Adaptive noise cancellation:

1. PREDICTIVE ANALYSIS

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define FRAME_SIZE 1024
#define PREDICTION_ORDER 3  // Number of past samples to use for prediction

// WAV header structure
typedef struct {
    char chunkID[4];
    int chunkSize;
    char format[4];
    //Stores "WAVE", indicating this is a WAV file.
    char subchunk1ID[4];
    //Stores "fmt " (ASCII) to mark the format chunk.
    int subchunk1Size;
    //Size of this subchunk (always 16 for PCM)
    short audioFormat;
    //Audio encoding type (1 = PCM, uncompressed).
    short numChannels;
    //1 = Mono, 2 = Stereo (number of audio channels).
    int sampleRate;
    int byteRate;
    short blockAlign;
    short bitsPerSample;
    char subchunk2ID[4];
    int subchunk2Size;
} WAVHeader;

// Predict noise using an Auto-Regressive (AR) Model
short predict_noise(short *noise_history) {
    // Simple AR model: Weighted sum of past samples
    float weights[PREDICTION_ORDER] = {0.5, -0.3, 0.2}; // Example coefficients
    float predicted_value = 0.0;

    for (int i = 0; i < PREDICTION_ORDER; i++) {
        predicted_value += weights[i] * noise_history[i];
```

```
    }

    return (short)predicted_value;
}

// Adaptive Noise Cancellation using Predictive Filtering
void predictive_anc(short *input, short *output, int numSamples) {
    short noise_history[PREDICTION_ORDER] = {0};

    for (int i = 0; i < numSamples; i++) {
        // Predict noise from previous samples
        short predicted_noise = predict_noise(noise_history);

        // Remove predicted noise from the current input sample
        output[i] = input[i] - predicted_noise;

        // Update noise history
        for (int j = PREDICTION_ORDER - 1; j > 0; j--) {
            noise_history[j] = noise_history[j - 1];
        }
        noise_history[0] = input[i]; // Store current input as next history sample
    }
    //Predicts noise using past samples.
    //Subtracts the predicted noise from the input sample.
    //Updates the noise history for the next iteration.

}

// Read WAV file
short *read_wav(const char *filename, WAVHeader *header, int *numSamples) {
    FILE *file = fopen(filename, "rb");
    if (!file) {
        printf("Error opening input file!\n");
        return NULL;
    }

    fread(header, sizeof(WAVHeader), 1, file);
    *numSamples = header->subchunk2Size / sizeof(short);
    //16 bytes pcm

    short *data = (short *)malloc(*numSamples * sizeof(short));
    fread(data, sizeof(short), *numSamples, file);
    fclose(file);
```

```
    return data;
}

// Write WAV file
void write_wav(const char *filename, WAVHeader *header, short *data, int numSample
    FILE *file = fopen(filename, "wb");
    if (!file) {
        printf("Error opening output file!\n");
        return;
    }

    fwrite(header, sizeof(WAVHeader), 1, file);
    fwrite(data, sizeof(short), numSamples, file);
    fclose(file);
}

int main(int argc, char *argv[]) {
    if (argc != 3) {
        printf("Usage: %s <input.wav> <output.wav>\n", argv[0]);
        return 1;
    }

    WAVHeader header;
    int numSamples;

    // Read input WAV file
    short *input = read_wav(argv[1], &header, &numSamples);
    if (!input) return 1;

    // Allocate memory for output
    short *output = (short *)malloc(numSamples * sizeof(short));

    // Apply Predictive ANC
    predictive_anc(input, output, numSamples);

    // Write output WAV file
    write_wav(argv[2], &header, output, numSamples);

    // Clean up
    free(input);
    free(output);

