EXPERIMENT NO 3

NAME: Vishal Shashikant Salvi **UID: 2019230069**
CLASS: TE COMPS **BATCH: C**

Aim: To find Discrete Fourier Transform and Inverse Discrete Fourier Transform of given digital signal.

Theory:

Fast-Fourier transform

- ✓ Fast Fourier transform (FFT) computes the discrete Fourier transform (DFT) and its inverse. The FFT algorithm is used to convert a digital signal (*x*) with length (*N*) from the time domain into a signal in the frequency domain (*X*), since the amplitude of vibration is recorded on the basis of its evolution versus the frequency at that the signal appears.
- ✓ The frequency spectrum vector is divided into different ranges of frequencies to automate the selection process of the sensitive frequencies to the fault under investigation. The average of each range is then taken as a sensory feature for the system.
- ✓ FFT is used in the wind turbine gearbox data analysis, for example for fault detection and evaluation of bearings, shafts, and gears. If damage initiates in a bearing, the shape of the vibration distribution deviates to a gauss-shaped curve. Frequencies are conditioned by the rotational shaft velocity, as well as by shape and size of faults in bearings. Originally, its purpose was served by band selective filters. The advantage of frequency domain analysis over time-domain analysis is its ability to identify and isolate certain frequency component of interest.

The fast Fourier transform (FFT), as the name implies, is a fast version of the DFT. The FFT exploits the fact that the straightforward approach to computing the Fourier transform performs the many of the exact same multiplications repeatedly.

The FFT algorithm organizes these redundant computations in a very efficient manner by taking advantage of the algebraic properties in the Fourier matrix. Specifically, the FFT makes use of periodicities in the sines that are multiplied to perform the calculation. Basically, the FFT takes the Fourier matrix and factorizes it into several sparse matrices. These sparse matrices have many entries that are equal to zero. Using sparse matrices reduces the total amount of calculations required.

The FFT eliminates almost all of these redundant calculations, and this saves a significant amount of calculation, which makes the Fourier transform much more practical to use in many applications today.

DFT

The discrete Fourier transform (DFT) is the primary transform used for numerical computation in digital signal processing. It is very widely used for spectrum analysis, fast convolution, and many other applications. The DFT transforms NN discrete-time samples to the same number of discrete frequency samples, and is defined as

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

The DFT is widely used in part because it can be computed very efficiently using fast Fourier transform (FFT) algorithms.

Applications of the DFT

- The discrete Fourier transform (DFT) is one of the most important tools in digital signal processing. This chapter discusses three common ways it is used.
- First, the DFT can calculate a signal's *frequency spectrum*. This is a direct examination of information encoded in the frequency, phase, and amplitude of the component sinusoids. For example, human speech and hearing use signals with this type of encoding.
- Second, the DFT can find a system's frequency response from the system's impulse response, and vice versa. This allows systems to be analyzed in the *frequency domain*, just as convolution allows systems to be analyzed in the *time domain*.
- Third, the DFT can be used as an intermediate step in more elaborate signal processing techniques.
- The classic example of this is *FFT convolution*, an algorithm for convolving signals that is hundreds of times faster than conventional methods.

