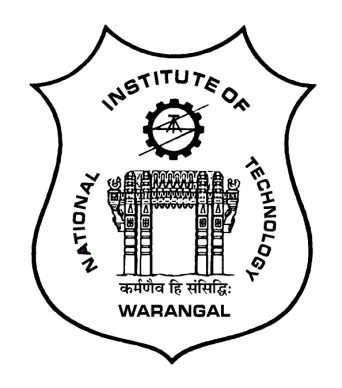
**NATIONAL INSTITUTE OF TECHNOLOGY, WARANGAL**

***Department of Electronics and Communication Engineering***



**Speech Signal Processing Using Butterworth filter on Arduino NANO 33 BLE Sense**

By:

**Anirudh Bandi (21ECB0B02)**

**Aditya Mishra (21ECB0B03)**

**Sandeep (21ECB0B14)**

Guided By:

**Dr. J. Ravi Kumar**

**Professor, ECE**

**TABLE OF CONTENTS:**

1. ABSTRACT
2. INTRODUCTION

* INFINITE IMPULSE RESPONSE(IIR) FILTER
* BUTTERWORTH FILTER
* ARDUINO NANO BLE 33 SENSE

1. FILTER USED

* FILTER SPECIFICATIONS
* MAGNITUDE RESPONSE
* PHASE RESPONSE
* ZERO – POLE PLOT
* DIRECT FORM – ll REALIZATION
* DIFFERENCE EQUATIONS

1. SOFTWARE USED

* Arduino IDE
* MATLAB

1. CODES

* Arduino Codes
* MATLAB Code

1. SIMULATION AND OUTPUTS

**ABSTRACT:**

The project aims to leverage the capabilities of the Arduino Nano BLE 33 Sense microcontroller in conjunction with a IIR Butterworth bandpass filter for advanced audio processing applications. The Nano BLE 33 Sense, equipped with a variety of sensors, including a MEMS microphone, accelerometer, and environmental sensors, provides a powerful platform for collecting and processing audio data in real-time.

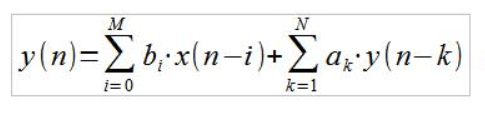
The primary focus of this project is to implement a bandpass filter as a crucial component in the audio processing pipeline. The bandpass filter is designed to selectively allow a specific range of frequencies to pass through, effectively isolating and enhancing targeted audio signals.

The MATLAB Filter Design Toolbox is employed for the design and analysis of the IIR Butterworth bandpass filter. MATLAB provides a comprehensive environment for filter design, allowing for precise tuning and optimization of filter parameters for the target application.

**INTRODUCTION:**

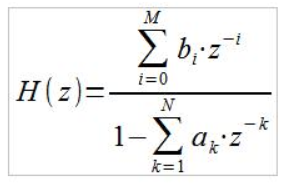
**INFINITE IMPULSE RESPONSE (IIR) FILTERS:**

An IIR filter, or Infinite Impulse Response filter, is a type of digital or analog filter that uses feedback to achieve its filtering characteristics. The "infinite impulse response" terminology refers to the fact that the filter's response to an impulse input does not immediately decay to zero but continues indefinitely. IIR filters are characterized by their recursive nature, where the output of the filter depends on both the current input and the past outputs.



The general difference between IIR filters and FIR (Finite Impulse Response) filters lies in their impulse response. In an IIR filter, the impulse response extends infinitely, while in an FIR filter, it has a finite duration.

The transfer function of a general IIR filter in the discrete-time domain is represented as the ratio of two polynomials in the z-transform domain:

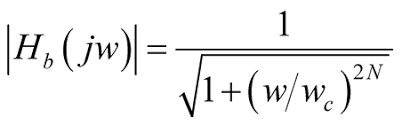


**BUTTERWORTH FILTER:**

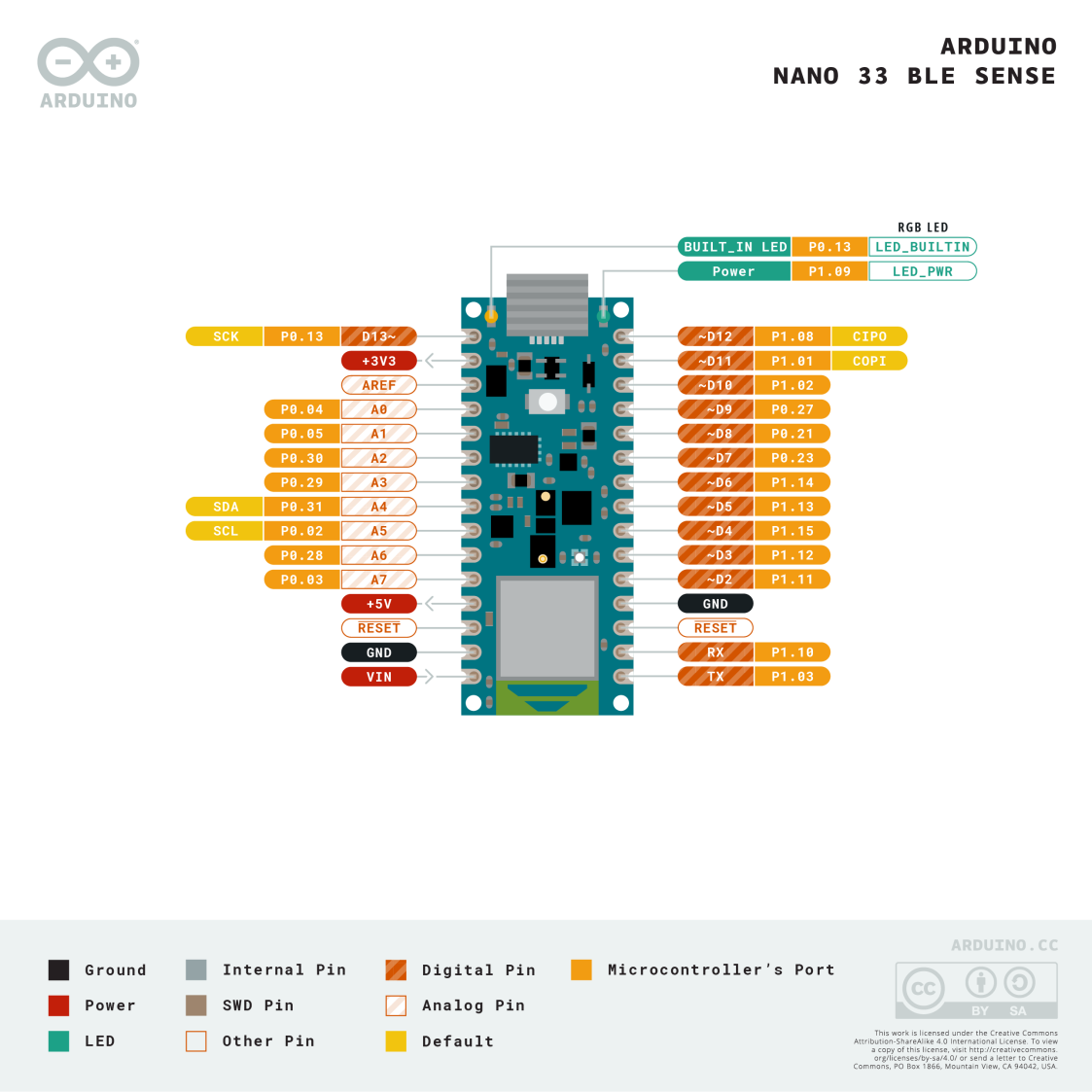
A Butterworth filter is a type of electronic filter designed to have a flat frequency response in the passband. It is a type of infinite impulse response (IIR) filter, which means that it uses feedback in its implementation. The key characteristic of a Butterworth filter is that it provides a maximally flat response in the passband, which means that the amplitude response is as flat as possible within that range.

The Butterworth filter is commonly used in various applications, including signal processing, audio systems, and communication systems. It is characterized by its smooth and monotonic frequency response, but it does not have as sharp a roll-off as some other filter types, such as Chebyshev or elliptic filters.

The filter design is defined by two main parameters: the order of the filter and the cutoff frequency. The order of the filter determines the rate of attenuation in the stopband, and the cutoff frequency is the frequency at which the response starts to roll off.



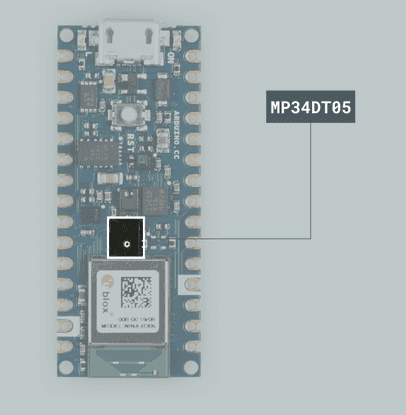
**ARDUINO NANO 33 BLE SENSE:**



Arduino Nano 33 BLE Sense is a miniature sized module containing a NINA B306 module, based on Nordic nRF52480 and containing a Cortex M4F, a crypto chip which can securely store certificates and pre shared keys and a 9 axis IMU. The module can either be mounted as a DIP component (when mounting pin headers), or as a SMT component, directly soldering it via the castellated pads.

Microphone (On Board): #include<PDM.h>

The MP34DT05 Sensor:



Microphones are components that convert physical sound into digital data. Microphones are commonly used in mobile terminals, speech recognition systems or even gaming and virtual reality input devices.

The MP34DT05 sensor is a ultra-compact microphone that use PDM (Pulse-Density Modulation) to represent an analog signal with a binary signal. The sensor's ranges of different values are the following:

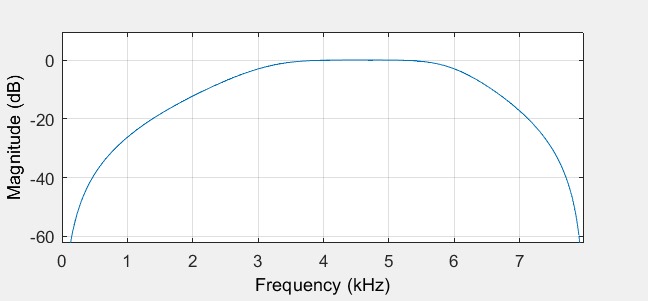
* Signal-to-noise ratio: 64dB
* Sensitivity: -26dBFS ±3dB
* Temperature range: -40 to 85°C

