MATHEMATICS FOR COMPUTING – 2 (22MAT122)

FOURIER TRANSFORM AND ITS APPLIATIONS

In partial fulfillment for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

CSE(AI)

Centre for Computational Engineering and Networking

AMRITA SCHOOL OF ARTIFICIAL INTELLIGENCE

AMRITA VISHWA VIDYAPEETHAM

COIMBATORE - 641 112 (INDIA)

JULY – 2023

A THESIS

Submitted by

GROUP 14

GANESH SUNDHAR S (CB.EN.U4AIE22017)

KARTHIGAI SELVAM R (CB.EN.U4AIE22025)

SHRUTHIKAA V (CB.EN.U4AIE22047)

BHAVYA SAINATH (CB.EN.U4AIE22055)

AMRITA SCHOOL OF ARTIFICIAL INTELLIGENCE
AMRITA VISHWA VIDYAPEETHAM
COIMBATORE - 641 112

DECLARATION

We, Ganesh Sundhar S (CB.EN.U4AIE22017), Karthigai Selvam R (CB.EN.U4AIE22025), Shruthikaa V (CB.EN.U4AIE22047) and Bhavya Sainath (CB.EN.U4AIE22055), hereby declare that this thesis entitled "Fourier Transform and its applications", is the record of the original work done by me under the guidance of Dr.Neethu Mohan Assistant Professor, Centre for Computational Engineering and Networking, Amrita School of Artificial Intelligence, Coimbatore. To the best of my knowledge this work has not formed the basis for the award of any degree/diploma/ associate ship/fellowship/or a similar award to any candidate in any University.

Place: Coimbatore **Signature of student**
Date: 12-07-2023

TABLE OF CONTENTS

1 INTRODUCTION:

The Fourier series allows us to decompose a periodic function into its constituent frequencies. It states that any periodic function with a finite period (or a function defined on a finite interval and then periodically extended) can be expressed as an infinite sum of sine and cosine functions, each having a frequency that is a multiple of the fundamental frequency.

Mathematically, the Fourier series representation of a periodic function f(x) with period T is given by:

$$\boldsymbol{f(X) = \frac{a_o}{2} + \sum_{n=1}^{\infty} a_n \cos(nw_o t) + \sum_{n=1}^{\infty} b_n \sin(nw_o t)}$$

The coefficients a_0, a_n, and b_n can be determined using integration techniques and the orthogonality properties of sine and cosine functions. The coefficients represent the amplitudes of the respective sine and cosine components at each frequency.

Fourier Transform is an extension of The Fourier Series. The Fourier transform is a mathematical technique used to transform a function from the time or spatial domain to the frequency domain. The Fourier transform allows us to represent a function as a sum of complex exponential functions, which are characterized by their frequencies. Unlike the Fourier series, which deals with periodic functions, the Fourier transform is applicable to a non-periodic ones.

$$\mathbf{F(\omega)} = \int [f(t) * e^{(-i\omega t)}]\, dt$$

The Fourier transform essentially decomposes a function into its constituent frequencies. It provides information about the magnitudes and phases of the different frequency components present in the original function. To extract meaningful information from signals, it is necessary to analyze their frequency content. By analyzing the Fourier transform of a signal, we can extract valuable insights into its frequency content and characteristics.

2 IMAGE AND AUDIO CODECS USING DCT:

2.1 DISCRETE COSINE TRANSFORM:

Discrete Cosine Transform (DCT) is a process of finding the Fourier Transform of an signal using real cosine bases. The Fourier Transform (FT) is a process that represents a aperiodic continuous time domain signal in its frequency domain. DCT is used because the Fourier Transform of some real functions are complex. As complex functions are harder to represent and compute, we go for Fourier Transform with real cosine bases that give only real dct coefficient values. We can use this for a lot of real life applications.

We now focus on using DCT for image and audio compression. These create the image and audio codec formats namely .jpeg and .mp3.

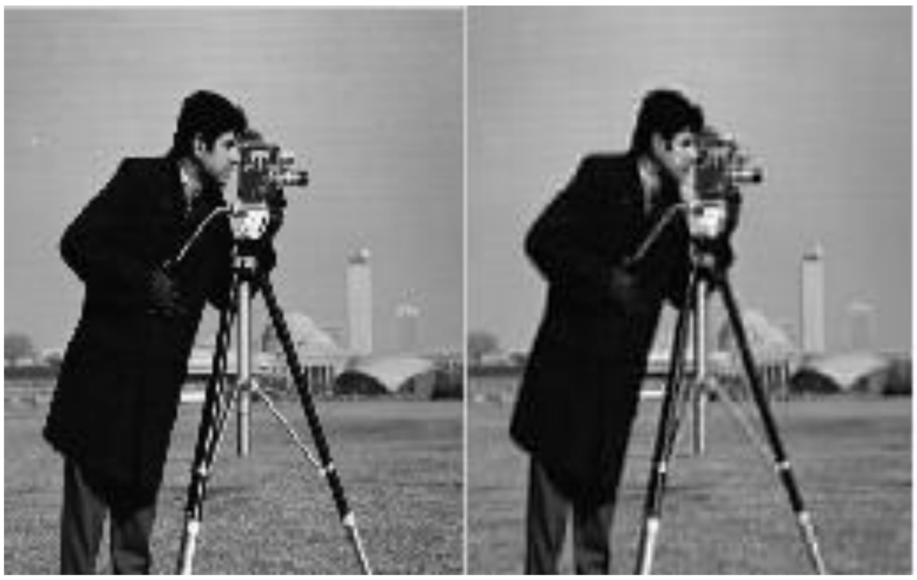


Figure 1 : Lossy image compression

The DCT image compression algorithm is a lossy image compression algorithm that drastically reduces the size of the image without significantly reducing the details captured. The size of the image would be the same, but it would consume much lesser space than the original one. This gave rise to various image and audio codec formats that we know today which take up much lesser space than usual.

2.2 PROCEDURE :

a) Image Compression :

- We first create a function called ‘ImgCompress’. We can call this function and it will open a window to choose the file. From there, we can choose the file to be compressed.
- We first get the file name and path name of the file using the ‘uigetfile’ function. We give ‘*.*’ inside the ‘uigetfile’ function so that it allows us to chose file of any format.
- The label would be ‘Select the image to compress’.
- We then read the image from the respective path and file name using the ‘imread’ function and store it in the ‘img’ variable.
- From here, there are two paths. It depends on whether the image is a color image or a black and white image. We check this using a ‘if-else’ conditional statement.
- Case 1 (Color Image) :
    - We first split the image into its respective RGB components using the ‘imsplit’ function.
    - We then calculate the DCT of all the 3 components seperately after converting them to double using the ‘dct2’ function.
    - The ‘dct2’ functions uses a much faster and efficient algorithm to calculate the dct coefficients than ‘dct’ function. So, we use that to calculate the dct.
- We then make all the values lesser than 15 as 0 in the dct matrix of all those components separately. This is done in order to remove the less important values to finally reduce the image size.
- We then calculate the Inverse Discrete Cosine Transform (IDCT) using the ‘idct2’ function to change the coefficient matrix back to the image matrix.
- We then convert them back to unsigned integer 8 bit (uint8) as image is of that form.
- We then concatenate the 3 different components into a new final image using the ‘cat’ function.

- Case 2 (B and W image) :
    - We convert the image values to double initially. This is done because the image contains values in the uint8 (unsigned integer 8 bit) format.
    - We then calculate the DCT of that matrix using the ‘dct2’ function.
    - We then make the values lesser than 15 from the DCT coefficient matrix to 0. This is done in order to remove the less important values to finally reduce the image size.

