2019

Internship Report

INDIAN INSTITUTE OF TECHNOLOGY GUWAHATI

Duration: 1st June -12th July, 2019

Report on

Frequency Domain Analysis and Discrete Wavelet Transform of Speech Signals

Submitted By:

Shibani Das

B. Tech (6th Semester)

Computer Science and Engineering

Tezpur University

Mentored By:

Prof. Pradip Kumar Das

Professor

Computer Science and Engineering

IIT-Guwahati

CERTIFICATE

This is to certify that the work contained in the project entitled "**Frequency Domain Analysis and Discrete Wavelet Transform of Speech Signals**" is a bonafide work of **Shibani Das**, carried out in the Department of Computer Science and Engineering, Indian Institute of Technology, Guwahati under my supervision and that has not been submitted elsewhere for a degree.

Supervisor: Prof. Pradip Kumar Das,

Professor

12 July, 2019

Department of Computer Science and Engineering

Indian Institute of Technology Guwahati, ASSAM.

CONTENTS

Page No.

ACKNOWLEDGEMENTS

I would like to express a deep sense of gratitude to my supervisor and mentor, **Prof. Pradip Kumar Das**, for giving me the opportunity to do an internship under his guidance. During the project, I got a chance to improve my practical skills beyond the limits of laboratories. I learned a lot of concepts of Frequency Domain Analysis of Speech Signals and the tasks were really helpful for me.

I would like to express my sincere thanks to my senior PhD Scholar P. Bhagath and my friends who helped me directly or indirectly during the accomplishment of my tasks.

I have no words to express my sincere gratitude to my parents for their immense support.

Finally, I also wish to thank everyone in the Computer Science and Engineering Department who created such a lively and friendly atmosphere that it was always exciting to go to the department.

ABSTRACT

Frequency domain method has many outstanding advantages compared with the classical time domain method. Fourier Transform converts a signal in time domain to frequency domain. Fourier transform is used to realize the filtering, modulation and sampling of the signal, which is the most important application of Fourier transform in signal processing. This report elaborates the description of Convolution method, Discrete Time Fourier Transform, Discrete Fourier Transform and Fast Fourier Transform methods. Also, it summarises the Python implementation of the above mentioned methods and Graphical Analysis of the processing of some of the input vowel signals. Discrete Wavelet Transform helps to analyse signals by using filters. A study of discrete wavelet transform is also done. This report shows analysis of recorded vowel signals of my own voice taking discrete as well as overlapped window frames.

Keywords : DTFT, DFT, FFT, DWT, fourier transform

Chapter 1

INTRODUCTION

1.1 Frequency Domain Analysis of Signals :

Frequency of a speech signal refers to the “PITCH” of the sound. The frequency domain refers to the analysis of signals with respect to time. A frequency-domain graph shows how much of the signal lies within each given frequency band over a range of frequencies.

1.2 Advantages of using frequency domain analysis :

One of the main reasons for using a frequency-domain representation of a problem is to simplify the mathematical analysis. For mathematical systems governed by linear differential equations, a very important class of systems with many real-world applications, converting the description of the system from the time domain to a frequency domain converts the differential equations to algebraic equations, which are much easier to solve.

An example of a field in which frequency-domain analysis gives a better understanding than time domain is music; the theory of operation of musical instruments and the musical notation used to record and discuss pieces of music is implicitly based on the breaking down of complex sounds into their separate component frequencies (musical notes).

1.3 Convolution :

In mathematics (in particular, functional analysis) **convolution** is a mathematical operation on two functions (*f* and *g*) to produce a third function that expresses how the shape of one is modified by the other. The term *convolution* refers to both the result function and to the process of computing it. It is defined as the integral of the product of the two functions after one is reversed and shifted.

Convolution in the time domain corresponds to ordinary multiplication in the frequency domain.

1.4 Fourier Transform :

A given function or signal can be converted between the time and frequency domains with a pair of mathematical operators called transforms. An example is the Fourier transform, which converts a time function into a sum or integral of sine waves of different frequencies, each of which represents a frequency component. The "spectrum" of frequency components is the frequency-domain representation.

of the signal. The inverse Fourier transform converts the frequency-domain function back to the time function.

1.4.1 Discrete Time Fourier Transform :

The DTFT is often used to analyze samples of a continuous function. The term *discrete-time* refers to the fact that the transform operates on discrete data, often samples whose interval has units of time. From uniformly spaced samples it produces a function of frequency that is a periodic summation of the continuous Fourier transform of the original continuous function.
Under certain theoretical conditions, described by the sampling theorem, the original continuous function can be recovered perfectly from the DTFT and thus from the original discrete samples.

1.4.2 Discrete Fourier Transform :

In mathematics, the **discrete Fourier transform** (**DFT**) converts a finite sequence of equally-spaced samples of a function into a same-length sequence of equally-spaced samples of the discrete-time Fourier transform (DTFT), which is a complex-valued function of frequency.

The DTFT itself is a continuous function of frequency, but discrete samples of it can be readily calculated via the discrete Fourier transform (DFT) , which is by far the most common method of modern Fourier analysis.

1.4.3 Fast Fourier Transform :

A **fast Fourier transform** (**FFT**) is an algorithm that computes the discrete Fourier transform (DFT) of a sequence, or its inverse (IDFT). It is basically an algorithm to solve DFT in a faster manner and is based on the idea of "Divide and Conquer"

This operation is useful in many fields, but computing it directly from the definition is often too slow to be practical. An FFT rapidly computes such transformations by factorizing the DFT matrix into a product of sparse (mostly zero) factors.

1.4.4 Discrete Wavelet Transform :

In wavelet analysis, the Discrete Wavelet Transform (DWT) decomposes a signal into a set of mutually orthogonal wavelet basis functions. These functions differ from sinusoidal basis functions in that they are spatially localized – that is, nonzero over only part of the total signal length. Furthermore, wavelet functions are

dilated, translated and scaled versions of a a common function φ, known as the mother wavelet.

DWT is invertible, like Discrete Fourier Transform.

Chapter 2

CONVOLUTION

2.1 Introduction :

Convolution is a mathematical operation on two functions (f and g) to produce a third function that expresses how the shape of one is modified by the other. The term Convolution refers to both the result function and to the process computing it.

2.2 Definition :

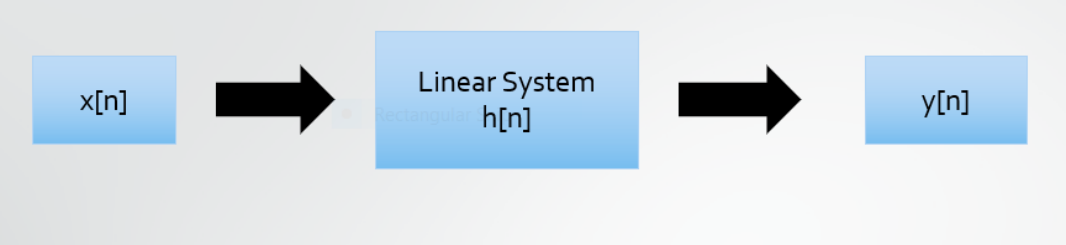


Fig 1 : Impulse response of a linear system

y[n] = x[n] * h[n] (for discrete convolution)

y(t) = x(t) * h(t) (for continuous convolution)

There are two types of convolution : 1. Continuous Convolution

2. Discrete Convolution

2.2.1 Continuous Convolution :

Step 1 : Express each function in terms of τ like x(t) → x(τ) and h(t) → h(τ)

Step 2 : Reflect one of the functions say h(τ) → h(-τ)

Step 3 : Shifting : The signal h(-τ) is shifted by t units to slide over τ axis like h(-τ) → h(t-τ)

Step 4 : Start *t* at −∞ and slide it all the way to +∞. Wherever the two functions intersect, find the integral of their product.

The resulting waveform (not shown here) is the convolution of functions *x* and *h*

$$\int_{-\infty}^{\infty} x(t).\, h(t-\tau)\, d\tau = y(t)$$

Example :

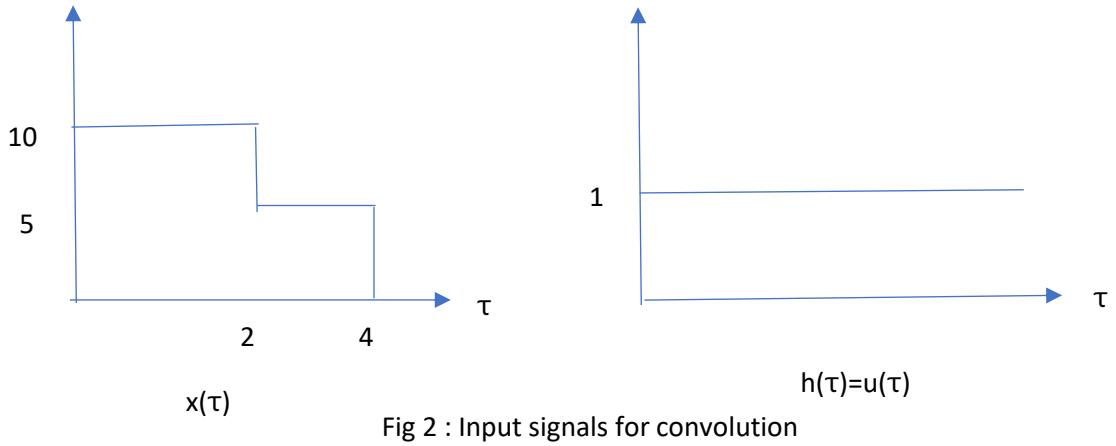
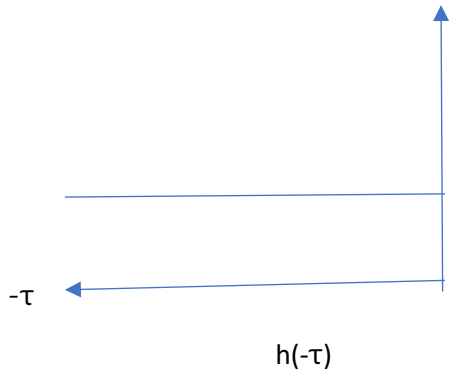
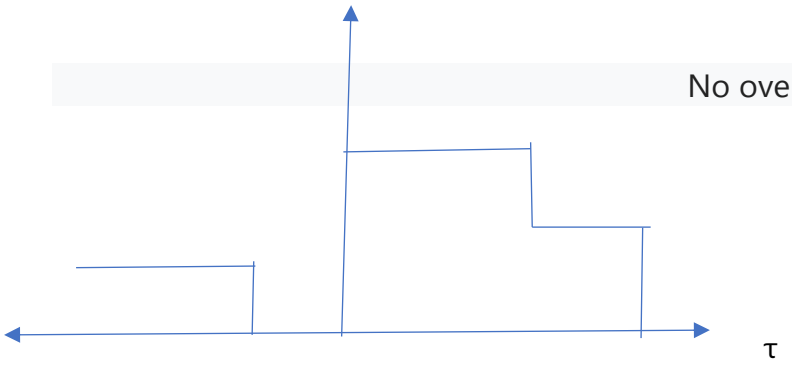


Fig 2 : Input signals for convolution



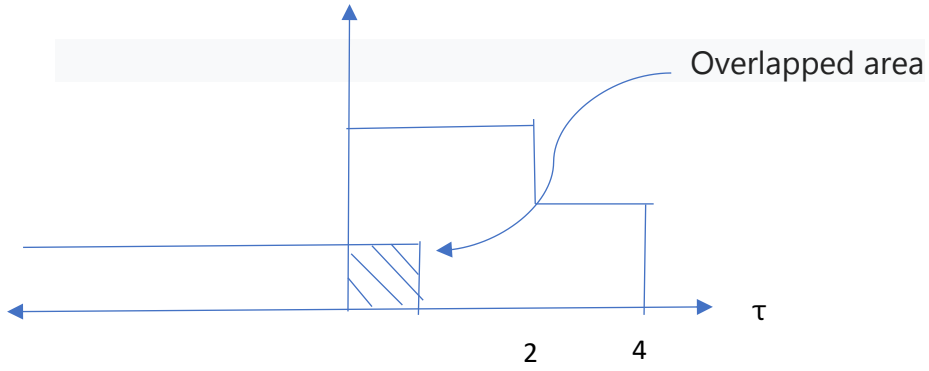
← Fig 3 : Reversed input signal

Case 1 : t < 0



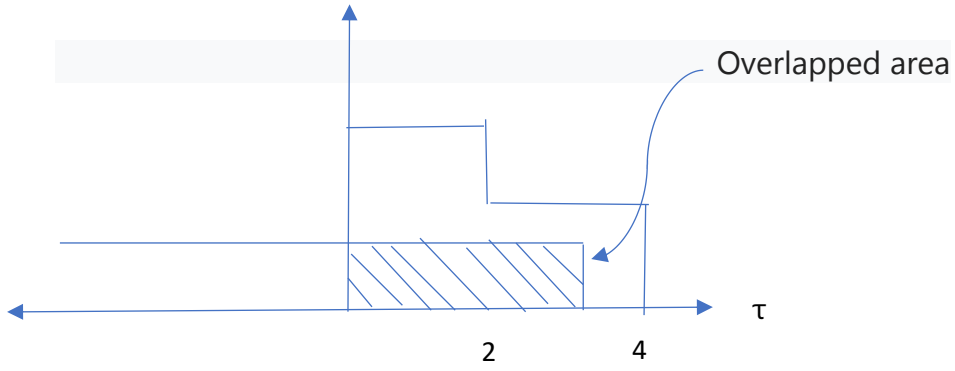
$$\int_{-\infty}^{\infty} x(t).\, h(t-\tau)\, d\tau = 0 = y_1(t)$$