**FILTER USED: IIR BUTTERWORTH BANDPASS FILTER**

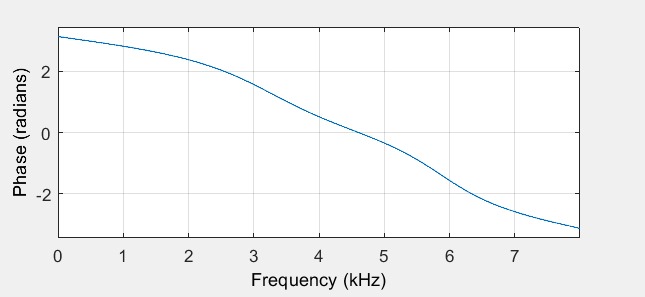
**Filter specifications:**

* Design method : IIR
* Response type : Bandpass
* Realization Structure : Direct form – ll
* Order : 4
* Sections : 2
* Sampling frequency (Fs) : 16 KHz
* Cut-off frequencies : Fc1 = 3 KHz& Fc2 = 6 KHz

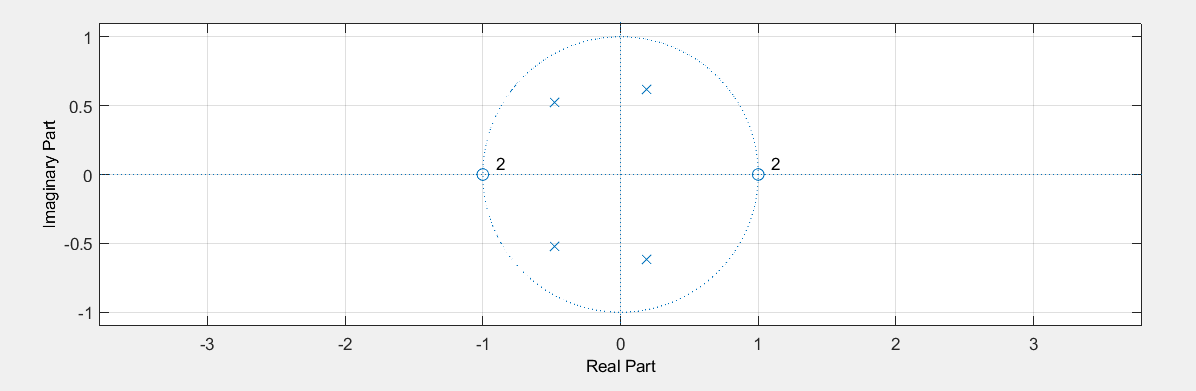
**Magnitude response:**

****

**Phase response:**

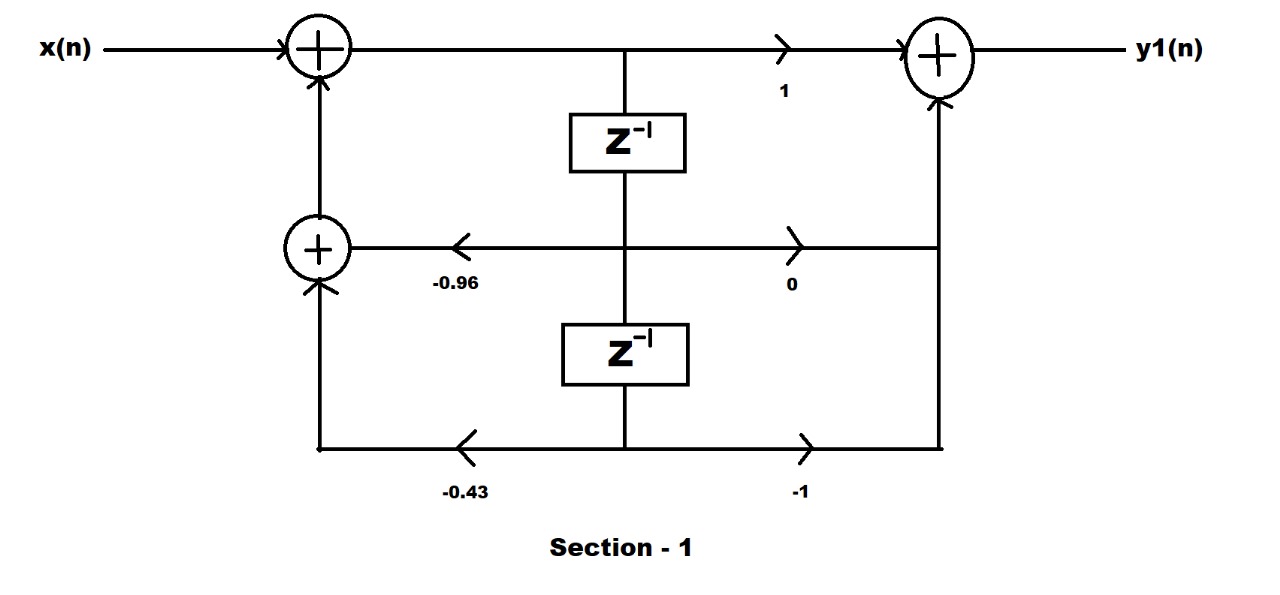


**Pole-Zero plot:**

****

**Direct form – ll structure:**

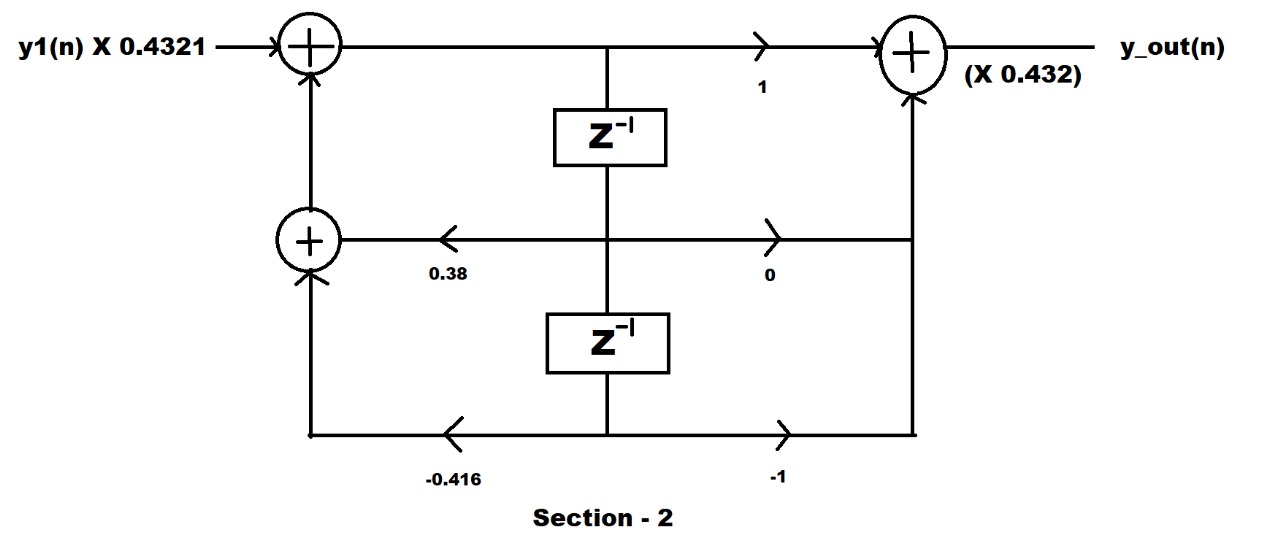
Section -1



Coefficients table

|  |  |  |
| --- | --- | --- |
| Numerator | Denominator | Gain |
| 1 | 1 | 0.432 |
| 0 | 0.96 |  |
| -1 | 0.43 |  |

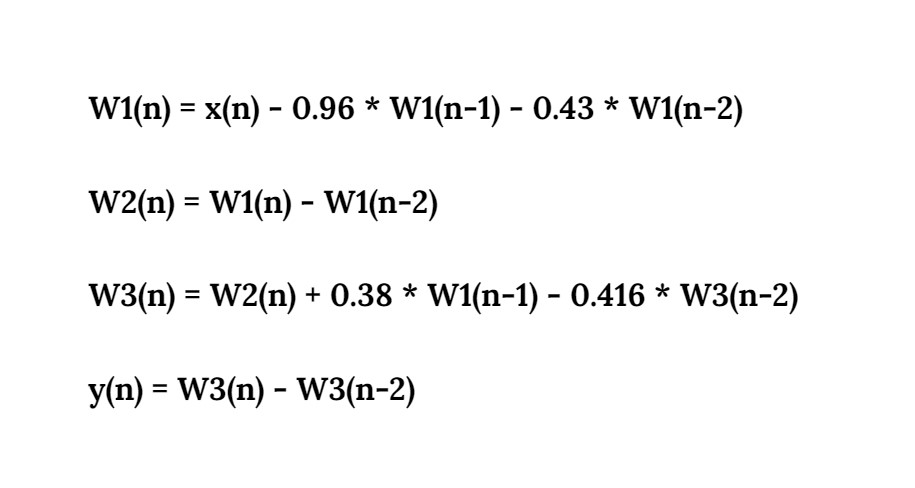
Section – 2



Coefficients table

|  |  |  |
| --- | --- | --- |
| Numerator | Denominator | Gain |
| 1 | 1 | 0.432 |
| 0 | -0.38 |  |
| -1 | 0.416 |  |

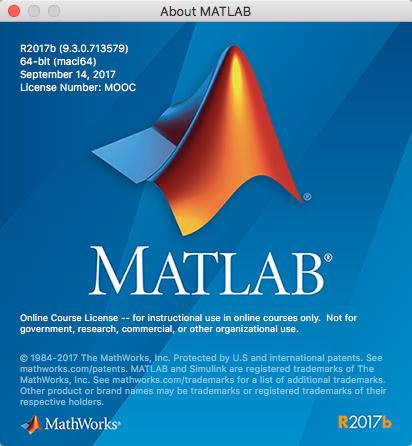