    - We got the value 15 using trial and error method. If we go above 15, the size will be much smaller but the compression loss increases. If we go below 15, the image size does not get reduced much.
    - We then calculate the IDCT using the ‘idct2’ function to convert the coefficient matrix back to the image matrix.
    - We then reconstruct the image by converting it to uint8 format.

- Finally, we write the output image to the directory from where we took the input. The naming convention would be Image_Name + ‘red’ where ‘red’ stands for ‘reduced’.
- We also change the format of the output image to JPEG for input image of any format.
- We achieve this using ‘strcat’ and ‘extractBefore’ matlab functions. We use the ‘extractBefore’ function to get the file name alone without the file type. We then combine the path, file name, ‘red’, ‘.jpeg’ using the ‘strcat’ function.
- We then give the new image and this file location inside the ‘imwrite’ function in order to write the output image.

b) Audio Compression :

- We first define the function with name ‘AudCompress’. We can call this function and it will open a window to choose the file. From there, we can choose the file to be compressed.
- We get the file name and path from the ‘uigetfile’ function. We pass ‘*.*’ as the file type to the ‘uigetfile’ function so that it gets file of any format and type as input.
- The label would be ‘Select the audio file to be compressed’.
- We then, open the audio file using the ‘audioread’ function with the path and file name that we got. The ‘audioread’ function also returns the sampling rate of the particular audio file selected in Hertz(Hz).
- We then calculate the DCT of the audio that we got using the ‘dct2’ function.
- We then remove the values lesser that 0.2 from the dct coefficient matrix. We got the value 0.2 using trial and error method. If the value is greater that 0.2, the size reduces but the compression losses are high. If it is lesser than 0.2, the audio size is not much reduced.
- We then calculate the IDCT of the coefficient matrix using the ‘idct2’ function.
- We then write the audio file into the same directory from which we got the input.
- The naming convention would be File_Name + ‘red’ where ‘red’ stands for ‘reduced’.
- We also change the format of the output audio to MP3 for input audio of any format.
- We achieve this using ‘strcat’ and ‘extractBefore’ matlab functions. We use the ‘extractBefore’ function to get the file name alone without the file type. We then combine the path, file name, ‘red’, ‘.mp3’ using the ‘strcat’ function.
- We then give the new audio and this file location along with the sampling rate that we got earlier inside the ‘audiowrite’ function in order to write the output audio to the directory.

2.3 SOURCE CODES :

a) <u>Image Compression</u> :

```
function ImgCompress

[file,path] = uigetfile('*.*','Select the image to compress');
name = strcat(path,file);
img = imread(name);

if(size(img,3) == 3)
    [R,G,B] = imsplit(img);

    R = dct2(double(R));

    R(abs(R) < 15) = 0;

    G = dct2(double(G));

    G(abs(G) < 15) = 0;

    B = dct2(double(B));

    B(abs(B) < 15) = 0;

    R = uint8(idct2(R));
    G = uint8(idct2(G));
    B = uint8(idct2(B));

    New_img = cat(3,R,G,B);

else
    Mat = dct2(double(img));
    Mat(abs(Mat) < 15) = 0;

    New_img = uint8(idct2(Mat));
end

imwrite(New_img,strcat(path,extractBefore(file,'.'),'
red','.jpeg'));

end
```

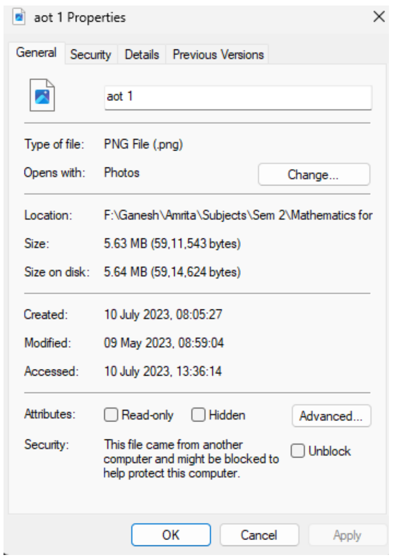
b) <u>Audio Compression</u> :

```
function AudCompress

[file,path] = uigetfile('*.*','Select the audio to compress');
name = strcat(path,file);
[Aud,rate] = audioread(name);

Mat = dct2(Aud);

Mat(abs(Mat) < 0.02) = 0;

New_Aud = idct2(Mat);

audiowrite(strcat(path,extractBefore(file,'.'),'
red','.mp3'),New_Aud,rate);

end
```

2.4 RESULTS :

a) Image Compression :

Case 1 :


aot 1 Properties

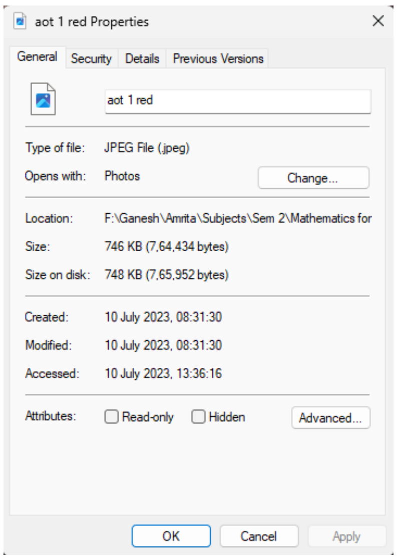
aot 1

Type of file: PNG File (.png)
Opens with: Photos

Location: F:\Ganesh\Amrita\Subjects\Sem 2\Mathematics for
Size: 5.63 MB (59,11,543 bytes)
Size on disk: 5.64 MB (59,14,624 bytes)

Created: 10 July 2023, 08:05:27
Modified: 09 May 2023, 08:59:04
Accessed: 10 July 2023, 13:36:14

Attributes: ☐ Read-only ☐ Hidden
Security: This file came from another computer and might be blocked to help protect this computer. ☐ Unblock


aot 1 red Properties

aot 1 red

Type of file: JPEG File (.jpeg)
Opens with: Photos

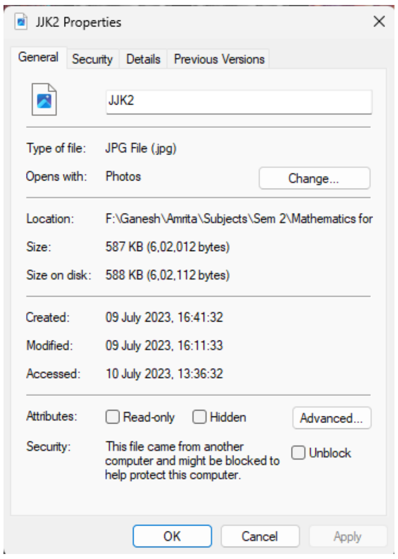
Location: F:\Ganesh\Amrita\Subjects\Sem 2\Mathematics for
Size: 746 KB (7,64,434 bytes)
Size on disk: 748 KB (7,65,952 bytes)

Created: 10 July 2023, 08:31:30
Modified: 10 July 2023, 08:31:30
Accessed: 10 July 2023, 13:36:16

Attributes: ☐ Read-only ☐ Hidden

We can see that high compression has been achieved in conversion from '.png' to '.jpeg' format.