    printf("Noise cancellation completed. Output saved to %s\n", argv[2]);
```

```
    return 0;
}
```

2. ADAPTIVE LEAST MEAN SQUARE

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define FRAME_SIZE 1024
#define MU 0.0001  // Learning rate

// WAV header structure
typedef struct {
    char chunkID[4];
    int chunkSize;
    char format[4];
    char subchunk1ID[4];
    int subchunk1Size;
    short audioFormat;
    short numChannels;
    int sampleRate;
    int byteRate;
    short blockAlign;
    short bitsPerSample;
    char subchunk2ID[4];
    int subchunk2Size;
} WAVHeader;

// Adaptive LMS Filter
void lms_filter(short *desired, short *reference, short *output, int numSamples) {
    float w = 0.0;  // Filter weight
    float error, y;

    for (int i = 0; i < numSamples; i++) {
        y = w * reference[i];   // Filtered output
        error = desired[i] - y; // Error signal
        MU = 0.0001 + 0.01 * fabs(error);
        // Variable Step-Size LMS (VSSLMS):
        w += MU * error * reference[i]; // Weight update
        output[i] = (short) error;
    }
}
```

```
// Read WAV file
short *read_wav(const char *filename, WAVHeader *header, int *numSamples) {
    FILE *file = fopen(filename, "rb");
    if (!file) {
        printf("Error opening input file!\n");
        return NULL;
    }

    fread(header, sizeof(WAVHeader), 1, file);
    *numSamples = header->subchunk2Size / sizeof(short);

    short *data = (short *)malloc(*numSamples * sizeof(short));
    fread(data, sizeof(short), *numSamples, file);
    fclose(file);
    return data;
}

// Write WAV file
void write_wav(const char *filename, WAVHeader *header, short *data, int numSample
    FILE *file = fopen(filename, "wb");
    if (!file) {
        printf("Error opening output file!\n");
        return;
    }

    fwrite(header, sizeof(WAVHeader), 1, file);
    fwrite(data, sizeof(short), numSamples, file);
    fclose(file);
}

int main(int argc, char *argv[]) {
    if (argc != 4) {
        printf("Usage: %s <desired.wav> <noise.wav> <output.wav>\n", argv[0]);
        return 1;
    }

    WAVHeader header;
    int numSamplesDesired, numSamplesNoise;

    // Read desired signal (speech + noise)
    short *desired = read_wav(argv[1], &header, &numSamplesDesired);
    if (!desired) return 1;
```

```
    // Read reference noise signal
    short *reference = read_wav(argv[2], &header, &numSamplesNoise);
    if (!reference || numSamplesDesired != numSamplesNoise) {
        printf("Error: Mismatched file sizes!\n");
        free(desired);
        return 1;
    }

    // Allocate memory for output
    short *output = (short *)malloc(numSamplesDesired * sizeof(short));

    // Apply LMS adaptive filter
    lms_filter(desired, reference, output, numSamplesDesired);

    // Write output WAV file
    write_wav(argv[3], &header, output, numSamplesDesired);

    // Clean up
    free(desired);
    free(reference);
    free(output);

    printf("Noise cancellation completed. Output saved to %s\n", argv[3]);
    return 0;
}
```

3. Recursive least square:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
// WAV header structure
typedef struct {
    char chunkID[4];
    int chunkSize;
    char format[4];
    char subchunk1ID[4];
    int subchunk1Size;
    short audioFormat;
    short numChannels;
    int sampleRate;
    int byteRate;
```

```
    short blockAlign;
    short bitsPerSample;
    char subchunk2ID[4];
    int subchunk2Size;
} WAVHeader;
// Read WAV file
short *read_wav(const char *filename, WAVHeader *header, int *numSamples) {
    FILE *file = fopen(filename, "rb");
    if (!file) {
        printf("Error opening input file!\n");
        return NULL;
    }

    fread(header, sizeof(WAVHeader), 1, file);
    *numSamples = header->subchunk2Size / sizeof(short);

    short *data = (short *)malloc(*numSamples * sizeof(short));
    fread(data, sizeof(short), *numSamples, file);
    fclose(file);
    return data;
}

// Write WAV file
void write_wav(const char *filename, WAVHeader *header, short *data, int numSample
    FILE *file = fopen(filename, "wb");
    if (!file) {
        printf("Error opening output file!\n");
        return;
    }

    fwrite(header, sizeof(WAVHeader), 1, file);
    fwrite(data, sizeof(short), numSamples, file);
    fclose(file);
}


int main(int argc, char *argv[]) {
    if (argc != 4) {
        printf("Usage: %s <desired_signal.wav> <reference_signal.wav> <output.wav>\n'
        return 1;
    }

    WAVHeader header;
```

```
int numSamplesDesired, numSamplesNoise;

// Read desired signal (clean speech)
short *desired = read_wav(argv[1], &header, &numSamplesDesired);
if (!desired) {
    fprintf(stderr, "Error: Failed to read desired signal from %s\n", argv[1]);
    return 1;
}

// Read reference noise signal
short *reference = read_wav(argv[2], &header, &numSamplesNoise);
if (!reference) {
    fprintf(stderr, "Error: Failed to read reference signal from %s\n", argv[2]);
    free(desired);  // Free previously allocated memory
    return 1;
}

// Ensure both signals have the same length
if (numSamplesDesired != numSamplesNoise) {
    fprintf(stderr, "Error: Mismatched signal lengths!\n");
    free(desired);
    free(reference);
    return 1;
}

// Allocate memory for output signal
short *output = (short *)malloc(numSamplesDesired * sizeof(short));
if (!output) {
    fprintf(stderr, "Error: Memory allocation failed for output signal\n");
    free(desired);
    free(reference);
    return 1;
}