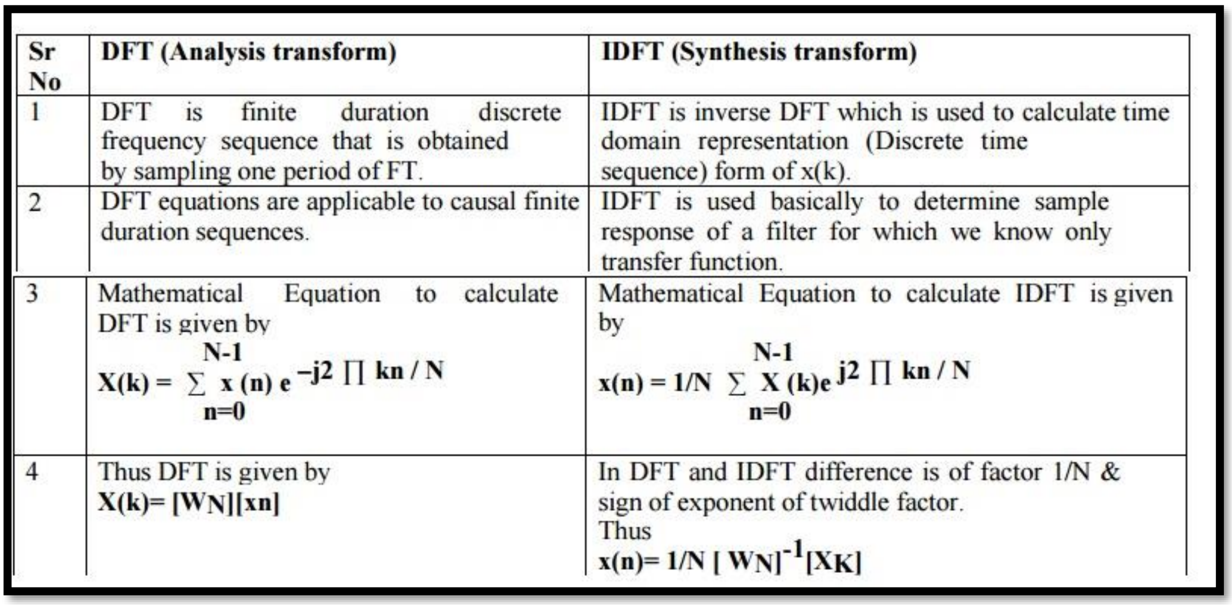
IDFT

The inverse DFT (IDFT) transforms NN discrete-frequency samples to the same number of discrete-time samples. The IDFT has a form very similar to the DFT,

$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k) e^{i \frac{2\pi nk}{N}}$$

and can thus also be computed efficiently using FFTs.

Difference:



Sr No	DFT (Analysis transform)	IDFT (Synthesis transform)
1	DFT is finite duration discrete frequency sequence that is obtained by sampling one period of FT.	IDFT is inverse DFT which is used to calculate time domain representation (Discrete time sequence) form of x(k).
2	DFT equations are applicable to causal finite duration sequences.	IDFT is used basically to determine sample response of a filter for which we know only transfer function.
3	Mathematical Equation to calculate DFT is given by $X(k) = \sum_{n=0}^{N-1} x(n)\, e^{-j2 \prod kn / N}$	Mathematical Equation to calculate IDFT is given by $x(n) = 1/N \sum_{n=0}^{N-1} X(k) e^{j2 \prod kn / N}$
4	Thus DFT is given by **$X(k)= [W_N][xn]$**	In DFT and IDFT difference is of factor 1/N & sign of exponent of twiddle factor. Thus **$x(n)= 1/N\, [W_N]^{-1}[X_K]$**

Algorithm:

Step I : Give input sequence .

Step II : Find the length of the input sequence using length command.

Step III : Find DFT and IDFT

Step IV : Display the results at each stage of deposition DFT

Code (DFT):

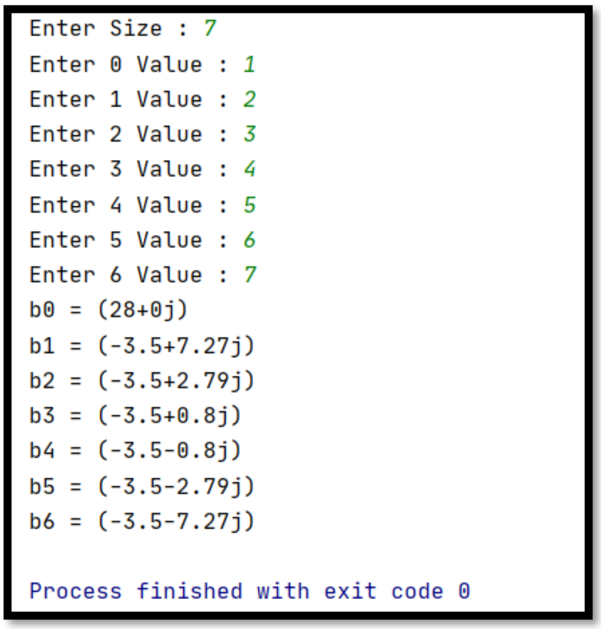
```
import math
import numpy as np
from cmath import sqrt
n = int(input("Enter Size : "))

a = [0]*n
for i in range(0,n):
    a[i] = int(input(f"Enter {i} Value : "))

b = [0]*n
j = sqrt(-1)
for k in range(0,n):
    for l in range(0,n):
        b[k] += a[l]*(np.exp((-np.pi *j*l* k*2)/n))

for i in range(0,n):
    print(f"b{i} = {np.round(b[i],2)}")
```