Case 2 :


JJK2 Properties

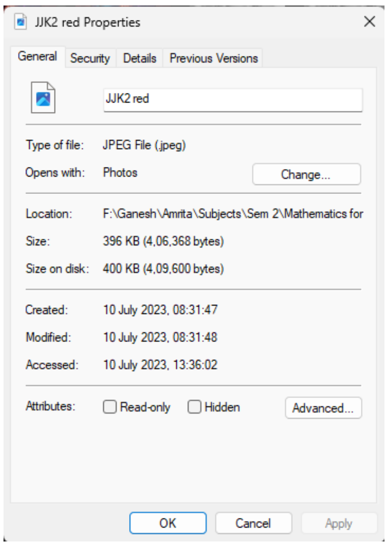
JJK2

Type of file: JPG File (.jpg)
Opens with: Photos

Location: F:\Ganesh\Amrita\Subjects\Sem 2\Mathematics for
Size: 587 KB (6,02,012 bytes)
Size on disk: 588 KB (6,02,112 bytes)

Created: 09 July 2023, 16:41:32
Modified: 09 July 2023, 16:11:33
Accessed: 10 July 2023, 13:36:32

Attributes: ☐ Read-only ☐ Hidden
Security: This file came from another computer and might be blocked to help protect this computer. ☐ Unblock


JJK2 red Properties

JJK2 red

Type of file: JPEG File (.jpeg)
Opens with: Photos

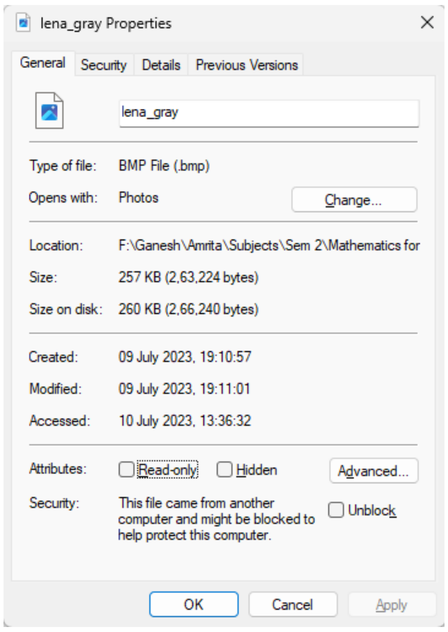
Location: F:\Ganesh\Amrita\Subjects\Sem 2\Mathematics for
Size: 396 KB (4,06,368 bytes)
Size on disk: 400 KB (4,09,600 bytes)

Created: 10 July 2023, 08:31:47
Modified: 10 July 2023, 08:31:48
Accessed: 10 July 2023, 13:36:02

Attributes: ☐ Read-only ☐ Hidden

We can see that not much compression is achieved in conversion from '.jpg' to '.jpeg' as '.jpg' is already a compressed file format.

Case 3 :


lena_gray Properties

General | Security | Details | Previous Versions

lena_gray

Type of file: BMP File (.bmp)

Opens with: Photos

Location: F:\Ganesh\Amrita\Subjects\Sem 2\Mathematics for

Size: 257 KB (2,63,224 bytes)

Size on disk: 260 KB (2,66,240 bytes)

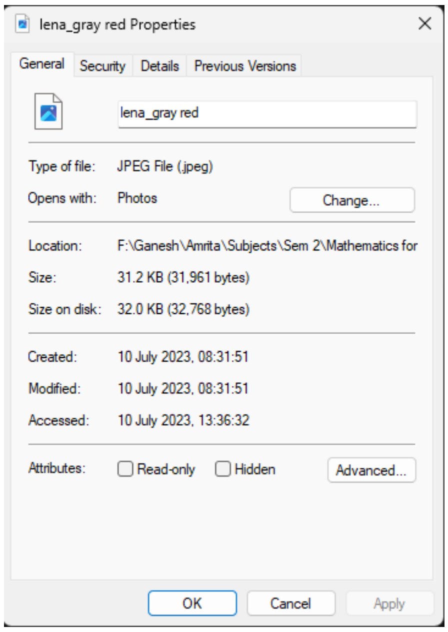
Created: 09 July 2023, 19:10:57

Modified: 09 July 2023, 19:11:01

Accessed: 10 July 2023, 13:36:32

Attributes: ☐ Read-only ☐ Hidden

Security: This file came from another computer and might be blocked to help protect this computer. ☐ Unblock


lena_gray red Properties

General | Security | Details | Previous Versions

lena_gray red

Type of file: JPEG File (.jpeg)

Opens with: Photos

Location: F:\Ganesh\Amrita\Subjects\Sem 2\Mathematics for

Size: 31.2 KB (31,961 bytes)

Size on disk: 32.0 KB (32,768 bytes)

Created: 10 July 2023, 08:31:51

Modified: 10 July 2023, 08:31:51

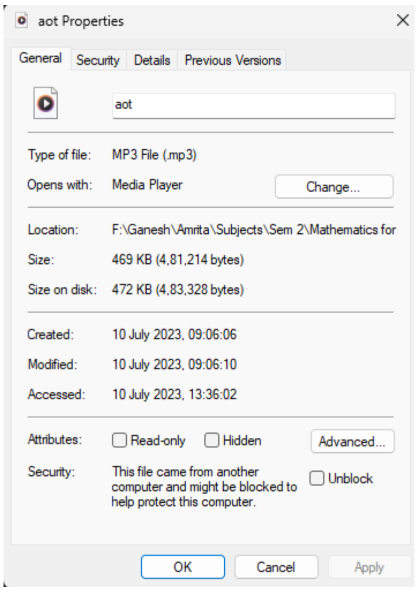
Accessed: 10 July 2023, 13:36:32

Attributes: ☐ Read-only ☐ Hidden

Here also, we can see that high compression is achieved.

b) **Audio Compression** :

Case1 :


aot Properties

General | Security | Details | Previous Versions

aot

Type of file: MP3 File (.mp3)

Opens with: Media Player

Location: F:\Ganesh\Amrita\Subjects\Sem 2\Mathematics for

Size: 469 KB (4,81,214 bytes)

Size on disk: 472 KB (4,83,328 bytes)

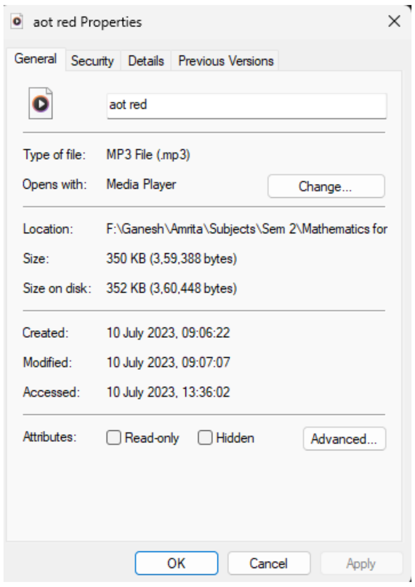
Created: 10 July 2023, 09:06:06

Modified: 10 July 2023, 09:06:10

Accessed: 10 July 2023, 13:36:02

Attributes: ☐ Read-only ☐ Hidden

Security: This file came from another computer and might be blocked to help protect this computer. ☐ Unblock


aot red Properties

General | Security | Details | Previous Versions

aot red

Type of file: MP3 File (.mp3)

Opens with: Media Player

Location: F:\Ganesh\Amrita\Subjects\Sem 2\Mathematics for

Size: 350 KB (3,59,388 bytes)

Size on disk: 352 KB (3,60,448 bytes)

Created: 10 July 2023, 09:06:22

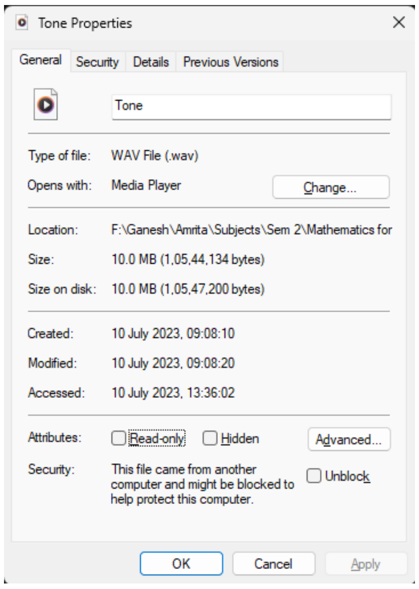
Modified: 10 July 2023, 09:07:07

Accessed: 10 July 2023, 13:36:02

Attributes: ☐ Read-only ☐ Hidden

Here, we can see that not much compression is achieved. This is because the input audio is in '.mp3' format which is already compressed.