Case 2 : 0 < t < 2



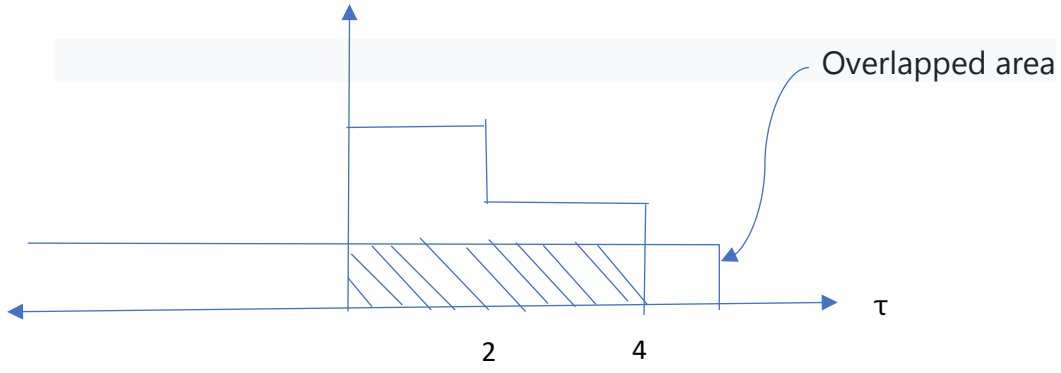
$$\int_0^t x(t).\, h(t-\tau)\, d\tau = 10.1.t = 10t = y_2(t)$$

Case 3 : $2 < t < 4$



$$\int_0^t x(t).\, h(t-\tau)\, d\tau = \int_0^2 10.1.\, d\tau + \int_2^t 5.1.\, d\tau = 10.2 + 5(t-2) = 10 + 5t = y_3(t)$$

Case 4 : $t > 4$



$$\int_0^t x(t).\, h(t-\tau)\, d\tau = \int_0^2 10.1.\, d\tau + \int_2^4 5.1.\, d\tau = 10.2 + 5.2 = 30 = y_4(t)$$

Therefore,

Convolution product= y(t) =0, t<0

10t , 0 <t <2, t=2

10 + 5t , 2 < t < 4, t=4

30, t > 4

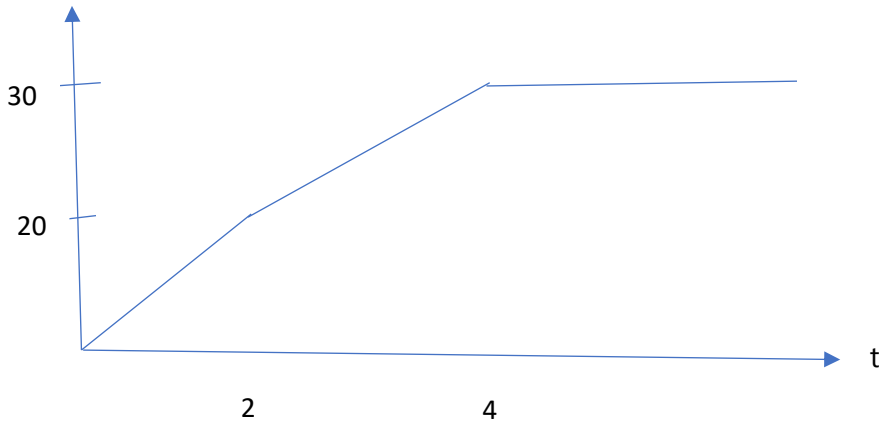


Fig 4 : Graph of Convolution product →

2.2.2 Discrete Convolution :

Similar to the Continuous Convolution,

$$(f * g)[n] = \sum_{m=-\infty}^{\infty} f[m]\, g[n-m]$$

Here, function g has been reversed and then shifted to n units to slide over m axis.

2.3 Applications :

- In image processing

  In digital image processing convolutional filtering plays an important role in many important algorithms in edge detection and related processes.

  In optics, an out-of-focus photograph is a convolution of the sharp image with a lens function. The photographic term for this is bokeh.

- In digital data processing

  In analytical chemistry, Savitzky–Golay smoothing filters are used for the analysis of spectroscopic data. They can improve signal-to-noise ratio with minimal distortion of the spectra.

  In statistics, a weighted moving average is a convolution.

- In acoustics, reverberation is the convolution of the original sound with echoes from objects surrounding the sound source.

  In digital signal processing, convolution is used to map the impulse response of a real room on a digital audio signal.

  In electronic music convolution is the imposition of a spectral or rhythmic structure on a sound. Often this envelope or structure is taken from another sound. The convolution of two signals is the filtering of one through the other.

Chapter 3

FOURIER TRANSFORM

3.1 Introduction :

The **Fourier transform** (**FT**) decomposes a function of time (a *signal*) into its constituent frequencies.

The term *Fourier transform* refers to both the frequency domain representation and the mathematical operation that associates the frequency domain representation to a function of time. The Fourier transform of a function of time is itself a complex-valued function of frequency, whose magnitude (modulus) represents the amount of that frequency present in the original function, and whose argument is the phase offset of the basic sinusoid in that frequency.

Fig 5 :Signal in the time domain

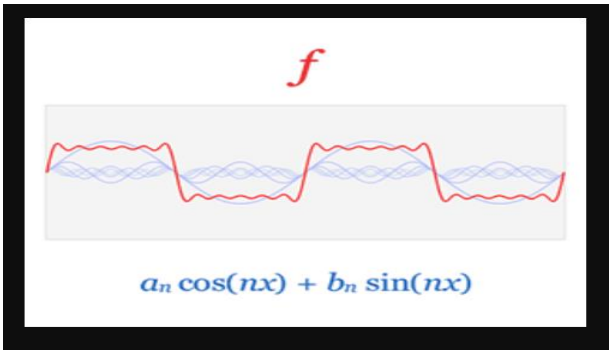


Fig 6 : Fourier series of the signal

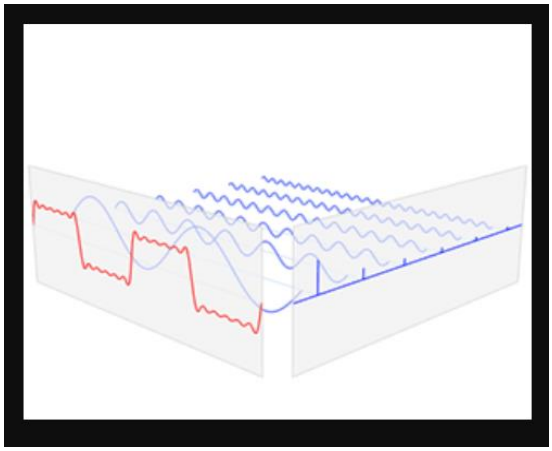


Fig 7 : Conversion to the signal on freq. domain

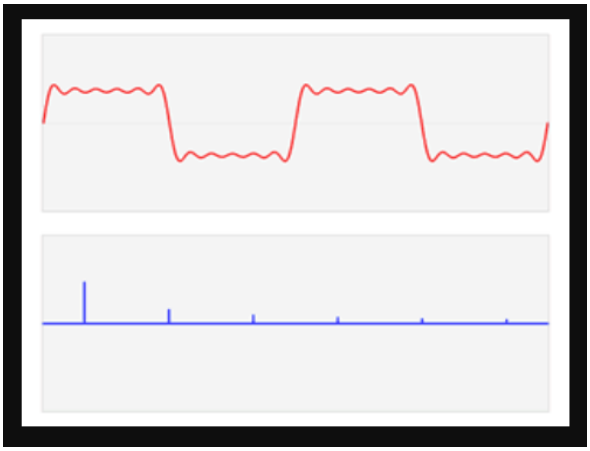


Fig 8 : Frequency domain vs time domain

Function (in red) is a sum of six sine functions of different amplitudes and harmonically related frequencies. Their summation is called a Fourier series. The

Fourier transform, (in blue), which depicts amplitude vs frequency, reveals the 6 frequencies (*at odd harmonics*) and their amplitudes (*1/odd number*).

3.2 Definition :

The Fourier transform of the function f is traditionally denoted by adding a circumflex: $\hat{f}$.

$$\hat{f}(\xi) = \int_{-\infty}^{\infty} f(x)\, e^{-2\pi i x \xi}\, dx,$$

for any real number, ξ. ξ is the frequency here.

Inverse fourier transform is given as,

$$f(x) = \int_{-\infty}^{\infty} \hat{f}(\xi)\, e^{2\pi i x \xi}\, d\xi,$$

For any real number, x.

Chapter 4

FOURIER SERIES

4.1 Introduction :

A Fourier series is an expansion of a periodic function $f(x)$ in terms of an infinite sum of sines and cosines. Fourier series make use of the orthogonality relationships of the sine and cosine functions. As such, the summation is a **synthesis** of another function. The process of deriving the weights that describe a given function is a form of Fourier **analysis**.

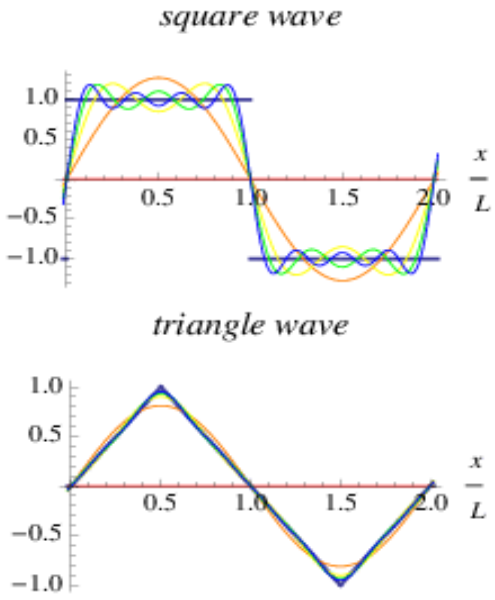


Fig 9 : Fourier series representation of the Square and triangle waves

4.2 Definition :

Using the method for a generalized Fourier series, the usual Fourier series involving sines and cosines is obtained by taking $f_1(x) = \cos x$ and $f_2(x) = \sin x$. Since these functions form a complete orthogonal system over $[-\pi, \pi]$, the Fourier series of a function $f(x)$ is given by

$$f(x) = \frac{1}{2} a_0 + \sum_{n=1}^{\infty} a_n \cos(n x) + \sum_{n=1}^{\infty} b_n \sin(n x),$$

where,

$$a_0 = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x)\, dx$$

$$b_n = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\sin(nx)\,dx$$

$$a_n = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\cos(nx)\,dx$$

Chapter 5

DISCRETE TIME FOURIER TRANSFORM

5.1 Introduction

In mathematics, the **discrete-time Fourier transform** (**DTFT**) is a form of Fourier analysis that is applicable to a sequence of values. The term *discrete-time* refers to the fact that the transform operates on discrete data, often samples whose interval has units of time. From uniformly spaced samples it produces a function of frequency that is a periodic summation of the continuous Fourier transform of the original continuous function.

The DTFT itself is a continuous function of frequency, but discrete samples of it can be readily calculated via the discrete Fourier transform (DFT) , which is by far the most common method of modern Fourier analysis.

5.2 Definition :

The discrete-time Fourier transform of a discrete set of real or complex numbers $x[n]$, for all integers n, is a Fourier series, which produces a periodic function of a frequency variable. When the frequency variable, ω, has normalized units of *radians/sample*, the periodicity is 2π, and the Fourier series is:

$$X_{2\pi}(\omega) = \sum_{n=-\infty}^{\infty} x[n]\, e^{-i\omega n}.$$

We can also write $X(e^{i\omega})$

5.2.1 Inverse DTFT :

An operation that recovers the discrete data sequence from the DTFT function is called an *inverse DTFT*.

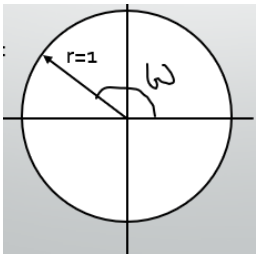
$$x[n] = \frac{1}{2\pi} \int_{(2\pi)} X(\omega) e^{j\omega n} d\omega$$

We did integration here because ω is continuous here and has the range from 0 to 2*pi or -π to π (in some cases)

IDTFT is a fourier series using the DTFT samples as co-efficients of complex sinusoids at the corresponding DTFT frequencies.