// Adaptive filtering using RLS algorithm
double lambda = 0.99; // Forgetting factor
double delta = 0.01;  // Initialization parameter

int filterOrder = 32; // Order of the adaptive filter
double *weights = (double *)calloc(filterOrder, sizeof(double));
double *buffer = (double *)calloc(filterOrder, sizeof(double));
double *P = (double *)malloc(filterOrder * filterOrder * sizeof(double));
```

```
if (!weights || !buffer || !P) {
    fprintf(stderr, "Error: Memory allocation failed for RLS parameters\n");
    free(desired);
    free(reference);
    free(output);
    free(weights);
    free(buffer);
    free(P);
    return 1;
}

// Initialize P matrix as identity
for (int i = 0; i < filterOrder; i++) {
    for (int j = 0; j < filterOrder; j++) {
        P[i * filterOrder + j] = (i == j) ? (1.0 / delta) : 0.0;
    }
}

// RLS Filtering Process
for (int n = 0; n < numSamplesDesired; n++) {
    // Shift buffer
    for (int k = filterOrder - 1; k > 0; k--) {
        buffer[k] = buffer[k - 1];
    }
    buffer[0] = reference[n];

    // Compute output
    double y = 0.0;
    for (int k = 0; k < filterOrder; k++) {
        y += weights[k] * buffer[k];
    }

    // Compute error signal
    double error = desired[n] - y;
    output[n] = (short)round(error);

    // Compute gain vector K
    double *K = (double *)malloc(filterOrder * sizeof(double));
    double den = lambda;
    for (int k = 0; k < filterOrder; k++) {
        den += buffer[k] * P[k * filterOrder + k] * buffer[k];
    }
    for (int k = 0; k < filterOrder; k++) {
```

```
            K[k] = 0;
            for (int j = 0; j < filterOrder; j++) {
                K[k] += P[k * filterOrder + j] * buffer[j];
            }
            K[k] /= den;
        }

        // Update weight vector
        for (int k = 0; k < filterOrder; k++) {
            weights[k] += K[k] * error;
        }

        // Update inverse correlation matrix P
        double *P_temp = (double *)malloc(filterOrder * filterOrder * sizeof(double));
        for (int i = 0; i < filterOrder; i++) {
            for (int j = 0; j < filterOrder; j++) {
                P_temp[i * filterOrder + j] = P[i * filterOrder + j] - K[i] * buffer[j] * P[j * filterOr
            }
        }
        for (int i = 0; i < filterOrder * filterOrder; i++) {
            P[i] = (P_temp[i] + P_temp[i]) / (2.0 * lambda); // Stabilization
        }

        free(K);
        free(P_temp);
    }

    // Write output to a WAV file
    write_wav(argv[3], &header, output, numSamplesDesired);

    // Free allocated memory
    free(desired);
    free(reference);
    free(output);
    free(weights);
    free(buffer);
    free(P);

    printf("RLS Noise Cancellation completed. Output saved to %s\n", argv[3]);
    return 0;
}
```

4. Conversion of audio to text:

```
#include <stdio.h>
#include <stdlib.h>
#include <sndfile.h>

#define BUFFER_SIZE 1024  // Number of samples processed per iteration

int main(int argc, char *argv[]) {
    if (argc < 2) {
        printf("Usage: %s <input_audio.wav>\n", argv[0]);
        return 1;
    }

    SNDFILE *infile;
    SF_INFO sfinfo;

    // Open the audio file
    infile = sf_open(argv[1], SFM_READ, &sfinfo);
    if (!infile) {
        printf("Error: Unable to open file %s\n", argv[1]);
        return 1;
    }

    printf("Audio file details:\n");
    printf("Sample Rate: %d Hz\n", sfinfo.samplerate);
    printf("Channels: %d\n", sfinfo.channels);
    printf("Frames: %lld\n", sfinfo.frames);

    // Open the output file
    FILE *outfile = fopen("numaudio.txt", "w");
    if (!outfile) {
        perror("Error opening numaudio.txt");
        sf_close(infile);
        return 1;
    }

    // Write metadata to the file
    fprintf(outfile, "Sample Rate: %d Hz\n", sfinfo.samplerate);
    fprintf(outfile, "Channels: %d\n", sfinfo.channels);
    fprintf(outfile, "Frames: %lld\n\n", sfinfo.frames);

    // Buffer to store samples
    float buffer[BUFFER_SIZE];
```

```
    // Read samples and write to file
    sf_count_t readcount;
    while ((readcount = sf_readf_float(infile, buffer, BUFFER_SIZE)) > 0) {
        for (sf_count_t i = 0; i < readcount * sfinfo.channels; i++) {
            fprintf(outfile, "%f\n", buffer[i]);  // Write each sample
        }
    }

    printf("Conversion complete. Data saved to numaudio.txt\n");