Case 2 :


Tone Properties

General | Security | Details | Previous Versions

Tone

Type of file: WAV File (.wav)

Opens with: Media Player

Location: F:\Ganesh\Amrita\Subjects\Sem 2\Mathematics for

Size: 10.0 MB (1,05,44,134 bytes)

Size on disk: 10.0 MB (1,05,47,200 bytes)

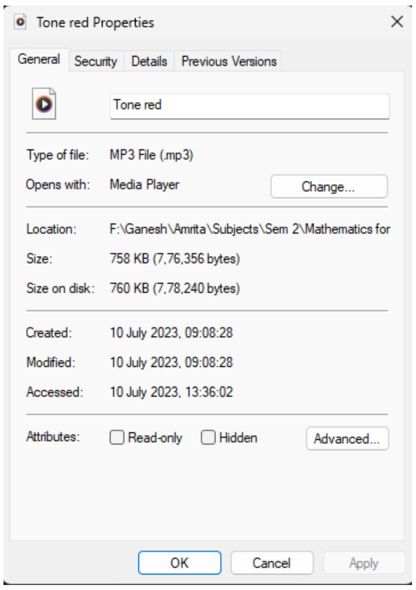
Created: 10 July 2023, 09:08:10

Modified: 10 July 2023, 09:08:20

Accessed: 10 July 2023, 13:36:02

Attributes: ☐ Read-only ☐ Hidden

Security: This file came from another computer and might be blocked to help protect this computer. ☐ Unblock


Tone red Properties

General | Security | Details | Previous Versions

Tone red

Type of file: MP3 File (.mp3)

Opens with: Media Player

Location: F:\Ganesh\Amrita\Subjects\Sem 2\Mathematics for

Size: 758 KB (7,76,356 bytes)

Size on disk: 760 KB (7,78,240 bytes)

Created: 10 July 2023, 09:08:28

Modified: 10 July 2023, 09:08:28

Accessed: 10 July 2023, 13:36:02

Attributes: ☐ Read-only ☐ Hidden

Here, we can see that high compression is achieved. This is because the input file format is '.wav' and the output file format is '.mp3'.

3 SIGNAL FILTERING

To filter a signal which to reduce and smooth out high-frequency noise associated with a measurement such as flow, pressure, level or temperature.

Signal filtering in Fourier transform involves removing unwanted frequency components to isolate a specific frequency range. This is done by applying a filter function to the Fourier transform of the signal.
In Fourier transform, a signal is represented as a sum of sine waves of different frequencies. By applying a filter function to the Fourier transform, we can remove unwanted frequency components from the signal. This is done by setting the values of the Fourier coefficients outside of the desired frequency range to zero. The filtered signal can then be obtained by taking the inverse Fourier transform of the filtered Fourier coefficients.

One of the importance of signal filtering is to tells the relative amplitude of frequencies present in the signal. The band pass of the signal will need to be same as the signal frequencies range.

3.1 ANALYSING THE IMPORTANCE OF DFT AND FFT:

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.

A fast Fourier transform (FFT) is an algorithm that computes the discrete Fourier transform (DFT) of a sequence, or its inverse (IDFT). Fourier analysis converts a signal from its original domain (often time or space) to a representation in the frequency domain and vice versa.

3.2 CODE:

```
% Input signal parameters
% Sampling frequency (Hz)
Fs = 1000;
% Sampling period (s)
T = 1/Fs;
% Signal length
L = 1000;
% Time vector
t = (0:L-1)*T;
% Create input signal (we are generating two sin waves and adding together to form the input signal)
%f1 and f2 represents the frequency of first and second sin waves in (Hz)
f1 = 50;
f2 = 120;
% A1 and A2 represents the amplitude of first and second sin waves
A1 = 0.7;
A2 = 1.2;
x = A1*sin(2*pi*f1*t) + A2*sin(2*pi*f2*t);

% here we are plotting the input signal
subplot(2,1,1);
plot(t, x);
title('Input Signal');
xlabel('Time (s)');
ylabel('Amplitude');

% Computing Fourier series coefficients of the input signal
N = length(x);
% by applying the 'fft' function we will obtain the frequency domain
%representaton of the signal
X = fft(x)/N;
f = Fs*(0:(N/2))/N;

% here we are plotting the amplitude spectrum of the input signal
subplot(2,1,2);
stem(f, 2*abs(X(1:N/2+1)));
title('Amplitude Spectrum');
xlabel('Frequency (Hz)');
ylabel('Magnitude');
% Filter the signal by removing unwanted frequencies
% Remove the frequencies below 80 Hz
cutoff_frequency = 80;
```

```
X_filtered = X;
X_filtered(f < cutoff_frequency) = 0;

% Reconstruct the filtered signal using inverse Fourier transform
x_filtered = ifft(X_filtered)*N;

% Plot the filtered signal
figure;
plot(t, x_filtered);
title('Filtered Signal');
xlabel('Time (s)');
ylabel('Amplitude');
```

INPUT PARAMETERS:

This section defines the sampling frequency, sampling period, signal length, and time vector.

INPUT SIGNAL:

This section creates an input signal that is the sum of two sine waves with different frequencies and amplitudes.

PLOT OF INPUT SIGNAL:

This section plots the input signal in the time domain.

COMPUTING FORIER SERIES:

This section involves computing fourier series coefficients of input signal to fast fourier transform.

PLOT OF AMPLTUDE SPECTRUM:

This section plots the amplitude spectrum of the input signal in the frequency domain.

FITERING:

This section filters the input signal by setting to zero all Fourier coefficients corresponding to frequencies below a certain cutoff frequency.

RECONSTRUCT FILTER SIGNAL:

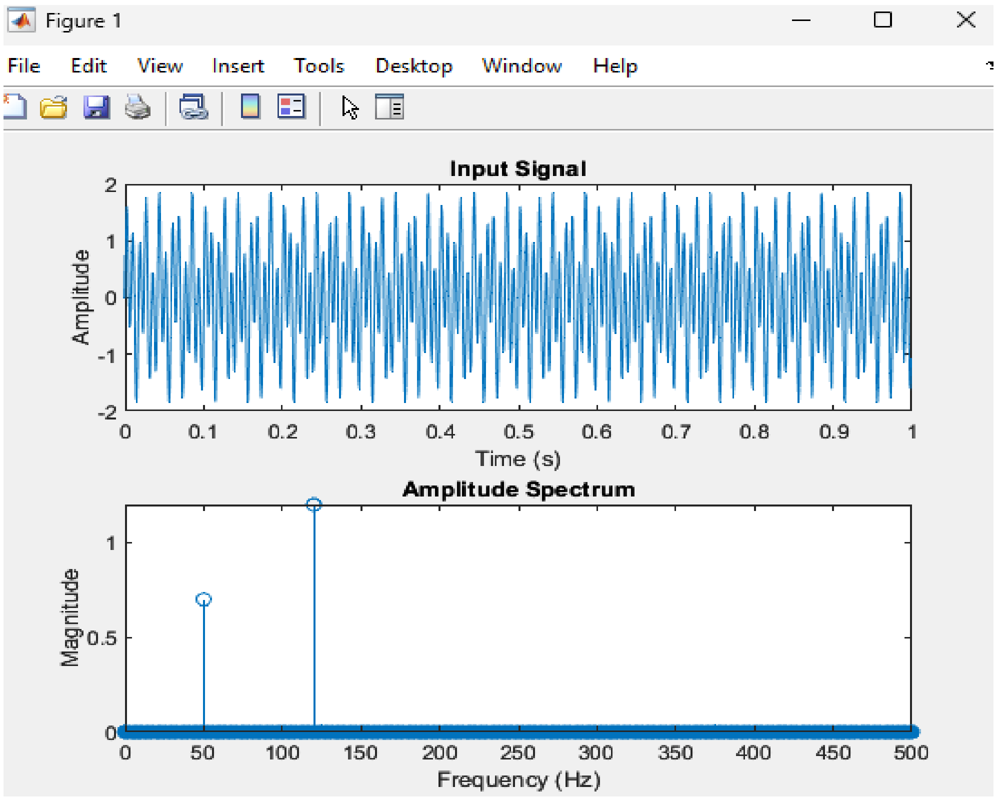
This section reconstructs the filtered signal using the Inverse Fast Fourier Transform (IFFT) algorithm.