5.3 Omega (ω) Domain :

Fig 10 : ω domain →



ω ranges from 0 to 2π
i.e. ω is continuous.

5.4 Sampling the DTFT :

When the DTFT is continuous, a common practice is to compute an arbitrary number of samples (N) of one cycle of the periodic function $X_{1/T}$:

$$X_{1/T}(k/NT) = \underbrace{\sum_{N} x_N[n] \cdot e^{-i2\pi \frac{kn}{N}}}_{DFT}, \quad \text{(sum over any } n\text{-sequence of length } N\text{)}$$

If we compare this equation with the equation of DTFT we get ω=(2*pi*k)/N

It makes N divisions on the domain of ω, and sample with the factor k/N of 2*pi. .This part introduces the concept of Discrete Fourier Transform.

Program 1 : Taking 128 vowel samples and implementing DTFT in Python

```
In [7]: from cmath import exp, pi
        import matplotlib.pyplot as plt
        from numpy.fft import fftfreq
        %matplotlib notebook
        import numpy as np
        import mpmath as mp
        import scipy
        import scipy.stats as sp
        import matplotlib.pyplot as plt
        import subprocess
        import cmath as cm
        x=[]
        y=[]
```

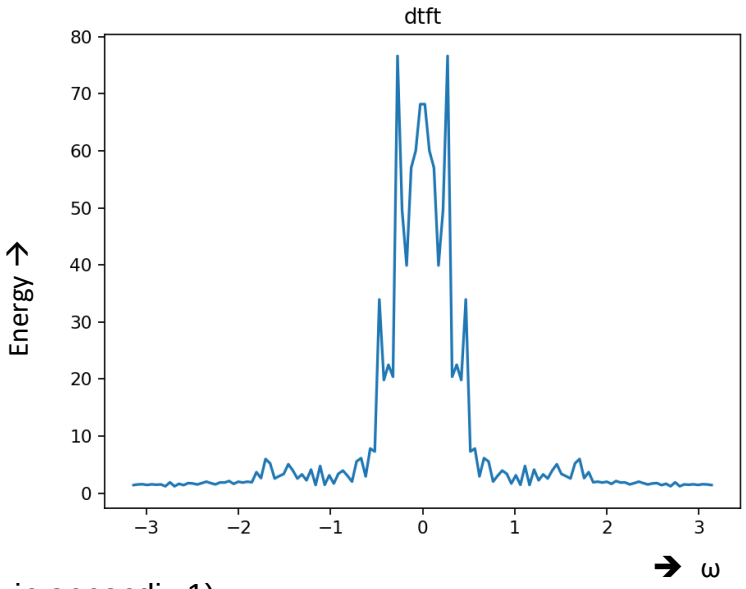
```
def dtft(x1,N):

    x=x1[0]
    j=cm.sqrt(-1)
    n=x1[1]
    X=[]

    w=np.linspace(-np.pi,np.pi,N)
    for i in range(0,N):
        w_tmp=w[i]
        X_tmp=0
        for k in range(0,len(x)):
            X_tmp+=(x[k]*np.exp(-n[k]*w_tmp*j))


        X.append(abs(X_tmp))
    print("lenX:",len(X))
    return w,X
```

```
f = open("Sample.norm", "r")
#print(f.read())
Nval=128
#print("\nEnter the values of the signal x[n]:\n")
for i in range(Nval):
    x.append(float(f.readline()))
#print(x)

for i in range(Nval):
    y.append(i)
x1=[x,y]
w,X=dtft(x1,Nval)

samples=dft(x,Nval)
```

```
plt.figure()
plt.title('dtft')
plt.plot(w,X)
#plt.show()
plt.figure()
plt.title('dft')
plt.plot(w,samples,label='dft')

plt.figure()
plt.title('fft')
plt.plot(w,fft(x),label='fft')

plt.show()
```

Fig 11: Snapshots of program code for dtft

OUTPUT :



<--Fig 12 : DTFT representation of 128 vowel samples

(Source code in appendix 1)

Program 2: Taking 5590 samples and implementing DTFT in Python

```
In [7]: from cmath import exp, pi
        import matplotlib.pyplot as plt
        from numpy.fft import fftfreq
        %matplotlib notebook
        import numpy as np
        import mpmath as mp
        import scipy
        import scipy.stats as sp
        import matplotlib.pyplot as plt
        import subprocess
        import cmath as cm
        x=[]
        y=[]
```

```
def dtft(x1,N):

    x=x1[0]
    j=cm.sqrt(-1)
    n=x1[1]
    X=[]

    w=np.linspace(-np.pi,np.pi,N)
    for i in range(0,N):
        w_tmp=w[i]
        X_tmp=0
        for k in range(0,len(x)):
            X_tmp+=(x[k]*np.exp(-n[k]*w_tmp*j))


        X.append(abs(X_tmp))
    print("lenX:",len(X))
    return w,X
```

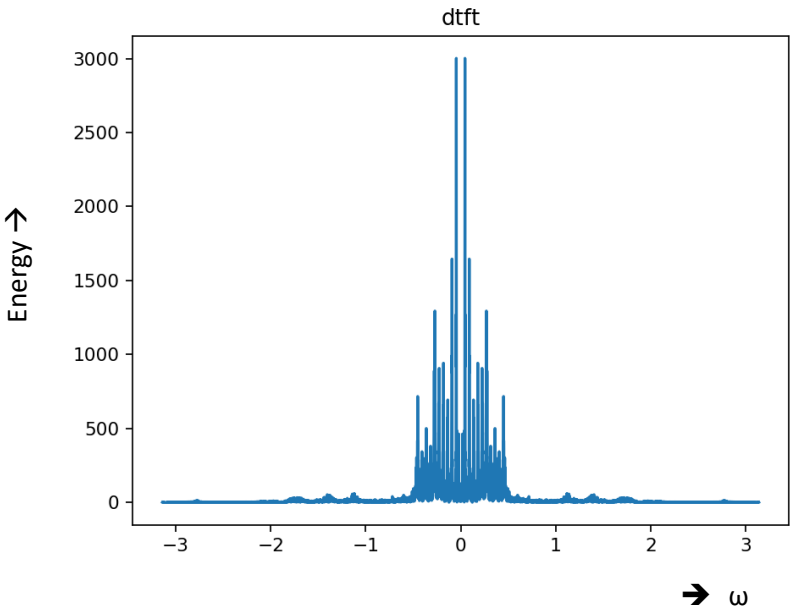
```
f = open("05010117_1a.norm", "r")
for line in f:
    x.append(float(line))
length=len(x)
#print("\nlen:",len(x))
lval=math.ceil(math.log(length,2))
Pval=2**(lval)
#print("\nPval=",Pval)
pad=Pval-len(x)
for i in range(pad):
    x.append(0.0)
#print("\nx:\n",x)
for i in range(Pval):
    y.append(i)
x1=[x,y]
w,X=dtft(x1,Pval)
#print("w\n",w)
absl=[]
samples=dft(x,Pval)
for iter in fft(x):
    absl.append(iter)
```

```
plt.figure()
plt.title('dtft')
plt.plot(w,X)
#plt.show()
plt.figure()
plt.title('dft')
plt.plot(w,samples,label='dft')

plt.figure()
plt.title('fft')
plt.plot(w,fft(x),label='fft')

plt.show()
```

Fig 13 : Code snapshots

Taking ω from -pi to pi i.e -3.14 to 3.14

OUTPUT :



← Fig 14: DTFT representation of 5590 vowel samples

(Source code in appendix 2)

Program 3: Recorded the vowel “AA” sound of my own voice, edited it in the software “Cool Edit Pro” and saved the file with .ascii extension.

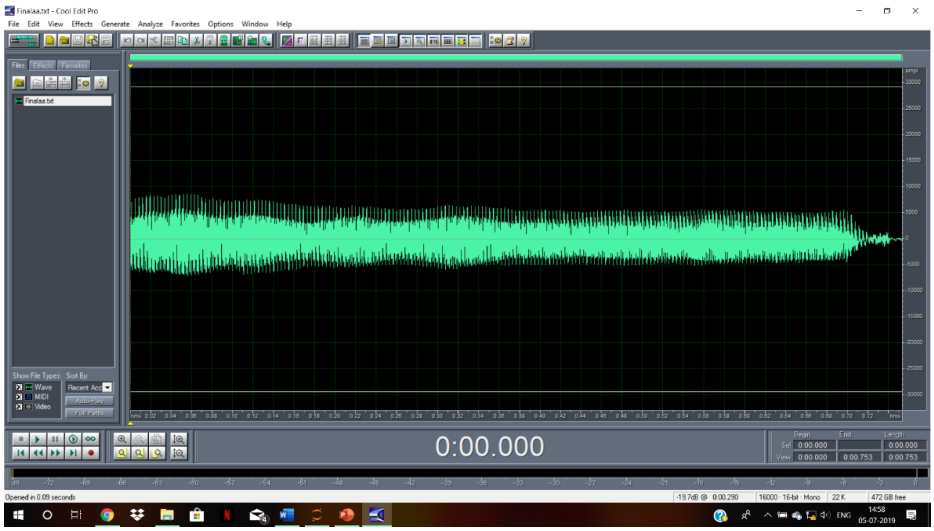


Fig 15: Recorded audio on CoolEdit Pro

The file contains 12,055 samples and implemented DTFT. ω ranges from -3.14 to 3.14.

```
f = open("Aa.txt", "r")
outf = open("AaFft.txt","w")
current_line=1
Nval=256
count=0
fs=[]
for line in f:
    x1.append(int(line))
for i in range(len(x1)):
    if count <Nval:
        x.append(x1[i])
        count=count+1
    else:
        for iter in fft(x):
            fs.append(iter)

        x=[]
        x.append(x1[i])
        count=1
#print("\nfs:",fs)
# print("\nlen of fs:",len(fs))

#print("\nx till 12032:",x1[:12032])
#print("\nlen of last x:",len(x)," ",x)
for i in range(len(x1)):
    y.append(i)
x2=[x1,y]
w,X=dtft(x2[:12032],12032)
```

Fig 16 : Code snapshot

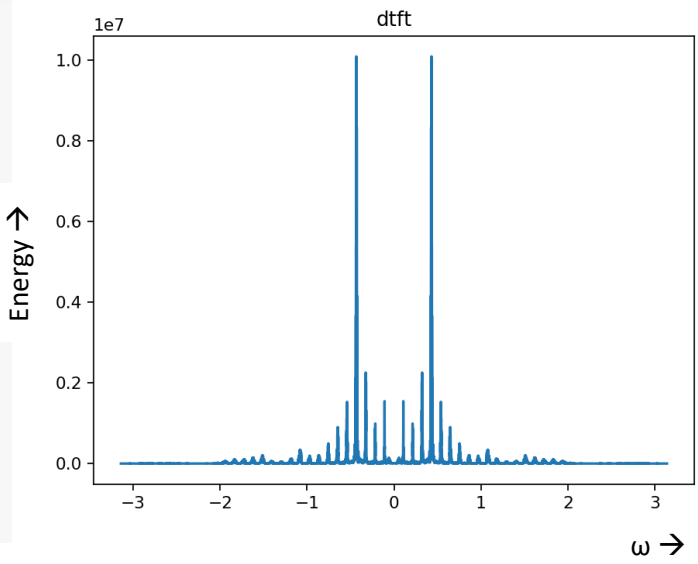


Fig 17 : DTFT rep. of recorded samples

(Source code in appendix 3)

Chapter 6

DISCRETE FOURIER TRANSFORM

6.1 Introduction :

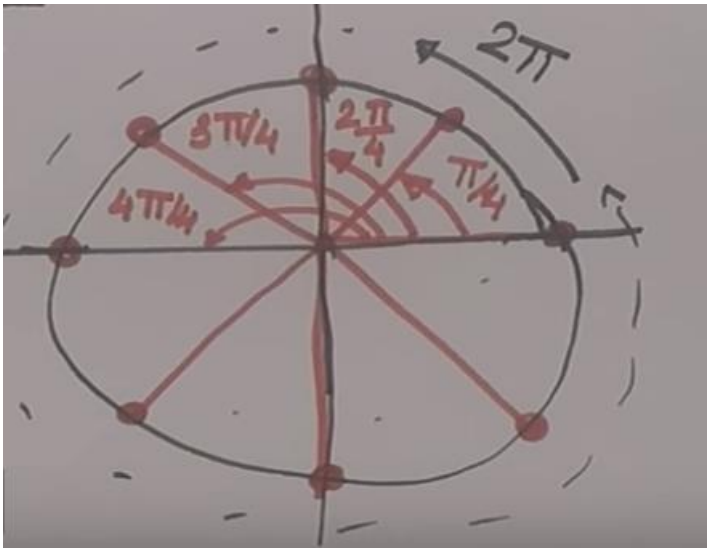
In mathematics, the **discrete Fourier transform** (**DFT**) converts a finite sequence of equally-spaced samples of a function into a same-length sequence of equally-spaced samples of the discrete-time Fourier transform (DTFT), which is a complex-valued function of frequency. The interval at which the DTFT is sampled is the reciprocal of the duration of the input sequence.