**Difference Equations:**

****

**Software Used – Arduino IDE**

****

**MATLAB**



**Codes:**

**Arduino Code:** Takes the input audio from on board microphone, generates the samples and produce the output samples after filtration.

#include<PDM.h>

#include<arduinoFFT.h>

#defineBUFFER\_SIZE512

#defineSAMPLING\_FREQUENCY16000

#defineSAMPLES512

#defineRECORDING\_DURATION5000

constuint8\_t amplitude = 100;

shortaudioBuffer[RECORDING\_DURATION \* SAMPLING\_FREQUENCY / 10000];

shortfinalaudioBuffer[RECORDING\_DURATION \* SAMPLING\_FREQUENCY / 10000];

shortsampleBuffer[SAMPLES];

int audioIndex = 0;

int finalaudioIndex = 0;

doublex[BUFFER\_SIZE];

doubley1[BUFFER\_SIZE];

doubley1\_1[BUFFER\_SIZE];

doubley\_out[BUFFER\_SIZE];

doubley\_out\_1[BUFFER\_SIZE];

doubley\_um[BUFFER\_SIZE];

doublevR\_um[SAMPLES];

doublevI\_um[SAMPLES];

volatileint samplesRead;

unsignedlong recordingStartTime;

arduinoFFT FFT = arduinoFFT();

voidonPDMdata(void);

voidsetup(){

  Serial.begin(115200);

  while(!Serial)

