Experiment No 07: Fast Fourier Transform

### **Name: Rebecca Dias Roll No.: 19 Class: BE CMPN A PID: 182027**

**Aim:** To perform the FFT of a discrete time signal in Python

# Theory:

A fast Fourier transform (FFT) is an algorithm that computes the discrete Fourier transform (DFT) in Nlog(N) time instead of N2 time by using the symmetry and periodicity properties of DFT and by reusing partial results. This method can save a huge amount of processingtime.

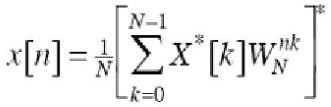
# Decimation-in-Time (DIT) Algorithm

This algorithm is also known as radix-2 DIT-FFT algorithm. As the name implies, the number of output points N can be expressed as a power of 2.

N= 2m where m is an integer.

# Inverse FFT algorithm

Inverse Fast Fourier transform (IFFT) is an algorithm to undo the process of DFT. It is also known as backward Fourier transform. It converts a signal of the frequency domain to time domain signal. The Inverse DFT of an N-point sequence X(k) is



# How to compute IDFT using FFT algorithm

1. The sequence is padded with zero to the right because the radix-2 FFT requires the sample point number as a power of 2.
2. The conjugate X\*(k) is computed and fed to flowgraph in bit reversed order

The output of flow graph is conjugate Nx\*(n) in sequence order. Divide by N and take complex conjugate to obtain x(n)

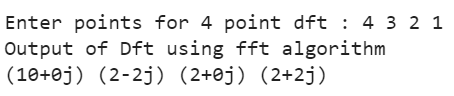
# Programming Exercises in Python

1. Compute DFT of x(n) = {4,3,2,1} using FFT algorithm
2. Compute DFT of x(n) = {5,6,7,8,1,2,3,4} using FFT algorithm

3. Compute IDFT of [15 -2.5+3.44j -2.5+0.81j -2.5-0.81j -2.5-3.44j]

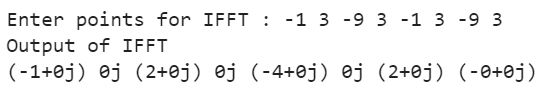
Code:

|  |
| --- |
| import numpy as np  def FFT(x):      N = len(x)      if N == 1:          return x      else:          X\_even = FFT(x[::2])          X\_odd = FFT(x[1::2])          factor = np.exp(-2j\*np.pi\*np.arange(N)/ N)          X = np.concatenate([X\_even+factor[:int(N/2)]\*X\_odd,X\_even+factor[int(N/2):]\*X\_odd])          return X  def IFFT(x):      N = len(x)      if N == 1:          return x      else:          X\_even = FFT(x[::2])          X\_odd = FFT(x[1::2])          factor = np.exp(2j\*np.pi\*np.arange(N)/ N)          X = (np.concatenate([X\_even+factor[:int(N/2)]\*X\_odd,X\_even+factor[int(N/2):]\*X\_odd]))/N          return X  d=list(map(int,input("Enter points for 4 point dft : ").split()))  x = np.array(d)  Y=FFT(x)  print("Output of Dft using fft algorithm ")  for each in Y:    print(round(each.real, 2) + round(each.imag, 2) \* 1j,end=" ") |



|  |
| --- |
| d=list(map(int,input("Enter points for 8 point dft : ").split()))  x = np.array(d)  Y=FFT(x)  print("Output of Dft using fft algorithm ")  for each in Y:    print(round(each.real, 2) + round(each.imag, 2) \* 1j,end=" ") |
|  |

|  |
| --- |
| d=list(map(complex,input("Enter points for IFFT : ").split()))  x = np.array(d)  Y=IFFT(x)  print("Output of IFFT")  for each in Y:    print(round(each.real, 2) + round(each.imag, 2) \* 1j,end=" ") |



|  |
| --- |
| import numpy as np  import scipy  np.set\_printoptions(precision=2, suppress=True)  y = np.array(list(map(complex,input("Enter points for IFFT : ").split())))  x = scipy.fft.ifft(y)  print('Output of IFFT: ')  for each in x:    print(round(each.real, 2) + round(each.imag, 2) \* 1j,end=" ") |
|  |

**Conclusion:** Studied the FFT transform of a discrete time signal and implemented it using python language , the output was obtained accurately. We also studied about computing IDFT using FFT algorithm.