PLOTTING:

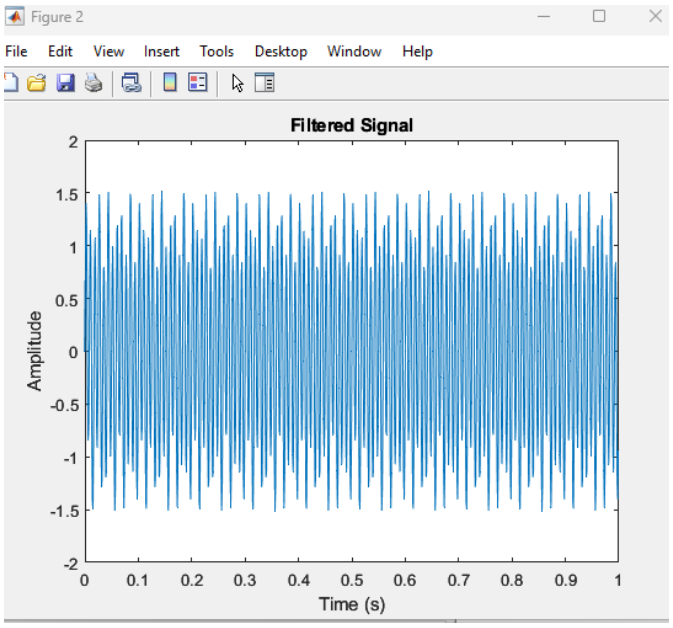
This section plots the filtered signal in the time domain.

3.3 RESULTS :

INPUT SIGNAL :



OUTPUT SIGNAL:



4 CONVOLUTION AND DECONVOLUTION USING FAST FOURIER TRANSFORM (FFT)

As The Fourier Transform allows us to decompose a signal into its constituent frequencies, and analyze the spectral properties as the Fourier transforms of the original white noise and the filtered noise signals are computed.

This study investigates how bandpass filtering methods can be used to recover the impulse response (IR) from filtered white noise signals. By examining the spectrum characteristics of the filtered white noise and separating the IR from it, the major goal is to characterise the response of a system.

Getting the IR out of the filtered noise signal is the primary goal of the first section of the code.Fast Fourier transform (FFT) methods are used to implement the deconvolution process.After being zeropadded to the same length, the FFTs of the filtered noise signal and the original white noise signal are calculated. The frequency responsiveness of the system is calculated by dividing the FFT of the filtered noise by the FFT of the original noise.
The estimated IR is obtained using the frequency response's inverse FFT.

A random number generator is used to create one second of white noise to start the experiment. When the white noise is normalised, its absolute maximum value is set to 1. The next step is to create a bandpass filter with a core frequency of 1000 Hz. The windowed sinc function is used to calculate the filter's impulse response, and the white noise signal is convolved with the filter to produce the filtered noise signal.

Finally, the extracted IR is compared with the original IR used in the bandpass filter design. The time-domain plots of both IRs are displayed for visual comparison. This step allows for evaluating the accuracy of the deconvolution process and the effectiveness of the bandpass filter in capturing the system's behavior.

The impulse response of a system is the output signal that results from applying an impulse input to the system. An impulse input is a very short-duration signal that has an amplitude of 1 at time zero and 0 everywhere else. The impulse response represents the response of the band-pass filter to an impulse input. It characterizes how the filter behaves and influences the input signal when an impulse is applied.

The impulse response provides information about the filter's frequency response, time-domain behavior, and its ability to pass or attenuate specific frequency components. By analyzing the shape and magnitude of the impulse response, one can gain insights into the filter's properties, such as its bandwidth, center frequency, and frequency selectivity.

In general, the impulse response describes how a system or filter responds to a brief input stimulus, and it is a fundamental concept in signal processing and system analysis.

4.1 ANALYSING THE SIGNIFICANCE OF DFT:

The DFT is an abbreviation of the Discrete Fourier Transform. So the DFT is just a type of Fourier Transform for the discrete-time x (n) instead of the continuous analog signal x (t). The Fourier Transform equation is as follow:

$$X(\omega) = \sum_{n=-\infty}^{\infty} x(n)e^{j\omega n}$$

From the equation, the main function of the Fourier Transform is to transform the variable from the variable n into the variable ω, which means transforming the signals from the time domain into the frequency domain.

4.2 METHODOLOGY:

Sampling frequency is set to analyse the signal which basically, refers to the number of samples taken per second from an analog signal to convert it into a digital representation.

In simple terms, the sampling frequency determines how frequently the amplitude of an analog signal is measured or to create discrete data points in the digital domain. It is typically expressed in Hertz (Hz) and represents the number of samples collected per second.

The wave signal we take in the experiment is white noise, which is a type of random signal that contains equal energy or power at all frequencies within a specified frequency range. Each sample of white noise is statistically independent of all other samples. This means that there is no correlation or predictable relationship between consecutive samples or any other group of samples. When the amplitude spectrum of white noise is plotted, it appears as a flat line, indicating that all frequencies have equal power.

After the generating the random wave for 1 second to analyse with given sampling frequency, it is normalised to make it’s amplitudes in the range [-1,1].

The central frequency is set for the wave along with it’s upper and lower cut-off frequency. By adjusting the values of central frequency, we can change the centre frequency of the band-pass filter. Similarly, by modifying the relationships between central frequency, lower limit frequency, and upper limit frequency, we can control the bandwidth and shape of the filter's passband.

The number filter co-efficients are set ensured it is a even number. The number of filter coefficients refers to the total number of values used to represent the impulse response of a filter. Each coefficient represents the contribution or weight given to a specific input sample and determines the filter's behavior in shaping the input signal. The filter order is closely related to the number of coefficients and provides information about the complexity and length of the filter and reflects about the complexity and length of the filter.

The filter coefficients are calculated using a windowed sinc function. The sin function is applied to the angular frequencies. Adjustment ensures that the filter has a zero DC gain, meaning that it does not introduce a constant component or offset to the input signal within the passband.

The fftconv function is used to convolve the white noise signal with the filter coefficients. Convolution is a mathematical operation that combines the input signal with the filter's impulse response to produce the filtered output signal.

The fftdec function performs deconvolution, which is the inverse operation of convolution. It extracts the impulse response from the filtered noise signal and the original input signal. The deconvolution process aims to recover the original impulse response of the filter that was applied to the noise signal.

The fft function is used to compute the FFT of the white noise signal. This calculates the frequency spectrum of the white noise. Frequency axis corresponding to the computed spectrum is computed. It creates a linearly spaced vector from 0 to 1 with number of points on FFT/2 + 1 points.

The fft function is used again to compute the FFT of the filtered white noise signal.This calculates the frequency spectrum of the filtered white noise.

The fftdec function essentially performs deconvolution using the FFT algorithm. It takes a system output and an original input, both assumed to be row vectors. It returns the impulse response of the system that relates the input and output signals. The fftconv function essentially performs convolution using the FFT algorithm. It takes an input signal and an impulse response , both assumed to be row vectors. It returns the convolved output signal (y).