    ;

  PDM.onReceive(onPDMdata);

  PDM.setBufferSize(SAMPLES);

  if(!PDM.begin(1, 16000)){

    Serial.println("Failed to start PDM!");

    while(1)

      ;

  }

  recordingStartTime = 0;

}

voidloop(){

  if(recordingStartTime == 0){

    recordingStartTime = millis();

  }

  if(millis() - recordingStartTime < RECORDING\_DURATION){

    if(samplesRead){

      for(int i = 0; i < SAMPLES; i++){

        audioBuffer[audioIndex] = sampleBuffer[i];

        audioIndex++;

        if(audioIndex >= sizeof(audioBuffer) / sizeof(audioBuffer[0])){

          recordingStartTime = RECORDING\_DURATION + 1;

          break;

        }

      }

    }

  }

   for(int i = 0; i < BUFFER\_SIZE; i++){

      x[i] = sampleBuffer[i];

    }

//Section 1

    y1[0] = x[0];

    y1[1] = x[1] - 0.9587\* y1[0];

    for(int i = 2; i < BUFFER\_SIZE; i++){

      y1[i] = x[i] - x[i - 2] - 0.9587 \* y1[i - 1] - 0.5039 \* y1[i - 2];

    }

    for(int i = 0; i < BUFFER\_SIZE; i++){

      y1[i] = 0.4321 \* y1[i];

    }

// Section 2

    y\_out[0] = y1[0];

    y\_out[1] = y1[1] + 0.3808 \* y\_out[0];

    for(int i = 2; i < BUFFER\_SIZE; i++){

      y\_out[i] = y1[i] - y1[i - 2]+ 0.3808 \* y\_out[i - 1] - 0.4161 \* y\_out[i - 2];

    }

    for(int i = 0; i < BUFFER\_SIZE; i++){

      y\_out[i] = 0.4321 \* y\_out[i];

    }

    for(int i = 0; i < SAMPLES; i++){

      vR\_um[i] = y\_out[i];

      vI\_um[i] = 0;

      printf("%d ",vR\_um[i]);

    }

     for(int i = 0; i < SAMPLES; i++){

        finalaudioBuffer[finalaudioIndex] = vR\_um[i];

        Serial.println(vR\_um[i]);

        finalaudioIndex++;

     }

}

voidonPDMdata(){

  int bytesAvailable = PDM.available();

  PDM.read(sampleBuffer, bytesAvailable);

  samplesRead = bytesAvailable / 2;

}

**Matlab Code:** Takes the output samples from the text file containing filtered signal, generates FFT and plot the magnitude spectrum of final filtered signal.

% Specify the file path

file\_path = 'filteredsignal.txt';

% Read the CSV file into a table

table\_data = readtable(file\_path);

% Convert the table to an array

x = table2array(table\_data);

while true

% Plot Configuration

fm = 10000; % Maximum display frequency.

plot\_title = 'Output Signal Spectrum';

x\_label = '$f$(Hz)';

y\_label = '$|Y(e^{j2\pi f})|\_{dB}$';

% Realization of FFT and normalization.

fft\_signal = fft(x);

fft\_length = length(fft\_signal);

fft\_signal = abs(fft\_signal)/fft\_length;

f0 = Fs/fft\_length;

f0max = (fft\_length - 1)\*f0;

f = 0:f0:f0max;

% Frequency signal correction.

metade = fft\_length/2;

fp = f(1:metade);

fft\_signal\_positive = fft\_signal(1:metade);

fn = f(metade+1:fft\_length) - f0max - f0;

fft\_signal\_negative = fft\_signal(metade+1:fft\_length);

ft = [fn, fp];

fft\_signal\_total = [fft\_signal\_negative', fft\_signal\_positive'];

fft\_signal\_total = fft\_signal\_total';

y\_db = 20\*log(fft\_signal\_total);

% Display of the spectrum of the input signal.

plot(ft, y\_db); grid on;

xlim([0 fm])

ylim([-400 0])

title(plot\_title);

hTitle = get(gca, 'title');

set(hTitle, 'FontSize', 48, 'FontWeight', 'bold')