Output:



```
Enter Size : 7
Enter 0 Value : 1
Enter 1 Value : 2
Enter 2 Value : 3
Enter 3 Value : 4
Enter 4 Value : 5
Enter 5 Value : 6
Enter 6 Value : 7
b0 = (28+0j)
b1 = (-3.5+7.27j)
b2 = (-3.5+2.79j)
b3 = (-3.5+0.8j)
b4 = (-3.5-0.8j)
b5 = (-3.5-2.79j)
b6 = (-3.5-7.27j)

Process finished with exit code 0
```

Code (IDFT):

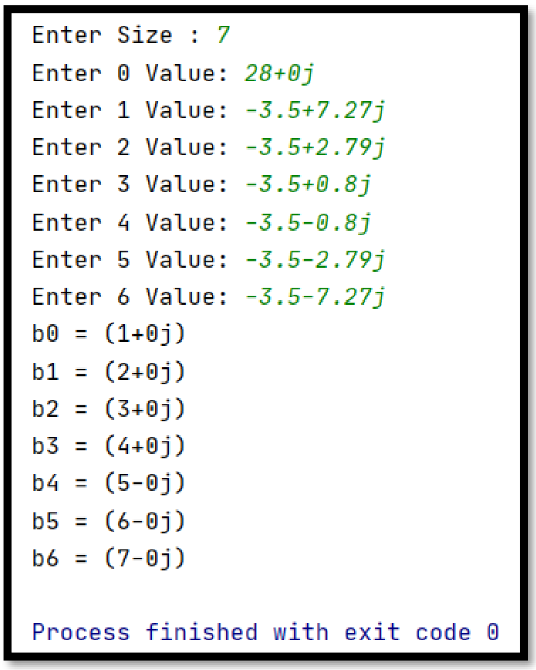
```
import math
import numpy as np
from cmath import sqrt
n = int(input("Enter Size : "))

a = [0]*n
for i in range(0,n):
    a[i] = eval(input(f"Enter {i} Value: "))

b = [0]*n
j = sqrt(-1)
for k in range(n):
    for l in range(n):
        b[l] += (1/n)*(a[k]*(np.exp((np.pi *j*l* k*2)/n)))

for i in range(0,n):
    print(f"b{i} = {np.round(b[i],2)}")
```

Output:



```
Enter Size : 7
Enter 0 Value: 28+0j
Enter 1 Value: -3.5+7.27j
Enter 2 Value: -3.5+2.79j
Enter 3 Value: -3.5+0.8j
Enter 4 Value: -3.5-0.8j
Enter 5 Value: -3.5-2.79j
Enter 6 Value: -3.5-7.27j
b0 = (1+0j)
b1 = (2+0j)
b2 = (3+0j)
b3 = (4+0j)
b4 = (5-0j)
b5 = (6-0j)
b6 = (7-0j)

Process finished with exit code 0
```

Conclusion:

Hence, I learn how to implement Fast Fourier Transform and Inverse Fast Fourier Transform using Python.

Reference:

1) **https://www.tutorialspoint.com/digital_signal_processing/dsp_in_place_computation.htm**
2) **https://cnx.org/contents/qAa9OhlP@2.44:AePPYjup@5/DFT-Definition-and-Properties**
3) **https://www.sciencedirect.com/topics/engineering/fast-fourier-transform**