4.3 CODE:

```
clc; clear;close all;
Fs=44100;
% Generate 1 second of white noise
rand('state',sum(100 * clock));
noise = randn(1, Fs);
noise = noise / max(abs(noise));
% BAND PASS FILTER
fc=1000;                % Central Frequency
f1=fc/2^0.5;
f2=fc*2^0.5;
w1=2*pi*f1/Fs;
w2=2*pi*f2/Fs;
wc=2*pi*fc/Fs;
ncof=200;               % number of coeficients,
if rem(ncof, 2)         % must be even
   ncof=ncof+1;
end
n=0:ncof;
M=length(n)-1;          % filter order
% Filter
h=sin(w2*(n-(M/2)))./((n-(M/2))*pi)-sin(w1*(n-(M/2)))./((n-(M/2))*pi);
h(M/2+1)=(w2-w1)/pi;
% Filter white noise
filternoise = fftconv(noise,h);
% Extract IR from filtered noise
hnew = fftdec(filternoise,noise);
% White noise spectrum
nf=4096; %number of point in DTFT
Yw = fft(noise,nf);
fw = Fs/2*linspace(0,1,nf/2+1);
% Filtered white noise spectrum
Yf = fft(filternoise,nf);

figure
plot(fw, abs(Yw(1:nf/2+1)));
title('White noise spectrum');
xlabel('Frequency (Hz)');
ylabel('Magnitude');
xlim([500 10000]);
figure
plot(fw, abs(Yf(1:nf/2+1)));
```

```
title('Filtered white noise spectrum');
xlabel('Frequency (Hz)');
ylabel('Magnitude');
xlim([100 10000]);

figure
plot(0:length(h)-1, h);
title('Original IR - BandPass Filter');
xlabel('Samples');
ylabel('Amplitude');
xlim([0 length(h)]);

figure
plot(0:length(hnew)-1, hnew);
title('Extracted IR');
xlabel('Samples');
ylabel('Amplitude');
xlim([0 length(h)]);

function [ h ] = fftdec( out,in )
no = length(out);
ni = length(in);
nfft = 2^nextpow2(no+ni-1);
ozp = [out, zeros(1,nfft-no)];
izp = [in, zeros(1,nfft-ni)];
O = fft(ozp);
I = fft(izp);
Y = O ./ I;
h = real(ifft(Y));
end

function [ y ] = fftconv( x, h )
nx = length(x);
nh = length(h);
nfft = 2^nextpow2(nx+nh-1);
xzp = [x, zeros(1,nfft-nx)];
hzp = [h, zeros(1,nfft-nh)];
X = fft(xzp);
H = fft(hzp);
Y = H .* X;
y = real(ifft(Y));
end
```

4.4 RESULTS:

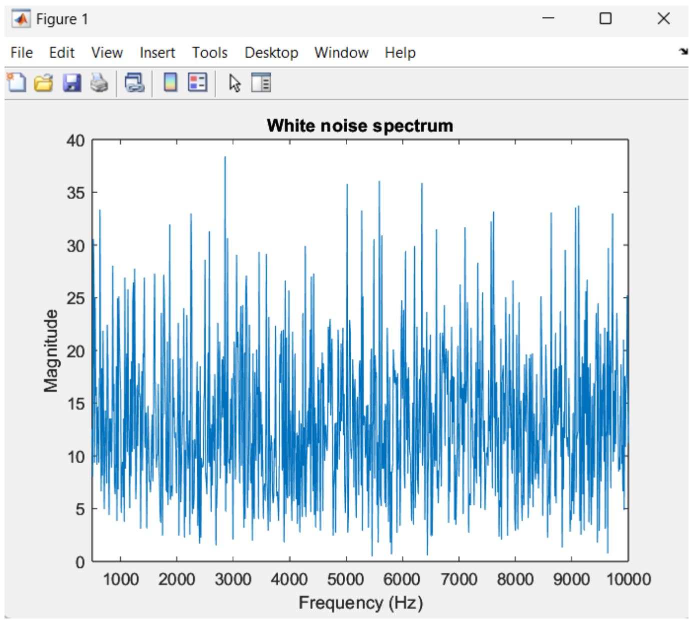


Figure 1
File Edit View Insert Tools Desktop Window Help
White noise spectrum
Magnitude
Frequency (Hz)
0
5
10
15
20
25
30
35
40
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000

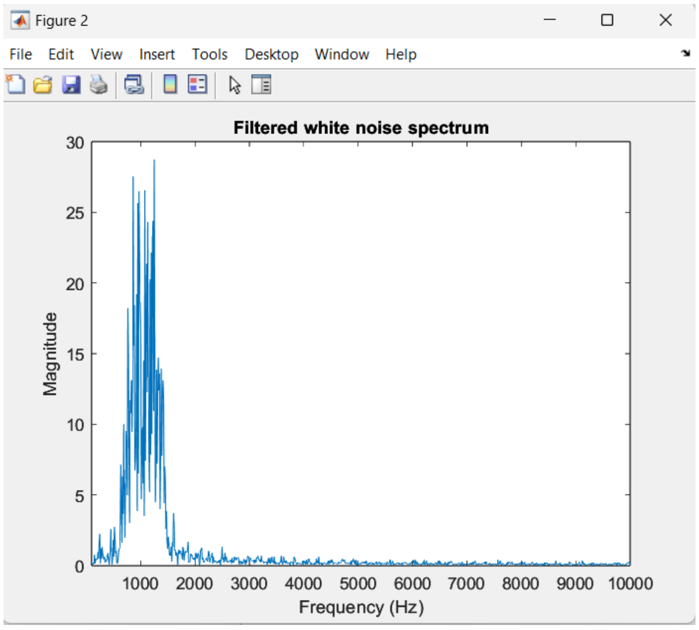


Figure 2
File Edit View Insert Tools Desktop Window Help
Filtered white noise spectrum
Magnitude
Frequency (Hz)
0
5
10
15
20
25
30
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000

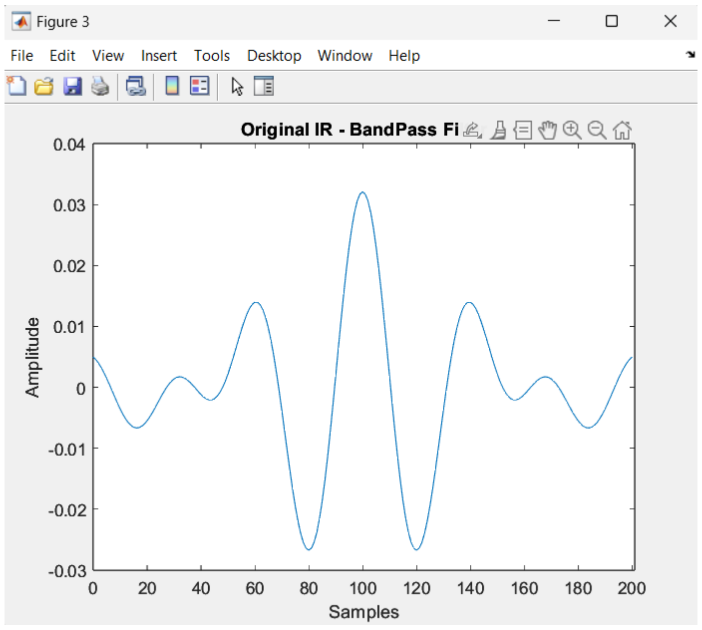


Figure 3
File Edit View Insert Tools Desktop Window Help
Original IR - BandPass Fi
Amplitude
Samples
-0.03
-0.02
-0.01
0
0.01
0.02
0.03
0.04
0
20
40
60
80
100
120
140
160
180
200