Since there are only a finite number of input data points, the DFT treats the data as if it were periodic (i.e. f(N) to f(2N-1) is same as f(0) to f(N-1)) T.

Since the operation treats the data as if it were periodic, we evaluate the DFT equation for the fundamental frequency (one cycle per sequenc, 1/NT Hz, 2π/NT rad/sec.) and its harmonics at ω=0).

i.e. set ω = 0, 2π/NT, 2π/NT x 2,…….2π/NT x (N-1)

If N=8 ,



← Fig 18: Discretised ω domain

So, in DFT we chopped ω into N equal divisions.

In other words we have discretised the frequency domain in DFT. In DFT, freq domain is discrete. In DTFT, we moved on a circle continuously but here we jump on a circle (from 0 to 1, then 1 to 2 and so on).

Since we find out the samples at the given points like here 0,1,2,……7 , therefore n ranges from 0 to N-1.

6.2 Definition :

$$F[n] = \sum_{k=0}^{N-1} f[k] e^{-j\frac{2\pi}{N}nk} \qquad (n = 0 : N-1)$$

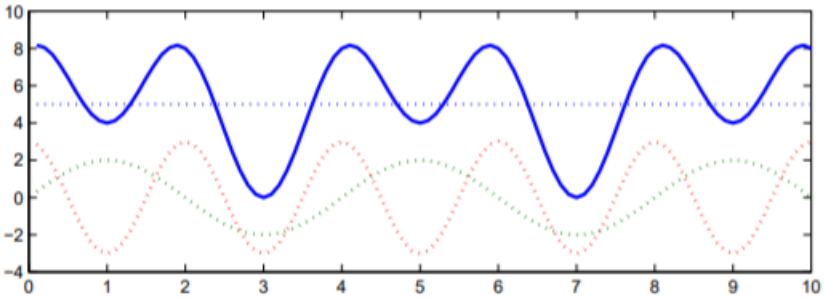
f[k] is the input sample and N is the number of divisions

6.2.1 Inverse DFT :

$$f[k] = \frac{1}{N} \sum_{n=0}^{N-1} F[n] e^{+j\frac{2\pi}{N}nk}$$

6.3 Example :

$$f(t) = \underbrace{5}_{\text{dc}} + \underbrace{2\cos(2\pi t - 90^o)}_{1\text{Hz}} + \underbrace{3\cos 4\pi t}_{2\text{Hz}}$$



← Fig 19: Original Signal

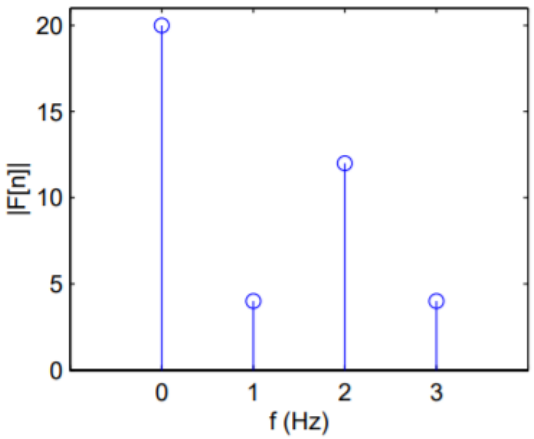
Let us sample f(t) at 4 times per second (i.e. fs= 4 Hz) from t=0 to t=3/4. The values of the discrete samples are given by :

$$f[k] = 5 + 2cos(\tfrac{\pi}{2}k - 90^o) + 3cos\pi k \qquad \text{by putting } t = kT_s = \tfrac{k}{4}$$

i.e. f[0] =8, f[1] =4, f[2] =8, f[3]=0 (N=4)

$$\text{Therefore} \quad F[n] = \sum_{0}^{3} f[k] e^{-j\frac{\pi}{2}nk} = \sum_{k=0}^{3} f[k](-j)^{nk}$$

$$\begin{pmatrix} F[0] \\ F[1] \\ F[2] \\ F[3] \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{pmatrix} \begin{pmatrix} f[0] \\ f[1] \\ f[2] \\ f[3] \end{pmatrix} = \begin{pmatrix} 20 \\ -j4 \\ 12 \\ j4 \end{pmatrix}$$



← Fig 20 : DFT of 4-point sequence

6.4 Time Complexity :

The time taken to evaluate a DFT on a digital computer depends principally on the number of multiplications involved, since these are the slowest operations. With the DFT, this number is directly related to N^2 (matrix multiplication of a vector), where N is the length of the transform. For most problems, is chosen to be at least 256 in order to get a reasonable approximation for the spectrum of the sequence under consideration – hence computational speed becomes a major consideration.

6.5 Programs :

Program 1 : Taking 128 vowel samples and implementing DFT in Python

In [7]:
```
from cmath import exp, pi
import matplotlib.pyplot as plt
from numpy.fft import fftfreq
%matplotlib notebook
import numpy as np
import mpmath as mp
import scipy
import scipy.stats as sp
import matplotlib.pyplot as plt
import subprocess
import cmath as cm
x=[]
y=[]
```

```
def dft(x,N):
    samples=[]
    for k in range(N):
        list=[]
        a=0
        for n in range(N):
            a=x[n]*cm.exp(-2j*cm.pi*k*n*(1/N))
            list.append(a)
            a=0
        summation=sum(list)
        samples.append(summation)
    #print("\nThe samples are:-\n[")
    #for i in range(N):
    #    print(samples[i]," ")
    #print("]")
    return samples
```

```
f = open("Sample.norm", "r")
#print(f.read())
Nval=128
#print("\nEnter the values of the signal x[n]:\n")
for i in range(Nval):
    x.append(float(f.readline()))
#print(x)

for i in range(Nval):
    y.append(i)
x1=[x,y]
w,X=dtft(x1,Nval)

samples=dft(x,Nval)
```

```
plt.figure()
plt.title('dtft')
plt.plot(w,X)
#plt.show()
plt.figure()
plt.title('dft')
plt.plot(w,samples,label='dft')

plt.figure()
plt.title('fft')
plt.plot(w,fft(x),label='fft')

plt.show()
```

Fig 21 : Code snapshots

OUTPUT :

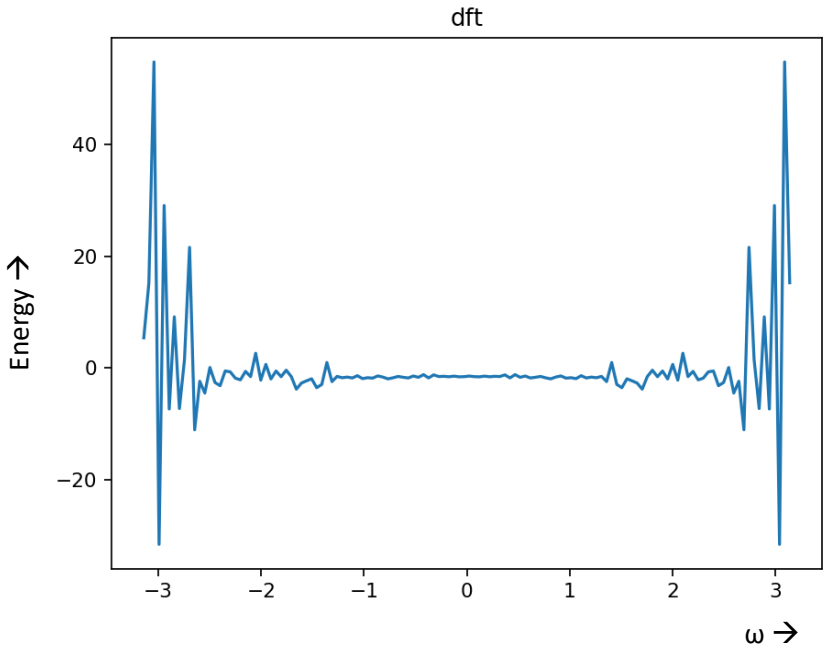


Fig 22: DFT rep of the 128 vowel samples

(Source code in appendix 1)

Program 2: Taking 5590 samples and implementing DFT in Python

```
In [7]: from cmath import exp, pi
        import matplotlib.pyplot as plt
        from numpy.fft import fftfreq
        %matplotlib notebook
        import numpy as np
        import mpmath as mp
        import scipy
        import scipy.stats as sp
        import matplotlib.pyplot as plt
        import subprocess
        import cmath as cm
        x=[]
        y=[]
```

```
def dft(x,N):
    samples=[]
    for k in range(N):
        list=[]
        a=0
        for n in range(N):
            a=x[n]*cm.exp(-2j*cm.pi*k*n*(1/N))
            list.append(a)
            a=0
        summation=sum(list)
        samples.append(summation)
    #print("\nThe samples are:-\n[")
    #for i in range(N):
    #    print(samples[i]," ")
    #print("]")
    return samples
```

```
f = open("05010117_1a.norm", "r")
for line in f:
    x.append(float(line))
length=len(x)
#print("\nlen:",len(x))
lval=math.ceil(math.log(length,2))
Pval=2**(lval)
#print("\nPval=",Pval)
pad=Pval-len(x)
for i in range(pad):
    x.append(0.0)
#print("\nx:\n",x)
for i in range(Pval):
    y.append(i)
x1=[x,y]
w,X=dtft(x1,Pval)
#print("w\n",w)
absl=[]
samples=dft(x,Pval)
for iter in fft(x):
    absl.append(iter)
#absl.append("%5.3f" % abs(f) for f in fft(x))
#print("absl\n",absl)
```

```
plt.figure()
plt.title('dtft')
plt.plot(w,X)
#plt.show()
plt.figure()
plt.title('dft')
plt.plot(w,samples,label='dft')

plt.figure()
plt.title('fft')
plt.plot(w,fft(x),label='fft')

plt.show()
```

Fig 23 : Code snapshots

OUTPUT :

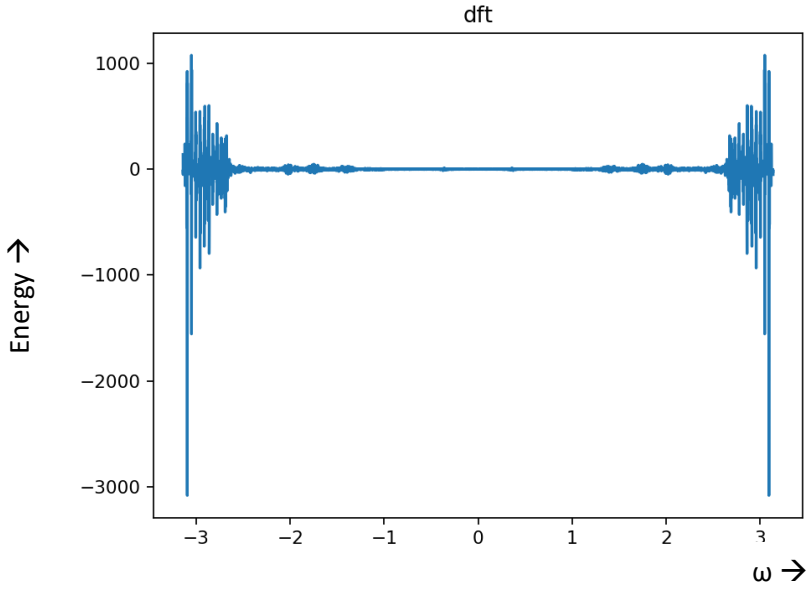


Fig 24 : DFT rep. of 5590 vowel samples

(Source code in appendix 2)

Program 3: Recorded the vowel “AA” sound of my own voice, edited it in the software “Cool Edit Pro” and saved the file with .ascii extension. It contained 12,055 samples.