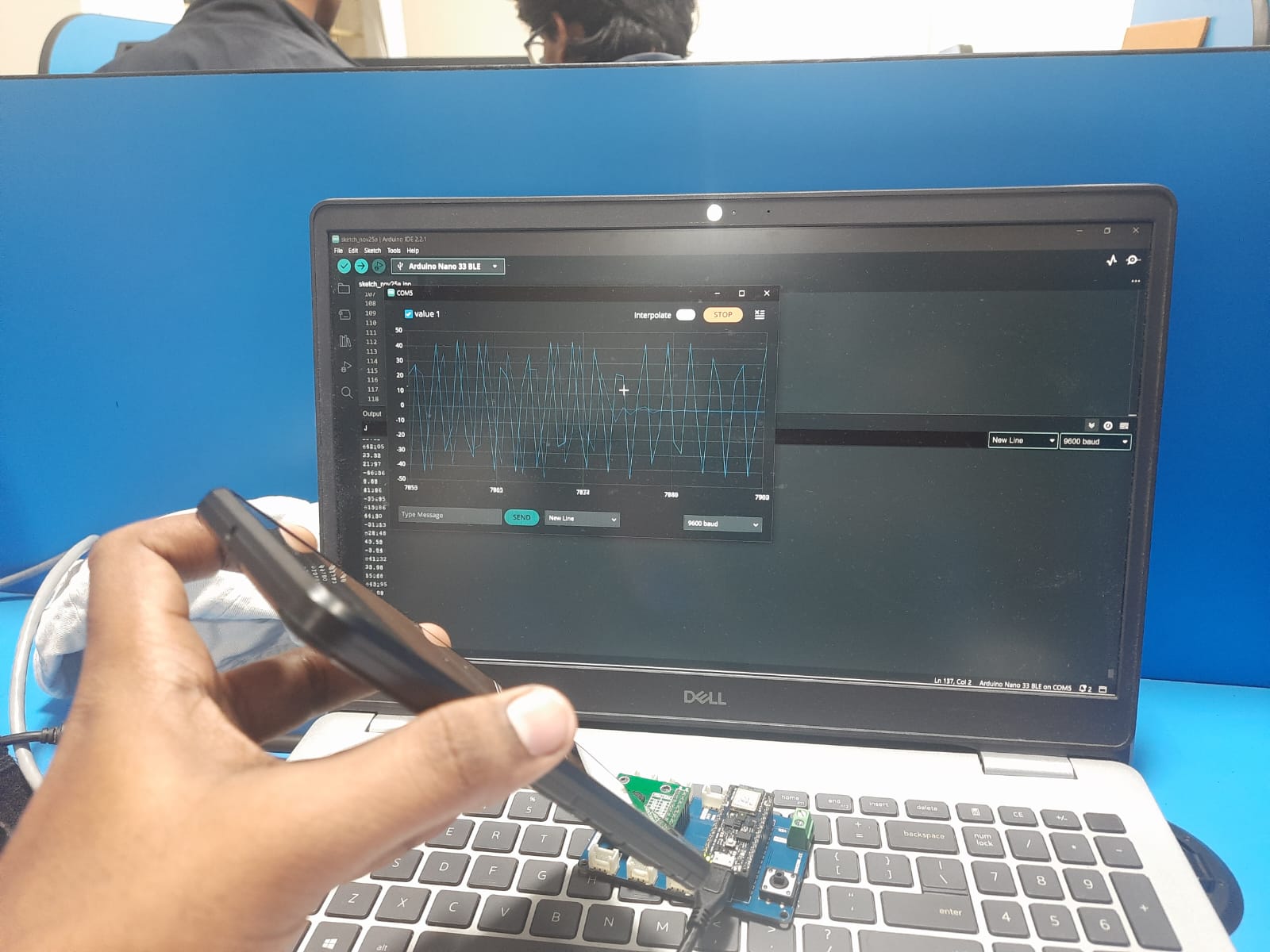
xlabel(x\_label, 'interpreter', 'latex','FontSize', 24);

ylabel(y\_label, 'interpreter', 'latex','FontSize', 24);