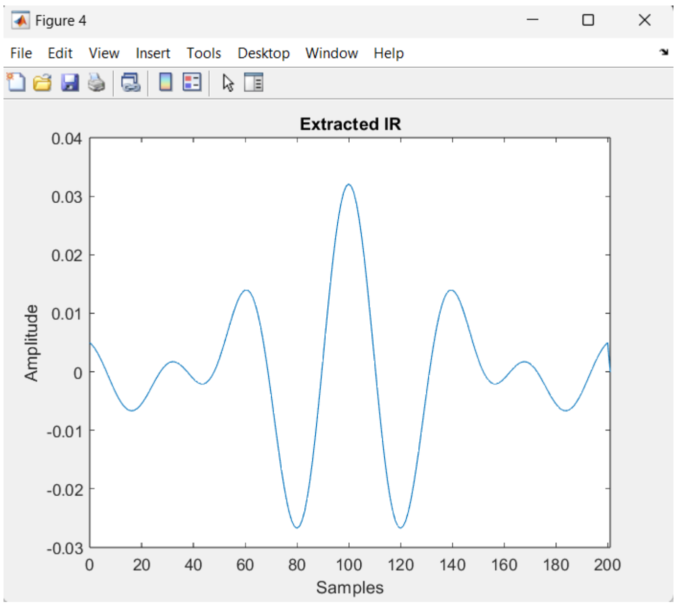
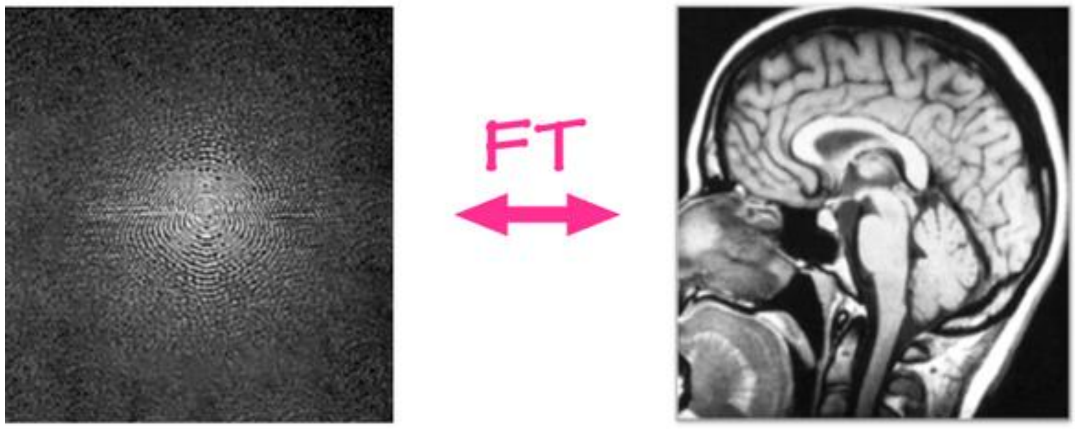


Figure 4
File Edit View Insert Tools Desktop Window Help
Extracted IR
Amplitude
Samples
-0.03
-0.02
-0.01
0
0.01
0.02
0.03
0.04
0
20
40
60
80
100
120
140
160
180
200

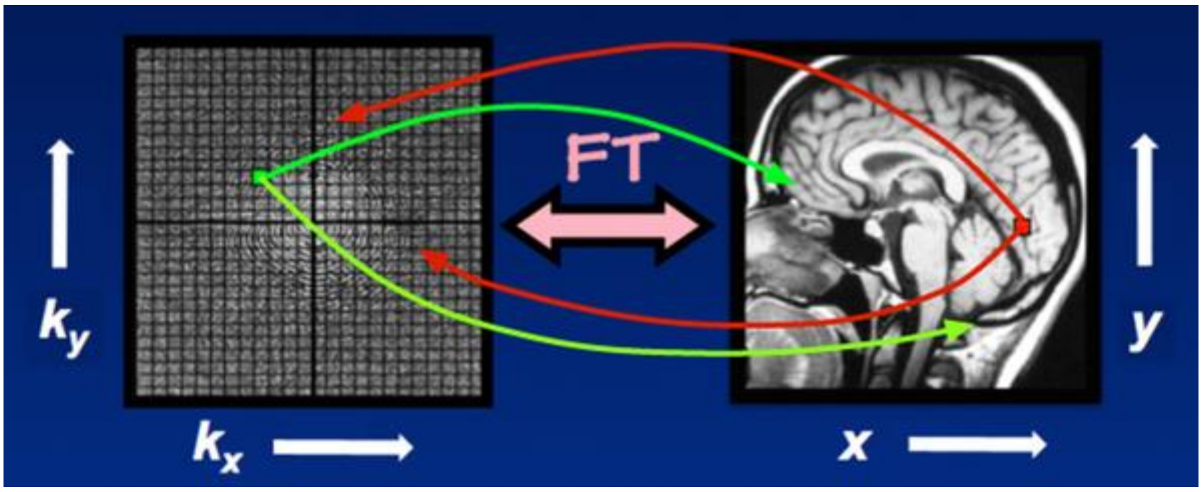
5 MEDICAL IMAGING USING FAST FOURIER TRANSFORM

The popular graphic representation of k-space as a "galaxy" adds to the mystery. Each "star" in k-space is simply a data point extracted from the MR signal. The brightness of each star symbolizes its unique spatial frequency's relative contribution to the final image.



Although the k-space “galaxy” and MRI image appear quite different, they contain identical information about the scanned object.The two representation may be converte to one another using an advanced mathematical procedure the ***Fourier Transform.***

The cells of k-space are often represented on a rectangular grid with the primary axes kx and ky. The kx and ky axes of k-space correspond to the image's horizontal (x-) and vertical (y-) axes. The k-axes, on the other hand, represent spatial frequencies rather than places in the x and y directions. Individual points (kx,ky) in k-space do not correlate to individual pixels (x,y) in the image one-to-one. Every pixel in the final image is represented by a k-space point, which comprises spatial frequency and phase information. In contrast, each pixel in the image corresponds to every point in k-space. As a result, the k-space representation of the MR picture is similar to the diffraction patterns produced by x-ray crystallography, optics, or holography.



5.1 FAST FOURIER TRANSFORM (FFT) :

The FFT is a fast algorithm used to compute the Discrete Fourier Transform (DFT) of a sequence or signal. The DFT represents a discrete-time signal as a sum of complex sinusoidal functions, which allows us to analyze the signal's frequency content.

X[k] = Σ [x[n] * e^(-j * 2π * k * n / N)]

- X[k]: This represents the frequency domain representation of the signal. It denotes the complex amplitude at frequency index k.
- x[n]: This represents the discrete-time sequence or signal in the time domain. It is the input sequence to the FFT.
- exp(-j * 2π * k * n / N): This term represents a complex sinusoidal function with a frequency index k. It is raised to the power of -j (the imaginary unit multiplied by π), and multiplied by 2π and k and n divided by N. This term is used to decompose the input signal into its frequency components.
- where k is the frequency index (ranging from 0 to N-1) and j represents the imaginary unit.

The FFT algorithm breaks down the computation of the DFT into smaller sub-problems, reducing the time complexity from O(N^2) to O(N log N). This makes it much more efficient for practical applications.

The key idea behind the FFT is the concept of "decimation in time" or "divide and conquer." It recursively splits the sequence into even and odd indexed elements and applies a butterfly operation to combine the results. This process is repeated until the sequence is reduced to its individual components, resulting in the final DFT.