Took window size as 256 samples and implemented DFT on those 256 samples every time till all the samples aren’t implemented.

```
f = open("Aa.txt", "r")
outf = open("AaFft.txt","w")
current_line=1
Nval=256
count=0
dfs=[]
for line in f:
    x1.append(int(line))
for i in range(len(x1)):
    if count <Nval:
        x.append(x1[i])
        count=count+1
    else:
        samples=dft(x,Nval)
        for i in range(Nval):
            dfs.append(samples[i])

        x=[]
        x.append(x1[i])
        count=1
w=np.linspace(-np.pi,np.pi,len(dfs))
print("\nlenfs:",len(dfs))
plt.figure()
#plt.subplot(3,1,2)
plt.title('dft')
plt.plot(w,dfs,label='dft')
```

```
def dft(x,N):
    samples=[]
    for k in range(N):
        list=[]
        a=0
        for n in range(N):
            a=x[n]*cm.exp(-2j*cm.pi*k*n*(1/N))
            list.append(a)
            a=0
        summation=sum(list)
        samples.append(summation)
    #print("\nThe samples are:-\n[")
    #for i in range(N):
    #    print(samples[i]," ")
    #print("]")
    return samples
```

Fig 25 : Code snapshots

OUTPUT :

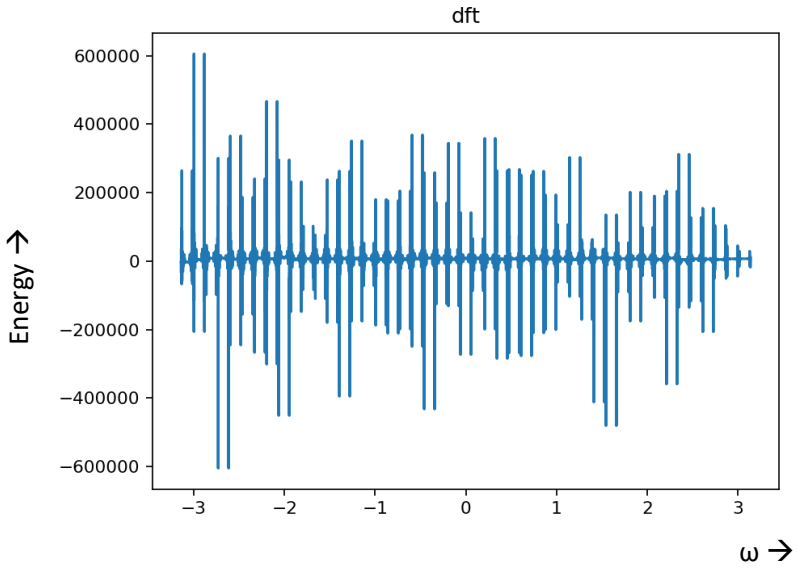


Fig 25 : DFT representation of 12055 recorded vowel samples taking window size as 256 samples

(Source code in appendix 4)

Chapter 7

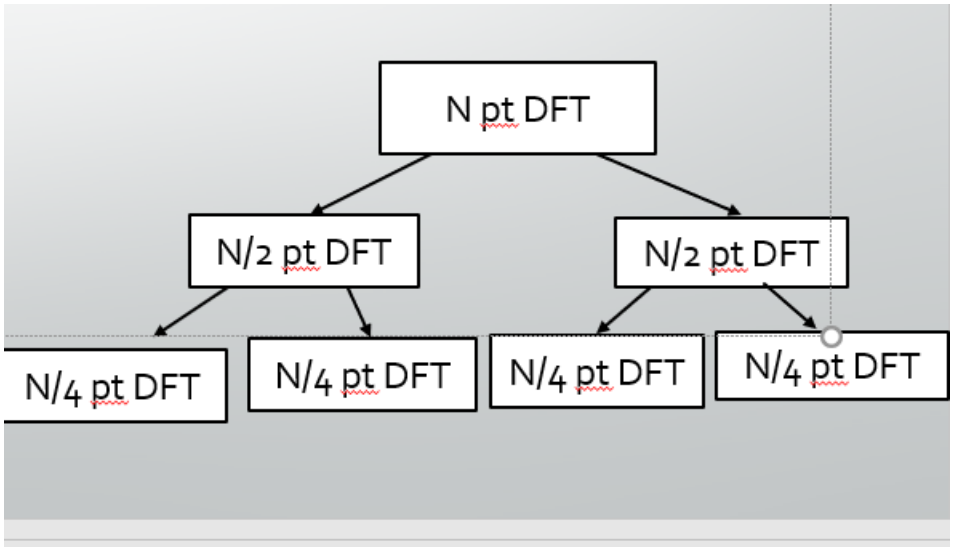
FAST FOURIER TRANSFORM

7.1 Introduction :

The fast Fourier transform (FFT) is a discrete Fourier transform algorithm which reduces the number of computations needed for N points from $2N^2$ to $2N \lg N$, where lg is the base-2 logarithm.

This algorithm was formulated by Cooley and Tuckey.

It is basically an algorithm to solve DFT in a faster manner and is based on the idea of “Divide and Conquer”.



← Fig 26: Divide and conquer concept of FFT

7.2 Definition :

$$\text{Re-writing} \quad F[n] = \sum_{k=0}^{N-1} f[k] e^{-j\frac{2\pi}{N}nk} \text{ as} F[n] = \sum_{k=0}^{N-1} f[k] W_N^{nk}$$

It is easy to realise that the same values of W_N^{nk} are calculated many times as the computation proceeds. Firstly, the integer product nk repeats for different combinations of k and n. Secondly, W_N^{nk} is a periodic function with only N distinct values.

For example, consider N=8 (the FFT is simplest by far if N is an integral power of 2)

$$W_8^1 = e^{-j\frac{2\pi}{8}} = e^{-j45^o} = \frac{1-j}{\sqrt{2}} = a, \text{ say.}$$

Then $a^2 = -j \qquad a^3 = -ja = -a^* \qquad a^4 = -1$

$$a^5 = -a \qquad a^6 = j \qquad a^7 = ja = a^* \qquad a^8 = 1$$

From the above we can see that,

$$\begin{aligned} W_8^4 &= -W_8^0 \\ W_8^5 &= -W_8^1 \\ W_8^6 &= -W_8^2 \\ W_8^7 &= -W_8^3 \end{aligned}$$

Also, if nk falls outside the range 0-7, we still get one of the above values:

eg. if $n = 5$ and $k = 7$, $\quad W_8^{35} = a^{35} = (a^8)^4 \cdot a^3 = a^3$

7.2.1 Decimation-in-time algorithm :

Let us begin by splitting the single summation over N samples into 2 summations, each with N/2 samples, one for k even and the other for k odd. Substitute m =k/2 for k even and m=(k-1)/2 for k odd and write :

$$F[n] = \sum_{m=0}^{\frac{N}{2}-1} f[2m] W_N^{2mn} + \sum_{m=0}^{\frac{N}{2}-1} f[2m+1] W_N^{(2m+1)n}$$

Note that $W_N^{2mn} = e^{-j\frac{2\pi}{N}(2mn)} = e^{-j\frac{2\pi}{\frac{N}{2}}mn} = W_{\frac{N}{2}}^{mn}$

Therefore $F[n] = \sum_{m=0}^{\frac{N}{2}-1} f[2m] W_{\frac{N}{2}}^{mn} + W_N^m \sum_{m=0}^{\frac{N}{2}-1} f[2m+1] W_{\frac{N}{2}}^{mn}$

ie. $F[n] = G[n] + W_N^m H[n]$

Thus the N-point DFT F[n] can be obtained from two N/2-point transforms, one on even input data, G[n] and one on odd input data, H[n]. Although the

frequency index n ranges over N values, only N/2 values of G[n] and H[n] need to be computed since G[n] and H[n] are periodic in n with period N/2.

For example N=8 :

Even input data : f[0] f[2] f[4] f[6]

Odd input data : f[1] f[3] f[5] f[7]

$$
\begin{aligned}
F[0] &= G[0] + W_8^0 H[0] \\
F[1] &= G[1] + W_8^1 H[1] \\
F[2] &= G[2] + W_8^2 H[2] \\
F[3] &= G[3] + W_8^3 H[3] \\
F[4] &= G[0] + W_8^4 H[0] = G[0] - W_8^0 H[0] \\
F[5] &= G[1] + W_8^5 H[1] = G[1] - W_8^1 H[1] \\
F[6] &= G[2] + W_8^6 H[2] = G[2] - W_8^2 H[2] \\
F[7] &= G[3] + W_8^7 H[3] = G[3] - W_8^3 H[3]
\end{aligned}
$$

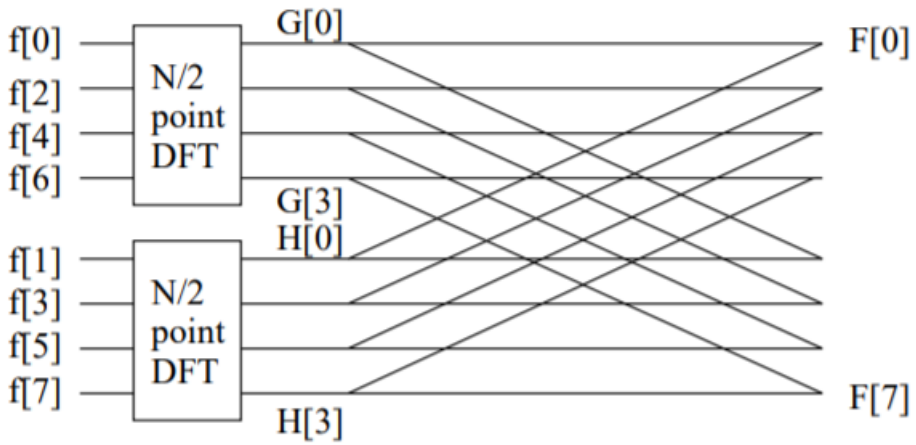


Fig 27: FFT flow graph

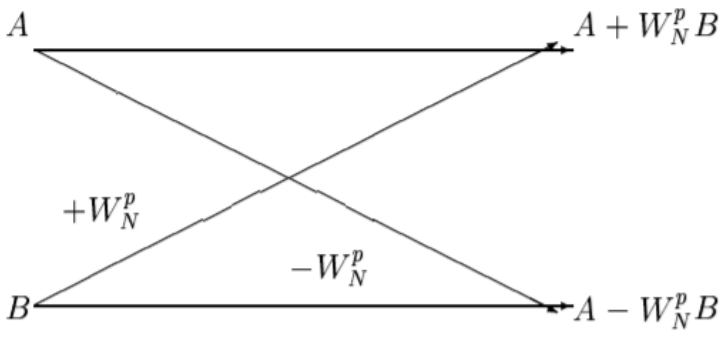
Assuming than N is a power of 2, we can repeat the above process on the two N/2 -point transforms, breaking them down to N/4 -point transforms, etc … we come down to 2-point transforms. For N=8,only one further stage is needed

Thus the FFT is computed by dividing up, or decimating, the sample sequence

f[k] into sub-sequences until only 2-point DFT's remain. Since it is the input, or time, samples which are divided up, this algorithm is known as the decimation-in-time (DIT) algorithm. The basic computation at the heart of the FFT is known as the butterfly because of its criss-cross appearance

7.2.2 Decimation-in-frequency :

An equivalent algorithm exists for which the output, or frequency, points are sub-divided – the decimation-in-frequency algorithm.



← Fig 28 : Butterfly operation in FFT

where A and B are complex numbers. Thus a butterfly computation requires one complex multiplication and 2 complex additions.

7.3 Computational Speed of FFT :

At each stage of the FFT (i.e. each halving) N/2 complex multiplications are required to combine the results of the previous stage. Since there are ($\log_2 N$) stages, the number of complex multiplications required to evaluate an N-point DFT with the FFT is approximately $N/2\log_2 N$.

7.4 Programs :

Program 1: Taking 128 (2^7) vowel samples and implementing FFT in Python

```
In [7]: from cmath import exp, pi
        import matplotlib.pyplot as plt
        from numpy.fft import fftfreq
        %matplotlib notebook
        import numpy as np
        import mpmath as mp
        import scipy
        import scipy.stats as sp
        import matplotlib.pyplot as plt
        import subprocess
        import cmath as cm
        x=[]
        y=[]
```

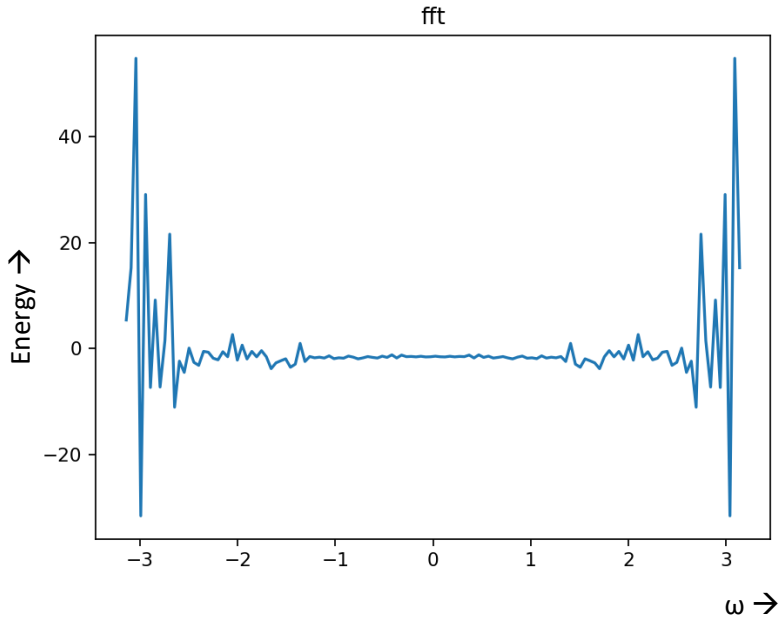
```
def fft(x1):
    N = len(x1)
    if N <= 1: return x1
    even = fft(x1[0::2])
    odd =  fft(x1[1::2])
    T= [exp(-2j*pi*k/N)*odd[k] for k in range(N//2)]
    return [even[k] + T[k] for k in range(N//2)] + \
           [even[k] - T[k] for k in range(N//2)]
```

```
f = open("Sample.norm", "r")
#print(f.read())
Nval=128
#print("\nEnter the values of the signal x[n]:\n")
for i in range(Nval):
    x.append(float(f.readline()))
#print(x)

for i in range(Nval):
    y.append(i)
x1=[x,y]
w,X=dtft(x1,Nval)

samples=dft(x,Nval)
```

```
plt.figure()
plt.title('dtft')
plt.plot(w,X)
#plt.show()
plt.figure()
plt.title('dft')
plt.plot(w,samples,label='dft')

plt.figure()
plt.title('fft')
plt.plot(w,fft(x),label='fft')

plt.show()
```