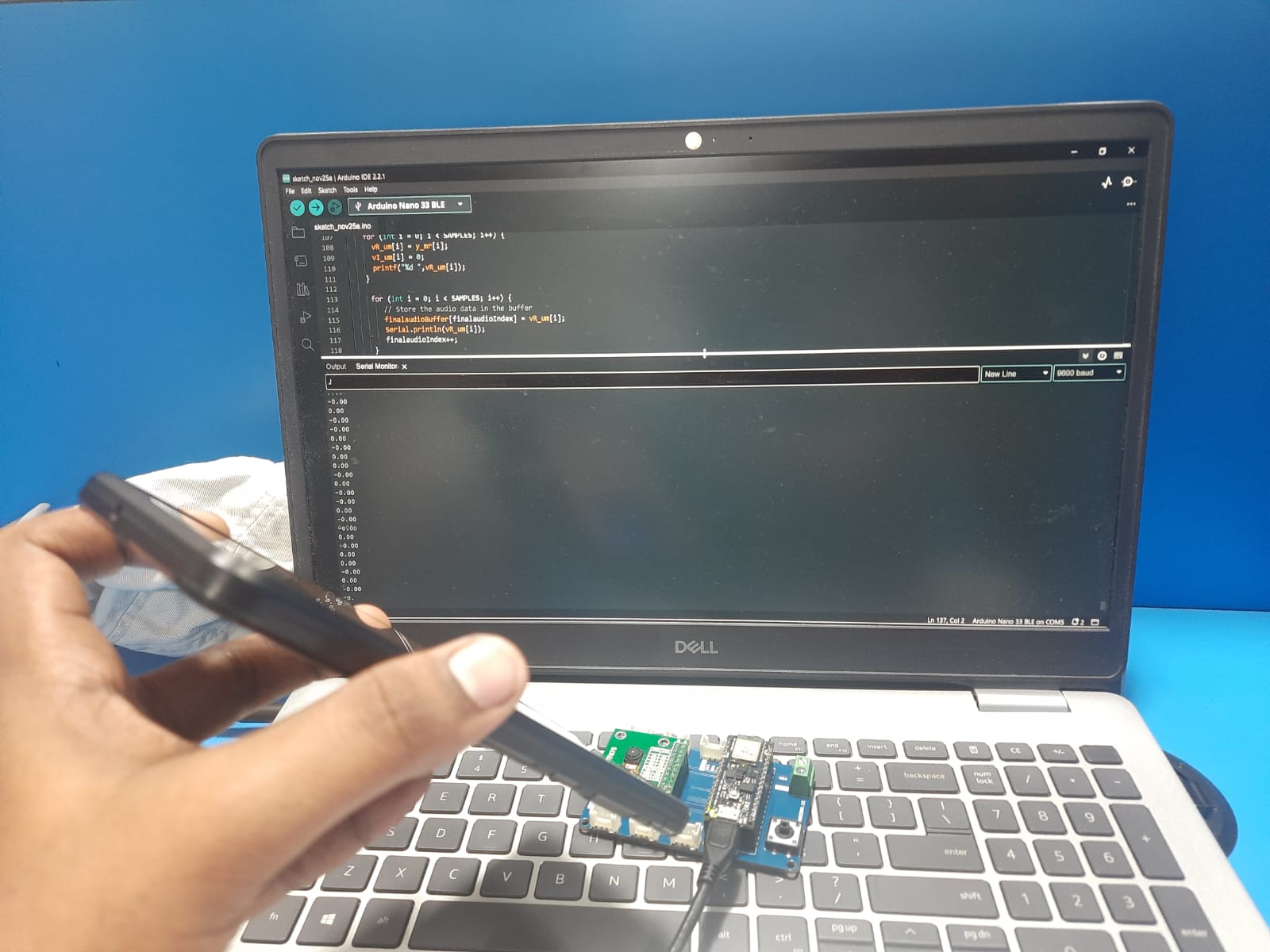
drawnow

end

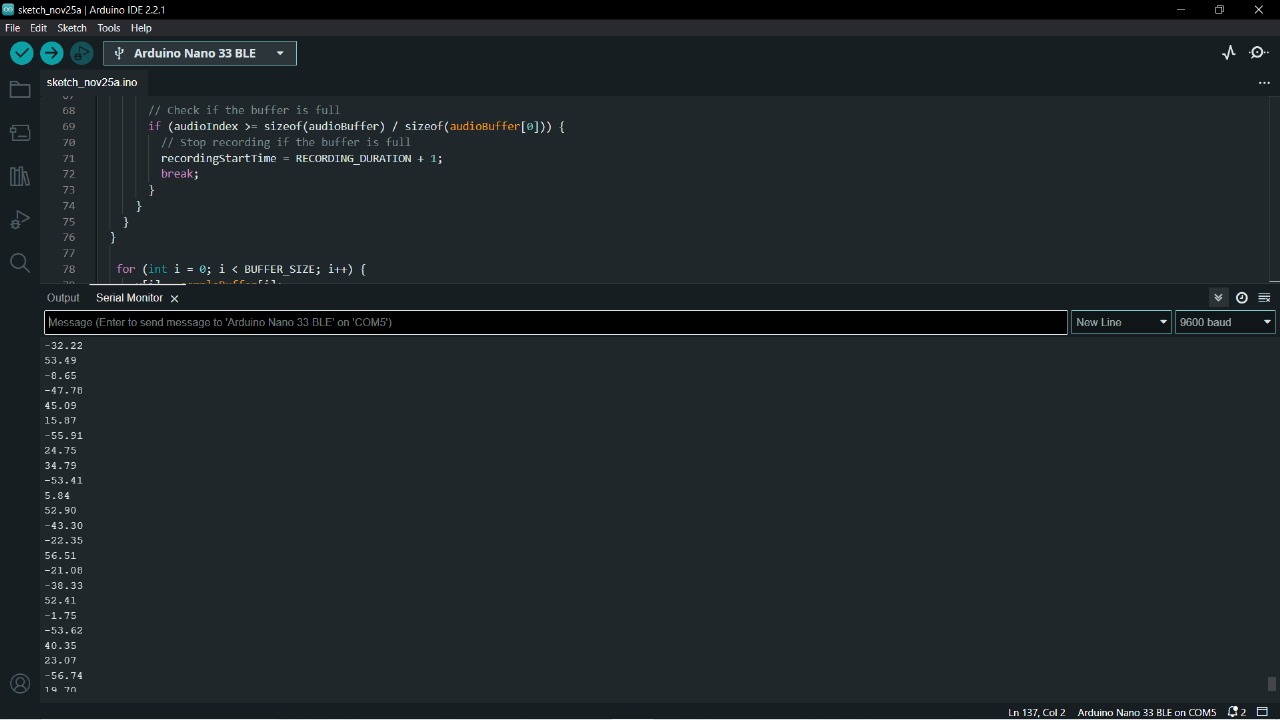
**Outputs: Serial Plotter Output signal Wave (Time Domain)**