5.2 INVERSE FAST FOURIER TRANSFORM (IFFT) :

The IFFT is the inverse operation of the FFT. It takes a frequency domain representation (obtained through the FFT) and reconstructs the original signal in the time domain.

$$\mathbf{x[n] = (1/N) * \Sigma\ [X[k] * e\hat{}(j * 2\pi * k * n / N)]}$$

- $x[n]$: This represents the reconstructed time domain signal obtained through the IFFT. It is the output signal.
- $X[k]$: This represents the frequency domain representation of the signal. It denotes the complex amplitude at frequency index k.
- $\exp(j * 2\pi * k * n / N)$: This term represents a complex sinusoidal function with a frequency index k. It is raised to the power of j (the imaginary unit multiplied by π), and multiplied by 2π and k and n divided by N. This term is used to reconstruct the time domain signal from its frequency components.

Similar to the FFT, the IFFT algorithm employs the "decimation in time" approach to efficiently compute the inverse transformation. It reverses the order of operations compared to the FFT, enabling the reconstruction of the original signal.

By applying the IFFT to the frequency domain representation obtained through the FFT, you can convert back from the frequency domain to the time domain, allowing further analysis or processing in the spatial domain.

Both the FFT and IFFT are powerful tools in signal processing and have extensive applications in various fields, including medical imaging.

ssp

5.3 CODE :

```
i = imread('MRI.jpeg');
subplot(1,3,1)
imshow(i);
title('Original Image');



grayi = rgb2gray(i);
fftimage = fftshift(fft2(grayi));
subplot(1,3,2)
fftshow = mat2gray(log(1+abs(fftimage)));
imshow(fftshow)
title('FFT of Image');


inverse = abs(ifft2(fftimage));
inverse = mat2gray(inverse);
subplot(1,3,3)
imshow(inverse);
title('IFFT of Image');
```

- fftshift Shift zero-frequency component to center of spectrum. For matrices, fftshift(X) swaps the first and thirdquadrants and the second and fourth quadrants.
- fft2 Two-dimensional discrete Fourier Transform.fft2(X) returns the two-dimensional Fourier transform of matrix X. If X is a vector, the result will have the same orientation.
- Two-dimensional inverse discrete Fourier transform.
- ifft2(F) returns the two-dimensional inverse Fourier transform of matrix

OUTPUT:

1.

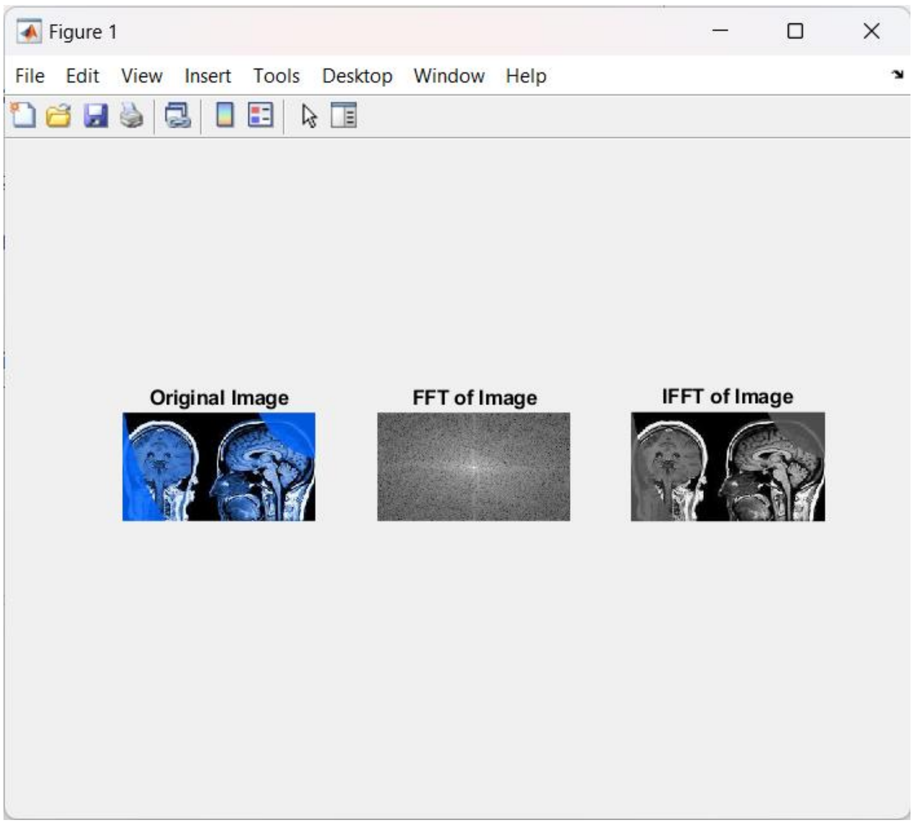


Figure 1
File Edit View Insert Tools Desktop Window Help
Original Image
FFT of Image
IFFT of Image

2.

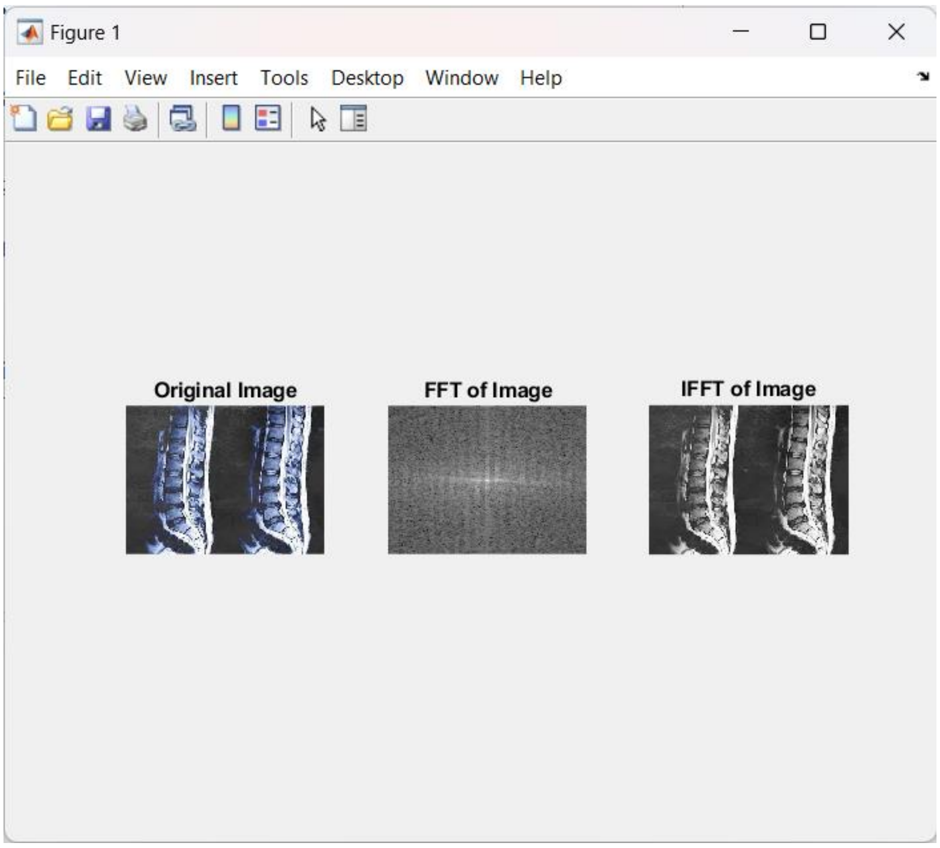


Figure 1
File Edit View Insert Tools Desktop Window Help
Original Image
FFT of Image
IFFT of Image

6 CONCLUSION :

Fourier transform apart from these has a vast number of applications. The Fourier transform finds its applications in fields such as electronics, communication, computer science, etc. Thus, we can conclude that Fourier transform has many real life applications and is much useful in our daily life of computers and electronics.