Fig 29 : Code snapshots

OUTPUT :



← Fig 30 : FFT rep of 128 vowel samples

(Source code in appendix 1)

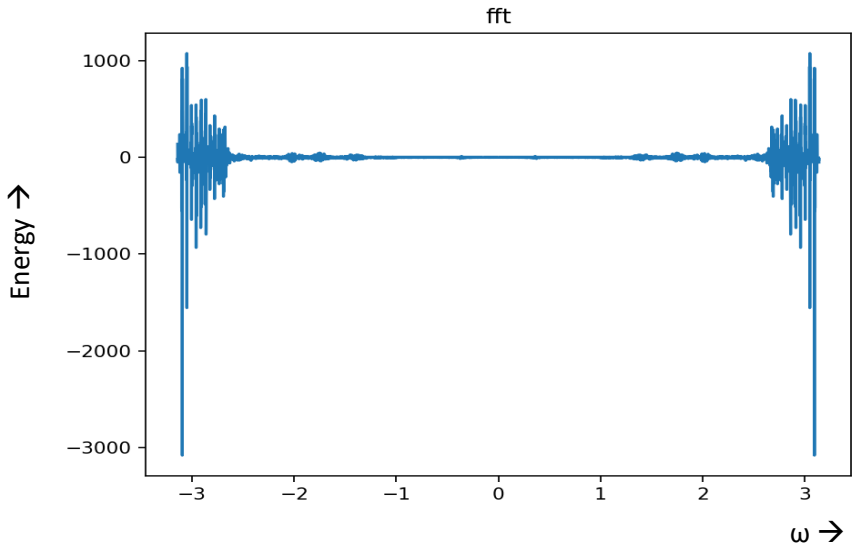
Program 2 : 5590 (not 2 power number) vowel samples hence to make nearest 2 power number of samples, that many padding is done and finally FFT is implemented to the entire value set.

```
f = open("05010117_1a.norm", "r")
for line in f:
    x.append(float(line))
length=len(x)
#print("\nLen:",Len(x))
lval=math.ceil(math.log(length,2))
Pval=2**(lval)
#print("\nPval=",Pval)
pad=Pval-len(x)
for i in range(pad):
    x.append(0.0)
#print("\nx:\n",x)
for i in range(Pval):
    y.append(i)
x1=[x,y]
w,X=dtft(x1,Pval)
#print("w\n",w)
absl=[]
samples=dft(x,Pval)
for iter in fft(x):
    absl.append(iter)
#absl.append("%5.3f" % abs(f) for f in fft(x))
#print("absl\n",absl)
```

```
def fft(x1):
    N = len(x1)
    if N <= 1: return x1
    even = fft(x1[0::2])
    odd =  fft(x1[1::2])
    T= [exp(-2j*pi*k/N)*odd[k] for k in range(N//2)]
    return [even[k] + T[k] for k in range(N//2)] + \
           [even[k] - T[k] for k in range(N//2)]
```

Fig 31 : Code snapshots

OUTPUT :



← Fig 32 : FFT rep. of 5590 (with padding) vowel samples

(Source code in appendix 2)

Program 3: Recorded the vowel “AA” sound of my own voice, edited it in the software “Cool Edit Pro” and saved the file with .ascii extension. It contained 12,055 (not 2 power number) samples.

Took window size as 256 samples and implemented FFT on those 256 samples every time till all the samples aren’t implemented.

It is to be noted that 256 x 47 =12032 < 12055. But 256 x 48 = 12,288 > 12,055

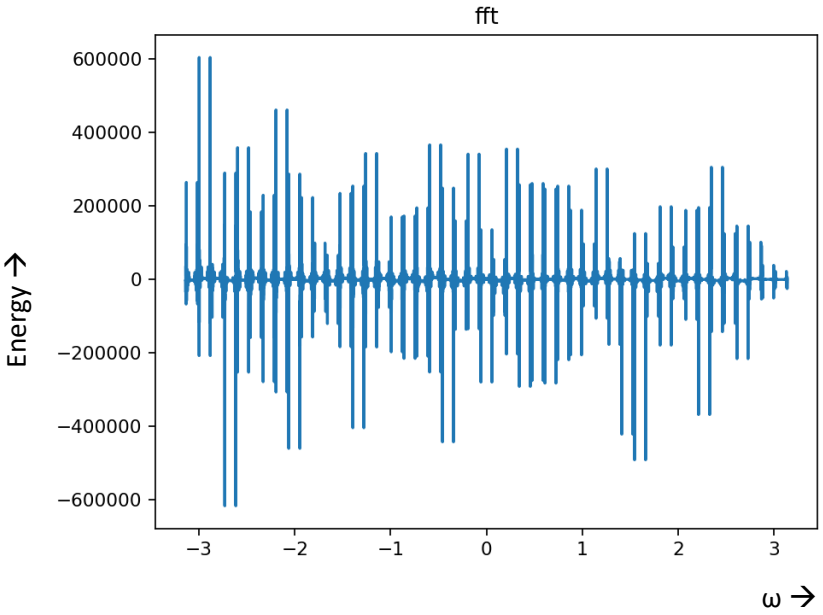
So had to ignore the last 23 samples.

```
f = open("Aa.txt", "r")
outf = open("AaFft.txt","w")
current_line=1
Nval=256
count=0
fs=[]
for line in f:
    x1.append(int(line))
for i in range(len(x1)):
    if count <Nval:
        x.append(x1[i])
        count=count+1
    else:
        for iter in fft(x):
            fs.append(iter)

        x=[]
        x.append(x1[i])
        count=1
#print("\nfs:",fs)
# print("\nlen of fs:",len(fs))

#print("\nx till 12032:",x1[:12032])
#print("\nlen of last x:",len(x)," ",x)
for i in range(len(x1)):
    y.append(i)
x2=[x1,y]
w,X=dtft(x2[:12032],12032)
```

```
def fft(x1):
    N = len(x1)
    if N <= 1: return x1
    even = fft(x1[0::2])
    odd =  fft(x1[1::2])
    T= [exp(-2j*pi*k/N)*odd[k] for k in range(N//2)]
    return [even[k] + T[k] for k in range(N//2)] + \
           [even[k] - T[k] for k in range(N//2)]
```

Fig 33 : Code snapshots

OUTPUT :



← Fig 34 : FFT rep of 12055 recorded vowel samples taking window size as 256 samples

(Source code in appendix 3)

Program 4 : Taking overlapped windows :

Have 12,055 vowel samples. Took window size as 256 samples. Every time window shifts by 100 samples, thus overlapping some samples of the preceding window.

For example, at first take 1- 256 samples, then 101-356 samples and so on.

Its to be noted that in the last window, there were 255 (not 256) samples so could not ignore those much samples. Therefore padded one dummy sample (0) to make 256 samples.

Padding limit fixed is : 10 numbers

```
:
from cmath import exp, pi
import matplotlib.pyplot as plt
%matplotlib notebook
from numpy.fft import fftfreq
import numpy as np
import mpmath as mp
import math
import scipy
import scipy.stats as sp
import matplotlib.pyplot as plt
import subprocess
import cmath as cm
x=[]
y=[]
x1=[]
```

```
def fft(x1):
    N = len(x1)
    if N <= 1: return x1
    even = fft(x1[0::2])
    odd =  fft(x1[1::2])
    T= [exp(-2j*pi*k/N)*odd[k] for k in range(N//2)]
    return [even[k] + T[k] for k in range(N//2)] + \
           [even[k] - T[k] for k in range(N//2)]
```

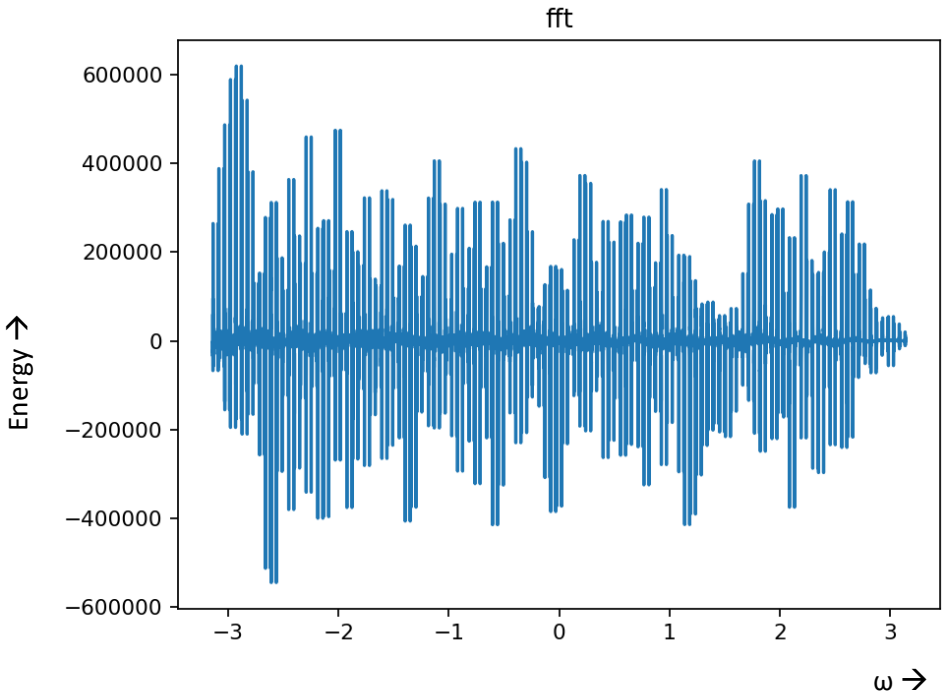
```
f = open("Aa.txt", "r")
Nval=256
count=0
fs=[]
overlap=0
sum=0
for line in f:
    x1.append(int(line))
# print("\nlen of x1:",len(x1))
j=overlap
length=len(x1)
i=0
while i < length:
#     print("\nlenx1:",length)
    if count < Nval:
        x.append(x1[j])
        count=count+1
        j=j+1
        i=i+1
    else:
```

```
    else:

#         print("\nx:\n",x)
#         sum+=len(x)
        for iteri in fft(x):
            fs.append(iteri)
        overlap=overlap+100
        j=overlap
        x=[]
        x.append(x1[j])
        count=1
        j=j+1
        i=i+1
        length=length+156

diff=Nval-len(x)
##Limit of padding is 10
if diff <= 10:
    i=1
    while i <= diff:
        x.append(0)
        i=i+1
for iteri in fft(x):
    fs.append(iteri)
```

```
w=np.linspace(-np.pi,np.pi,len(fs))
plt.figure()
#plt.subplot(3,1,3)
plt.title('fft')
plt.plot(w,fs,label='fft')
```

Fig 35 : Code snapshots

OUTPUT :



← Fig 36 : FFT rep. of 12055 recorded vowel samples taking overlapped windows of size =256 samples

(Source code in appendix 5)

Chapter 8

DISCRETE WAVELET TRANSFORM

8.1 Wavelet Introduction :

A wavelet is a rapidly decaying wave-like oscillation that has 0 mean. Unlike sinusoids which extend to infinity, a wavelet extends for a finite duration. These are basically used to analyse signals.

8.1.1 Wavelet Analysis :

Two major transforms in wavelet analysis are:

1. Continuous Wavelet Transform (for continuous signals)
2. Discrete Wavelet Transform (for discrete signals)

8.2 Introduction to Discrete Wavelet Transform :

In wavelet analysis, the Discrete Wavelet Transform (DWT) decomposes a signal into a set of mutually orthogonal wavelet basis functions. These functions differ from sinusoidal basis functions in that they are spatially localized – that is, nonzero over only part of the total signal length. Furthermore, wavelet functions are dilated, translated and scaled versions of a a common function φ, known as the mother wavelet.

DWT is invertible, like Discrete Fourier Transform.