****

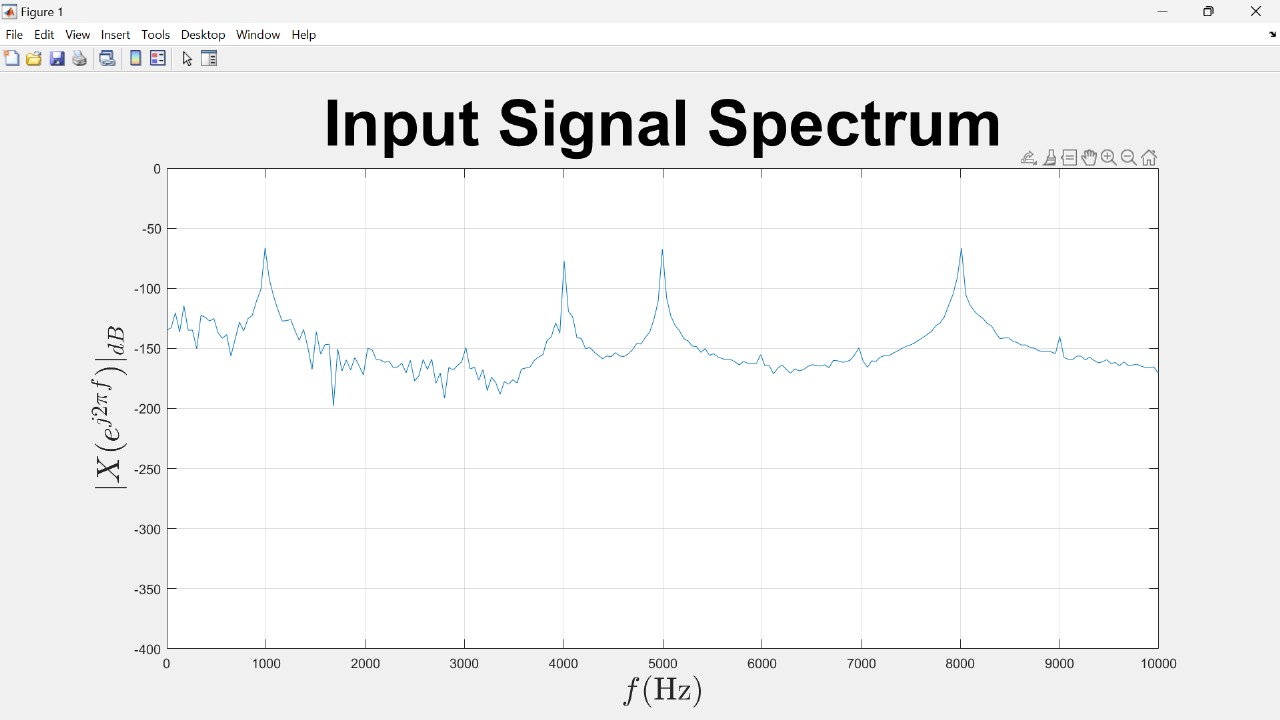
**Samples generation after filtration**

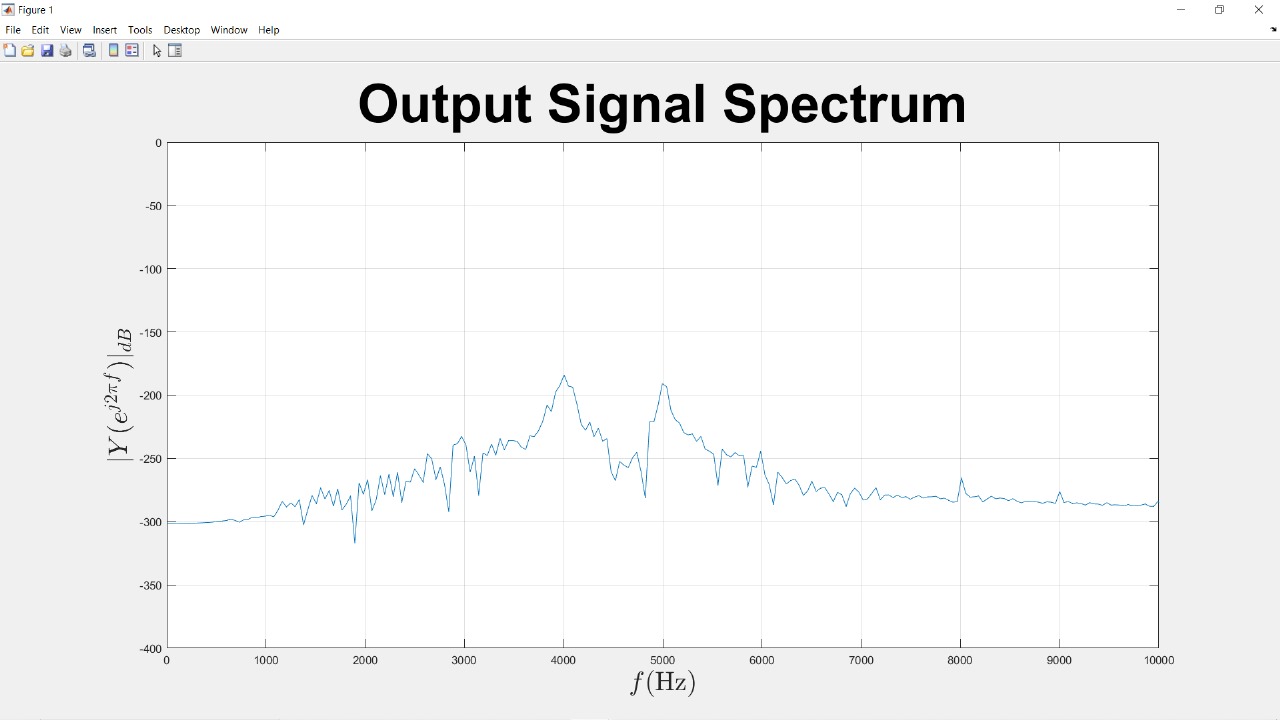
****

**Filtered samples generation on Arduino IDE**

****

**Comparing the signals:**





**Conclusion:**

Conclusion: In conclusion, the successful implementation of a bandpass filter and subsequent Fast Fourier Transform (FFT) analysis on the Arduino Nano BLE 33 Sense has proven to be an effective approach for isolating and visualizing specific frequency components in a given signal. The utilization of the on-board processing capabilities of the Arduino Nano BLE 33 Sense, coupled with its ability to interface with sensors, opens up opportunities for diverse applications in signal processing, data analysis, and sensor-based projects. This project showcases the adaptability and efficiency of the Arduino Nano BLE 33 Sense in handling real-time signal manipulation and analysis, underscoring its relevance in projects that require a compact, low power and versatile microcontroller platform.