which can be computed from $F_4$.

$$F_4 = \begin{bmatrix} I_2 & D_2 \\ I_2 & -D_2 \end{bmatrix} \begin{bmatrix} F_2 & 0 \\ 0 & F_2 \end{bmatrix} P_4 \tag{4.0.5}$$

$$F_2 = \begin{bmatrix} 1 & D_1 \\ 1 & -D_1 \end{bmatrix} \begin{bmatrix} F_1 & 0 \\ 0 & F_1 \end{bmatrix} P_2 \tag{4.0.6}$$

$$F_1 = 1 \tag{4.0.7}$$

Thus, we recursively compute 8 point DFT from two 4 point DFT's and 4 point DFT's from two 2 point DFT's. The following python code is the implementation of the above algorithm.

```
https://github.com/varunireddy/EE3025_IDP/blob/
    main/fft_implementation/codes/
    ee18btech11005.py
```

The above code generates the Figure.0 FFT implementation in python.

## 5 FFT Implementation in C

We can use the divide and conquer approach in C. The following C code generates the .dat binary file which contains the real and imaginary parts of the fft.

```
https://github.com/varunireddy/EE3025_IDP/blob/
    main/fft_implementation/codes/
    ee18btech11005.c
```

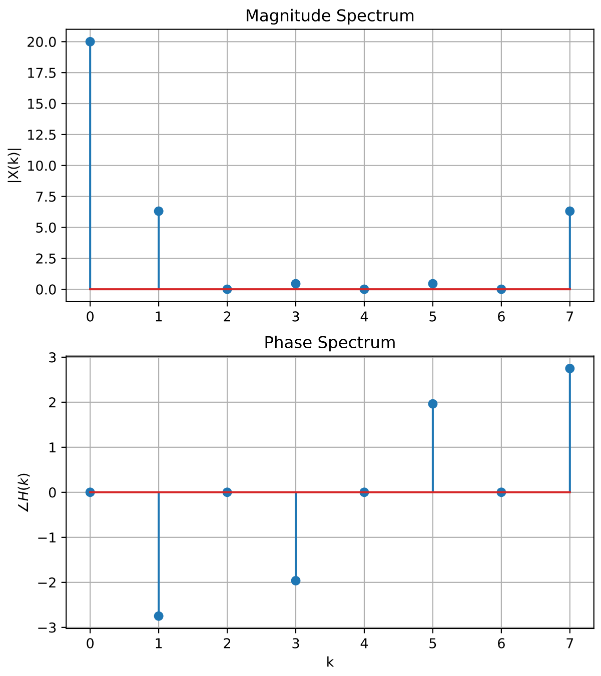

Fig. 0: FFT implementation in python

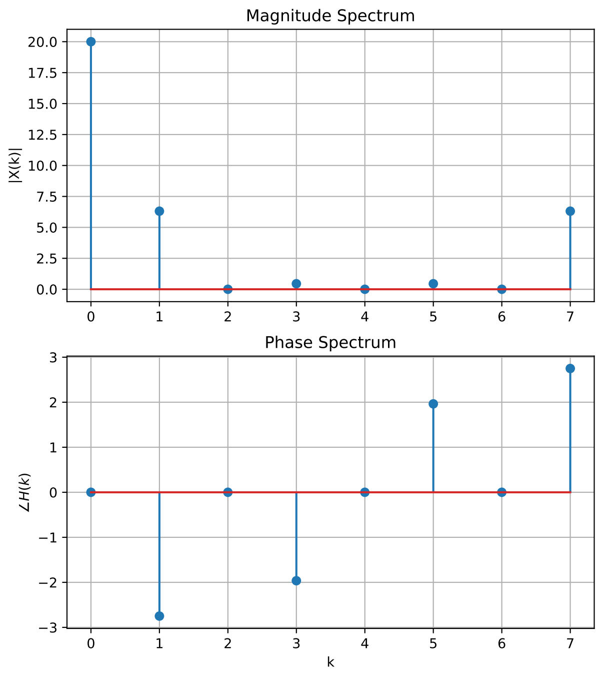

Fig. 0: FFT implementation in C

The following python code generates the plots from the .dat file generated by running the above C code

```
https://github.com/varunireddy/EE3025_IDP/blob/
    main/fft_implementation/codes/read_dat_file.
    py
```

Figure.0 is the implementation of the fft algorithm in C and plotting the spectrum in python.

## 6 Time Complexity

The traditional DFT algorithm has two nested for loops, So the time complexity is O($N^2$). But we can notice that the FFT algorithm with N inputs has $\log_2 N$ levels, each with N nodes, for a total of N$\log_2 N$ operations that is one N point FFT can be broken down into two N/2 point fft's.

$$T(n) = 2T(n/2) + O(n) \tag{6.0.1}$$

Thus, the time complexity of the FFT algorithm using recursive subblock approach is O(N$\log_2 N$) whereas for a DFT algorithm time complexity is O($N^2$).
The following code compares the computation times for a DFT algorithm and FFT algorithm.

```
https://github.com/varunireddy/EE3025_IDP/blob/
    main/fft_implementation/codes/comparetime.
    py
```

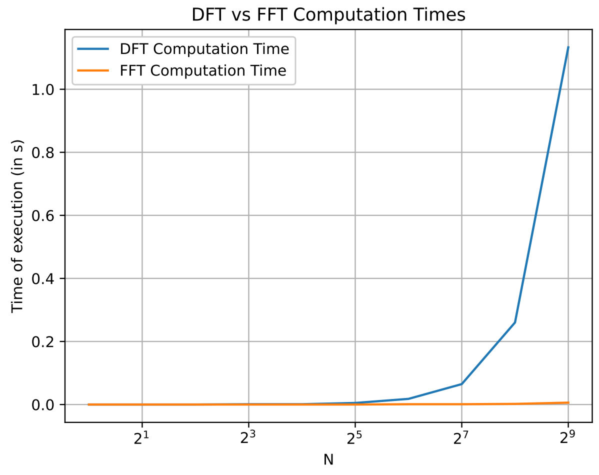

Fig. 0: DFT vs FFT

## 7 Conclusions

The Fast Fourier Transform (FFT) is an implementation of the DFT which produces almost the same results as the DFT, but it is incredibly more efficient and much faster which often reduces the computation time significantly. Therefore, FFT is a computational algorithm used for fast and efficient computation of the DFT.