8.3 Multilevel Frequency Decomposition in DWT :

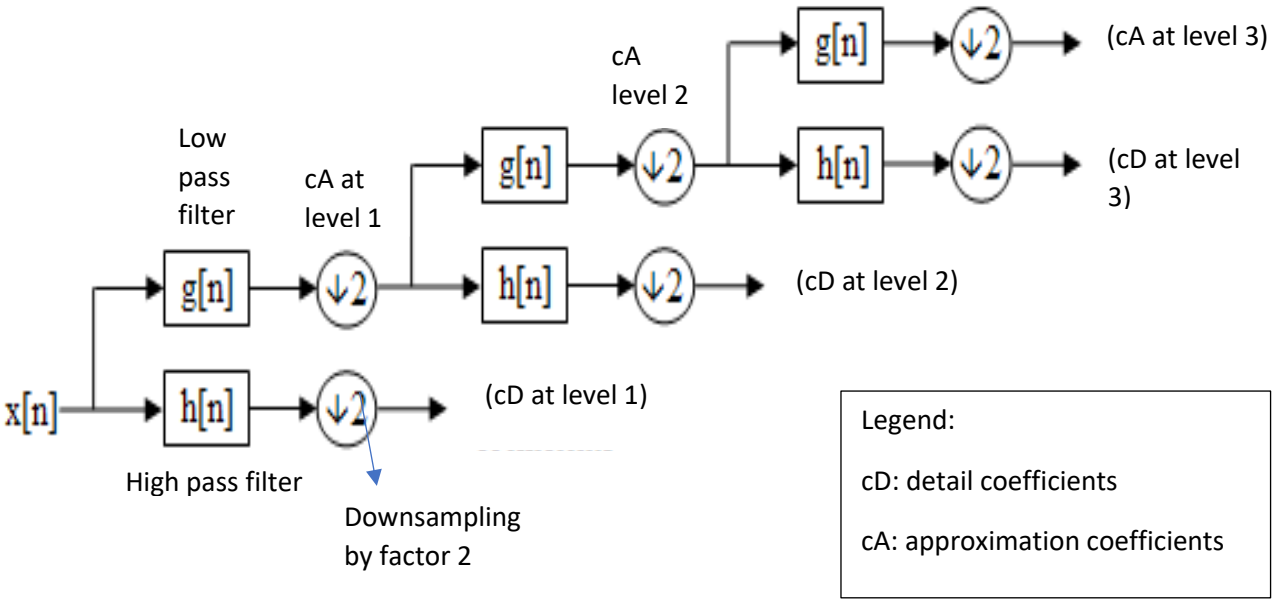


FIGURE 37: 3-level Frequency Decomposition of a signal

8.4 Example :

Let us take 16 discrete signals as input x vector.

First, the vector x is filtered with some discrete-time, low-pass filter (LPF) h of given length (in the Figures, we use length four for illustration purposes) at intervals of two, and the resulting values are stored in the first eight elements of w. This step is illustrated in Figure 2(a). Second, the vector x is filtered with some discrete-time, high-pass filter (HPF) g of given length (again, for illustration purposes, we use a filter of length four) at intervals of two, and the resulting high-pass values are stored in the last eight elements of w. This step is illustrated in Figure 2(b).

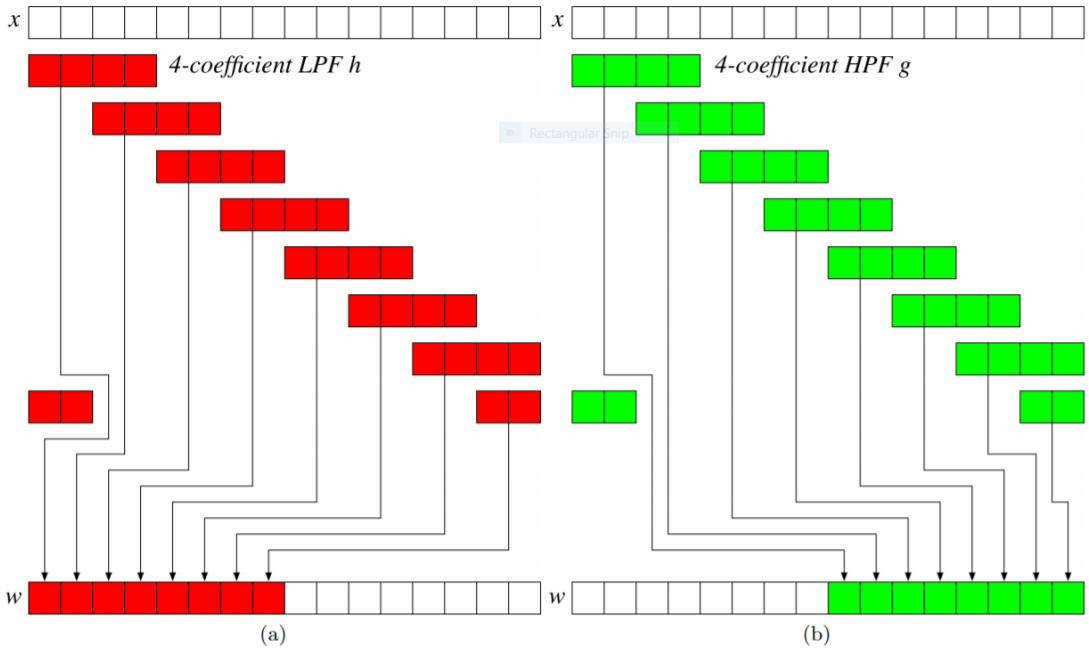


Figure 38: (a) First step of the DWT for a signal of length 16: The original signal is low-pass filtered in increments of two, and the resulting coefficients are grouped as the first eight elements of the vector. (b) Second step of the DWT: The original signal is high-pass filtered in increments of two, and the resulting coefficients are grouped as the last eight elements of the vector.

Note, qualitatively, how this procedure transforms the vector x. The low-pass part of the vector w is essentially a down-sampled version (down-sampled by a factor of two) of the original signal x, while the high-pass part of the vector w detects and localizes high frequencies in x. If we were to stop here, the vector w would be a one-level wavelet transform of x. We need not, however, stop here; the low-pass filtered part of w (first eight elements for this example) can be further transformed using the identical procedure as outlined above and shown in Figure 2. Figure 3, for example, illustrates a three-level, one-dimensional DWT.

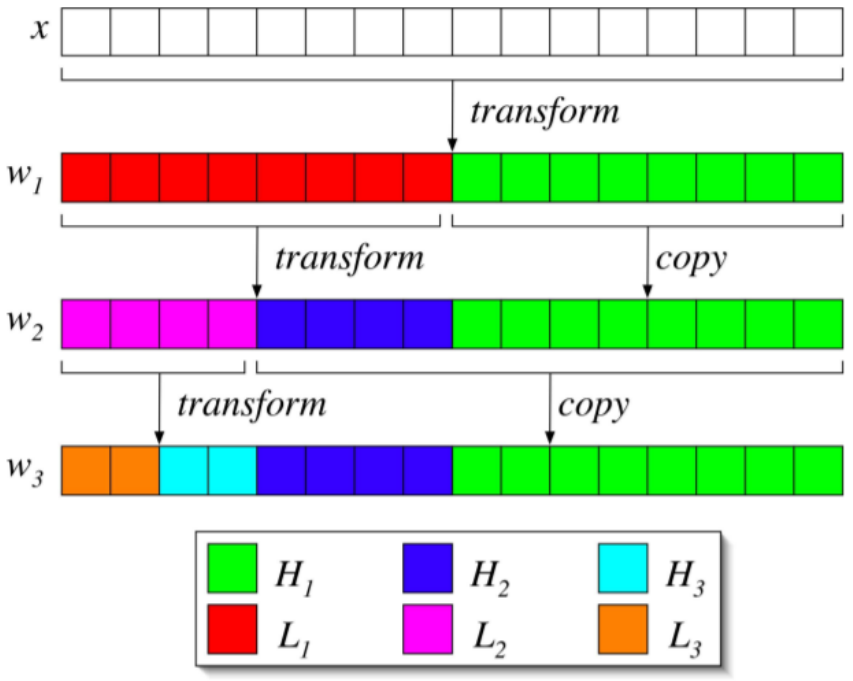


Figure 39: Three-level wavelet transform on signal x of length 16. Note that from w1 to w2, coefficients H1 remain unchanged, while from w2 to w3, coefficients H1 and H2 remain unchanged.

Note that in the final transform w3, values L3 are the result of three consecutive low-pass filters, values H3 are the result of two consecutive low-pass filtering operations followed by a high-pass filter, values H2 are the result of a low-pass filter followed by a high-pass filter, and values H1 are the result of one high-pass filter. Therefore, the highest frequencies will be isolated and localized in values H1 of w3, intermediate frequencies will be isolated and localized in values H2 of w3, etc. Note that for lower frequencies, the resolution is decimated by half for each level of the wavelet transform. In summary, the one-dimensional DWT is a multi-resolutional frequency decomposition and localization of a one-dimensional, discrete-time signal.

8.5 Interpretation of Coefficients :

The **approximation**, or scaling, **coefficients** are the lowpass representation of the signal and the details are the wavelet **coefficients**. At each subsequent level, the **approximation coefficients** are divided into a coarser **approximation** (lowpass) and highpass (**detail**) part.

8.6 Programs (Using PyWavelets) :

Program 1 : Python implementation of DWT taking vowel samples as input and no of levels fixed = 3

```
In [5]: from pywt import wavedec
        import matplotlib.pyplot as plt
        %matplotlib notebook
        x=[]

        f = open("05010117_1a.norm", "r")
        for line in f:
            x.append(line)
```

```
In [10]: import pywt
         coeffs = pywt.wavedec(x[:300],'db2',level=3)
         cA3,cD3,cD2,cD1=coeffs

         plt.figure()
         plt.plot(cA3)
         plt.title('Approx. coefficients at level3')
         plt.show()

         plt.figure()
         plt.plot(cD3)
         plt.title('Detailed coefficients at level3')

         plt.figure()
         plt.plot(cD2)
         plt.title('Detailed coefficients at level2')

         plt.figure()
         plt.plot(cD1)
         plt.title('Detailed coefficients at level1')
         plt.show()
```

Fig 40 : Code snapshots

OUTPUT :

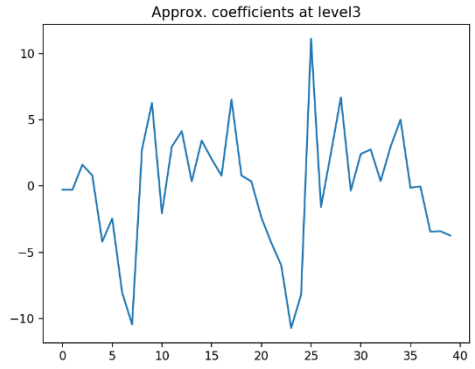


Fig 41: approx. coeff. at level 3

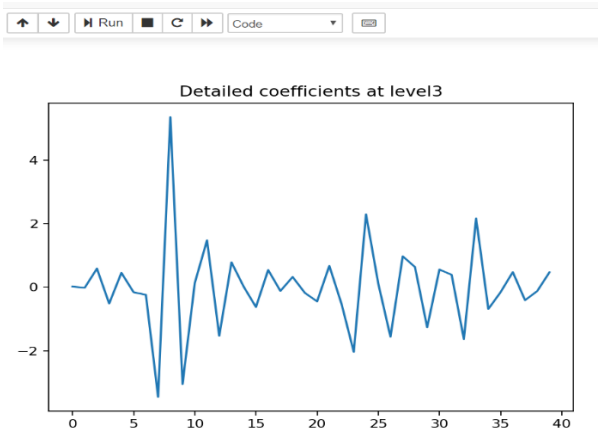


Fig 42: detail coeff. at level 3

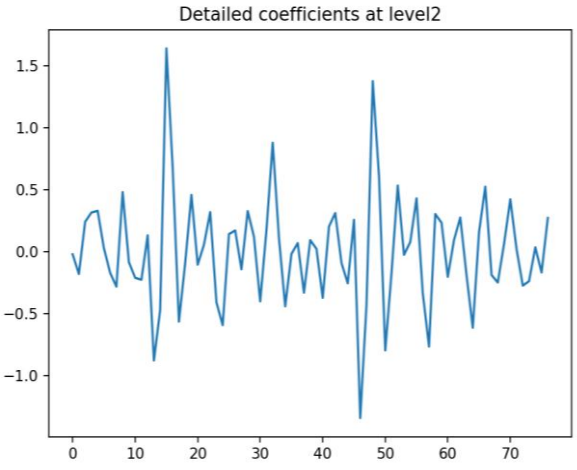


Fig 43: detail coeff. at level 2

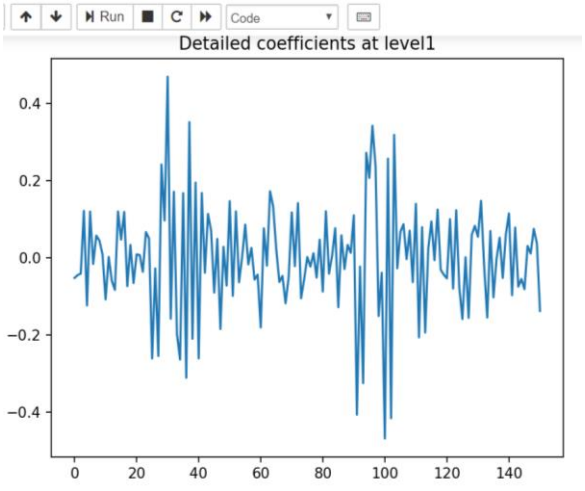


Fig 44: detail coeff. at level 1

Conclusion

Fourier Analysis stands as a tall pillar of Speech Signals Processing. Converting to frequency domain means how much of that particular frequency is present in the original signal, which simplifies a lot of complex tasks and helps in further processing a signal. Not only in digital speech processing, this fourier analysis is also of great demand in image processing field. Nowadays Discrete Wavelet Transform is an emerging, strong field which helps in de-noising signals, signal compression and what not, which further aids to Speech recognition in AI systems.

Thus, Speech Processing gives a lot to ever- growing technology of generations.

References

1. Wikipedia
2. https://www.youtube.com/watch?v=oLHPKyB8wlo&pbjreload=10
3. Ekeeda videos
4. https://www.quora.com
5. https://www.youtube.com/watch?v=xnVaHkRaJOw&list=PL4K9r9dYCOoqHtpgNSnoPlNFho_mq1eqz&index=7
6. www.google.com
7. Stanford.edu/materials
8. www.mathworks.com:Matlab documentations on fft,dft
9. Wolfram MathWorld
10. http://www.robots.ox.ac.uk/~sjrob/Teaching/SP/l7.pdf
11. Nptelhrd on youtube
12. http://matlab.izmiran.ru
13. https://www.youtube.com/watch?v=htCj9exbGo0
14. Akansu, Ali N.; Agirman-Tosun, Handan "*Generalized Discrete Fourier Transform With Nonlinear Phase*", IEEE *Transactions on Signal Processing*, vol. 58, no. 9, pp. 4547-4556, Sept. 2010
15. https://www.analog.com/media/en/technical-documentation/dsp-book/dsp_book_Ch6.pdf

Appendix 1

```
from cmath import exp, pi
import matplotlib.pyplot as plt
from numpy.fft import fftfreq
%matplotlib notebook
import numpy as np
import mpmath as mp
import scipy
import scipy.stats as sp
import matplotlib.pyplot as plt
import subprocess
import cmath as cm
x=[]
y=[]


def dtft(x1,N):

    x=x1[0]
    j=cm.sqrt(-1)
    n=x1[1]
    X=[]

    w=np.linspace(-np.pi,np.pi,N)
    for i in range(0,N):
        w_tmp=w[i]
        X_tmp=0
        for k in range(0,len(x)):
            X_tmp+=(x[k]*np.exp(-n[k]*w_tmp*j))



        X.append(abs(X_tmp))
    print("lenX:",len(X))
    return w,X

def dft(x,N):
    samples=[]
    for k in range(N):
        list=[]
        a=0
        for n in range(N):
            a=x[n]*cm.exp(-2j*cm.pi*k*n*(1/N))
            list.append(a)
            a=0
        summation=sum(list)
        samples.append(summation)
    #print("\nThe samples are:-\n[")
    #for i in range(N):
    #    print(samples[i]," ")
```

```
    #print("]")
    return samples

def fft(x1):
    N = len(x1)
    if N <= 1: return x1
    even = fft(x1[0::2])
    odd =  fft(x1[1::2])
    T= [exp(-2j*pi*k/N)*odd[k] for k in range(N//2)]
    return [even[k] + T[k] for k in range(N//2)] + \
           [even[k] - T[k] for k in range(N//2)]


f = open("Sample.norm", "r")
#print(f.read())
Nval=128
#print("\nEnter the values of the signal x[n]:\n")
for i in range(Nval):
    x.append(float(f.readline()))
#print(x)

for i in range(Nval):
    y.append(i)
x1=[x,y]
w,X=dtft(x1,Nval)

samples=dft(x,Nval)

plt.figure()
plt.title('dtft')
plt.plot(w,X)
#plt.show()
plt.figure()
plt.title('dft')
plt.plot(w,samples,label='dft')

plt.figure()
plt.title('fft')
plt.plot(w,fft(x),label='fft')

plt.show()
```

Appendix 2

```
from cmath import exp, pi
import matplotlib.pyplot as plt
%matplotlib notebook
from numpy.fft import fftfreq
import numpy as np
import mpmath as mp
import math
import scipy
import scipy.stats as sp
import matplotlib.pyplot as plt
import subprocess
import cmath as cm
x=[]
y=[]

def dtft(x1,N):

    x=x1[0]
    j=cm.sqrt(-1)
    n=x1[1]
    X=[]

    w=np.linspace(-np.pi,np.pi,N)
    for i in range(0,N):
        w_tmp=w[i]
        X_tmp=0
        for k in range(0,len(x)):
            X_tmp+=(x[k]*np.exp(-n[k]*w_tmp*j))
```

```
        X.append(abs(X_tmp))
    #print("lenX:",len(X))
    return w,X

def dft(x,N):
    samples=[]
    for k in range(N):
        list=[]
        a=0
        for n in range(N):
            a=x[n]*cm.exp(-2j*cm.pi*k*n*(1/N))
            list.append(a)
            a=0
        summation=sum(list)
        samples.append(summation)
    #print("\nThe samples are:-\n[")
    #for i in range(N):
    #    print(samples[i]," ")
    #print("]")
    return samples

def fft(x1):
    N = len(x1)
    if N <= 1: return x1
    even = fft(x1[0::2])
    odd =  fft(x1[1::2])
    T= [exp(-2j*pi*k/N)*odd[k] for k in range(N//2)]
    return [even[k] + T[k] for k in range(N//2)] + \
           [even[k] - T[k] for k in range(N//2)]
```

```
f = open("05010117_1a.norm", "r")
for line in f:
    x.append(float(line))
length=len(x)
#print("\nlen:",len(x))
lval=math.ceil(math.log(length,2))
Pval=2**(lval)
#print("\nPval=",Pval)
pad=Pval-len(x)
for i in range(pad):
    x.append(0.0)
#print("\nx:\n",x)
for i in range(Pval):
    y.append(i)
x1=[x,y]
w,X=dtft(x1,Pval)
#print("w\n",w)
absl=[]
samples=dft(x,Pval)
for iter in fft(x):
    absl.append(iter)
#absl.append("%5.3f" % abs(f) for f in fft(x))
#print("absl\n",absl)



plt.figure()
plt.title('dtft')
plt.plot(w,X)
plt.show()
```

```
plt.figure()
plt.title('dft')
plt.plot(w,samples,label='dft')

plt.figure()
plt.title('fft')
plt.plot(w,fft(x),label='fft')

plt.show()
```

Appendix 3

```
from cmath import exp, pi
import matplotlib.pyplot as plt
%matplotlib notebook
from numpy.fft import fftfreq
import numpy as np
import mpmath as mp
import math
import scipy
import scipy.stats as sp
import matplotlib.pyplot as plt
import subprocess
import cmath as cm
x=[]
y=[]
x1=[]

def dtft(x1,N):

    x=x1[0]
    j=cm.sqrt(-1)
    n=x1[1]
    X=[]

    w=np.linspace(-np.pi,np.pi,N)

    for i in range(0,N):
        w_tmp=w[i]
        X_tmp=0
        for k in range(0,len(x)):
            X_tmp+=(x[k]*np.exp(-n[k]*w_tmp*j))


        X.append(abs(X_tmp))
    #print("lenX:",len(X))
    return w,X


def fft(x1):
    N = len(x1)
    if N <= 1: return x1
    even = fft(x1[0::2])
    odd =  fft(x1[1::2])
    T= [exp(-2j*pi*k/N)*odd[k] for k in range(N//2)]
    return [even[k] + T[k] for k in range(N//2)] + \
           [even[k] - T[k] for k in range(N//2)]

f = open("Aa.txt", "r")
outf = open("AaFft.txt","w")
```

```
current_line=1
Nval=256
count=0
fs=[]
for line in f:
   x1.append(int(line))
for i in range(len(x1)):
   if count <Nval:
      x.append(x1[i])
      count=count+1
   else:
      for iter in fft(x):
         fs.append(iter)

      x=[]
      x.append(x1[i])
      count=1
#print("\nfs:",fs)
# print("\nlen of fs:",len(fs))

#print("\nx till 12032:",x1[:12032])
#print("\nlen of last x:",len(x)," ",x)
for i in range(len(x1)):
   y.append(i)
x2=[x1,y]
w,X=dtft(x2[:12032],12032)

plt.figure()
#plt.subplot(3,1,1)
plt.title('dtft')
plt.plot(w,X)
plt.show()


#w=np.linspace(-np.pi,np.pi,12032)
plt.figure()
#plt.subplot(3,1,3)
plt.title('fft')
plt.plot(w,fs,label='fft')
```

Appendix 4

```
from cmath import exp, pi
import matplotlib.pyplot as plt
%matplotlib notebook
from numpy.fft import fftfreq
import numpy as np
import mpmath as mp
import math
import scipy
import scipy.stats as sp
import matplotlib.pyplot as plt
import subprocess
import cmath as cm
x=[]
y=[]
x1=[]


def dft(x,N):
    samples=[]
    for k in range(N):
        list=[]
        a=0
        for n in range(N):
            a=x[n]*cm.exp(-2j*cm.pi*k*n*(1/N))
            list.append(a)
            a=0
        summation=sum(list)
        samples.append(summation)
    #print("\nThe samples are:-\n[")
    #for i in range(N):
    #    print(samples[i]," ")
    #print("]")
    return samples

f = open("Aa.txt", "r")
outf = open("AaFft.txt","w")
current_line=1
Nval=256
count=0
dfs=[]
for line in f:
    x1.append(int(line))
for i in range(len(x1)):
    if count <Nval:
        x.append(x1[i])
        count=count+1
    else:
```

```
        samples=dft(x,Nval)
        for i in range(Nval):
            dfs.append(samples[i])

        x=[]
        x.append(x1[i])
        count=1
w=np.linspace(-np.pi,np.pi,len(dfs))
print("\nlenfs:",len(dfs))
plt.figure()
#plt.subplot(3,1,2)
plt.title('dft')
plt.plot(w,dfs,label='dft')
```

Appendix 5

```
from cmath import exp, pi
import matplotlib.pyplot as plt
%matplotlib notebook
from numpy.fft import fftfreq
import numpy as np
import mpmath as mp
import math
import scipy
import scipy.stats as sp
import matplotlib.pyplot as plt
import subprocess
import cmath as cm
x=[]
y=[]
x1=[]
def fft(x1):
    N = len(x1)
    if N <= 1: return x1
    even = fft(x1[0::2])
    odd =  fft(x1[1::2])
    T= [exp(-2j*pi*k/N)*odd[k] for k in range(N//2)]
    return [even[k] + T[k] for k in range(N//2)] + \
           [even[k] - T[k] for k in range(N//2)]

f = open("Aa.txt", "r")
Nval=256
count=0
fs=[]
overlap=0
sum=0
for line in f:
    x1.append(int(line))
# print("\nlen of x1:",len(x1))
j=overlap
length=len(x1)
i=0
while i < length:
#     print("\nlenx1:",length)
    if count < Nval:
        x.append(x1[j])
        count=count+1
        j=j+1
        i=i+1
    else:

#         print("\nx:\n",x)
```

```
#         sum+=len(x)
        for iteri in fft(x):
            fs.append(iteri)
        overlap=overlap+100
        j=overlap
        x=[]
        x.append(x1[j])
        count=1
        j=j+1
        i=i+1
        length=length+156


diff=Nval-len(x)
##limit of padding is 10
if diff <= 10:
    i=1
    while i <= diff:
        x.append(0)
        i=i+1
for iteri in fft(x):
    fs.append(iteri)


w=np.linspace(-np.pi,np.pi,len(fs))
plt.figure()
#plt.subplot(3,1,3)
plt.title('fft')
plt.plot(w,fs,label='fft')
```

Appendix 6

```
from pywt import wavedec
import matplotlib.pyplot as plt
%matplotlib notebook
x=[]

f = open("05010117_1a.norm", "r")
for line in f:
    x.append(line)
import pywt
coeffs = pywt.wavedec(x[:300],'db2',level=3)
cA3,cD3,cD2,cD1=coeffs

plt.figure()
plt.plot(cA3)
plt.title('Approx. coefficients at level3')
plt.show()

plt.figure()
plt.plot(cD3)
plt.title('Detailed coefficients at level3')

plt.figure()
plt.plot(cD2)
plt.title('Detailed coefficients at level2')

plt.figure()
plt.plot(cD1)
plt.title('Detailed coefficients at level1')
plt.show()
```