    // Cleanup
    fclose(outfile);
    sf_close(infile);
    return 0;
}
```

- **RIFF organizes data into "chunks,"** where each chunk has:
  - A **4-character identifier** (e.g., `"RIFF"`, `"fmt "`, `"data"`)
  - A **size field** specifying the number of bytes in the chunk.
  - The **actual data**.
- **A RIFF file always starts with the "RIFF" chunk,** which contains:
  - The total file size **minus 8 bytes** (since the `chunkID` and `chunkSize` fields are not included).
  - The **format type** (e.g., `"WAVE"` for audio files).

- `int byteRate;`
  - **Speed of data flow** (calculated as):

$$\text{byteRate} = \text{sampleRate} \times \text{numChannels} \times \frac{\text{bitsPerSample}}{8}$$

  - Example: `44100 × 1 × 16/8 = 88200 bytes/sec` for 16-bit mono.
- `short blockAlign;`
  - **Size of one sample frame** (calculated as):

$$\text{blockAlign} = \text{numChannels} \times \frac{\text{bitsPerSample}}{8}$$

  - Example: `2 bytes` per mono sample, `4 bytes` per stereo sample.
- `short bitsPerSample;`
  - **Bit depth** (e.g., `16-bit`, `24-bit`, etc.).

3 Data Chunk

- `char subchunk2ID[4];`
  - Stores **"data"**, marking where actual sound data begins.
- `int subchunk2Size;`
  - **Size of the audio data in bytes.**
  - Equal to:

$$\text{sampleRate} \times \text{numChannels} \times \frac{\text{bitsPerSample}}{8} \times \text{Duration}$$

working principle:

- The ALMS filter estimates the noise using a weight **w(n)**:

$$y(n) = w(n) \cdot x(n)$$

- This is the estimated noise in the speech signal.

3. **Compute Error Signal:**

- The **error** (cleaned output signal) is calculated by subtracting the estimated noise from the desired signal:

$$e(n) = d(n) - y(n)$$

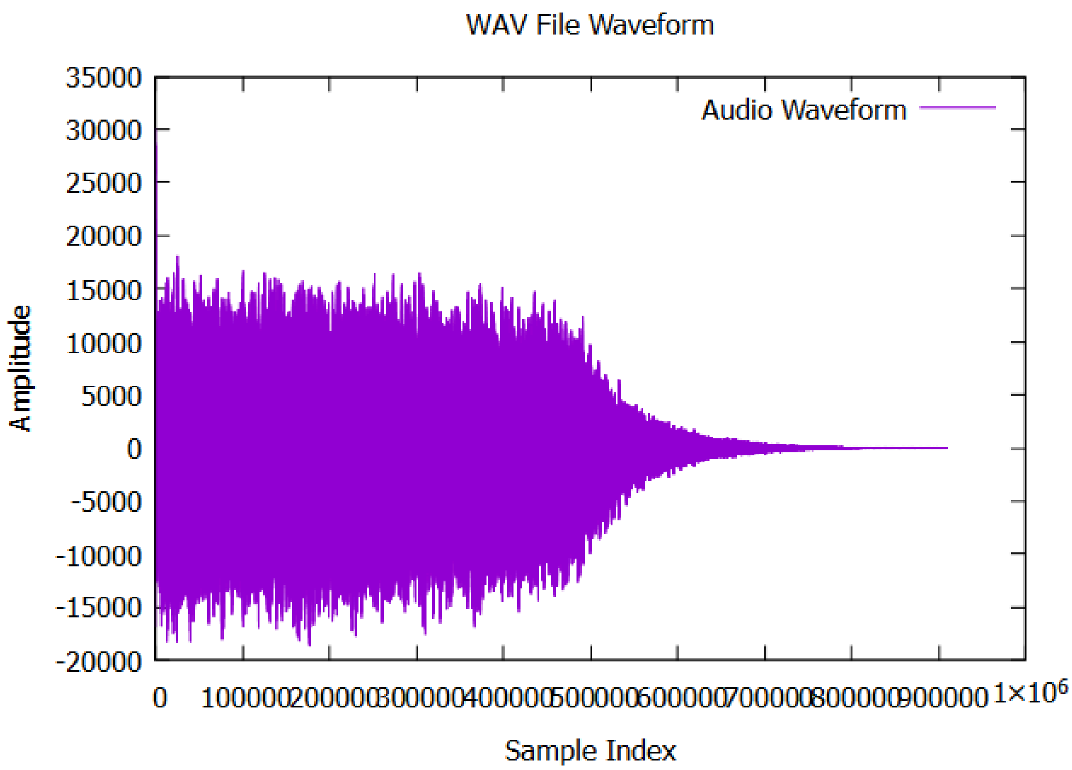
- The error represents the residual signal (ideally, just speech without noise).

4. **Update the Weight:**

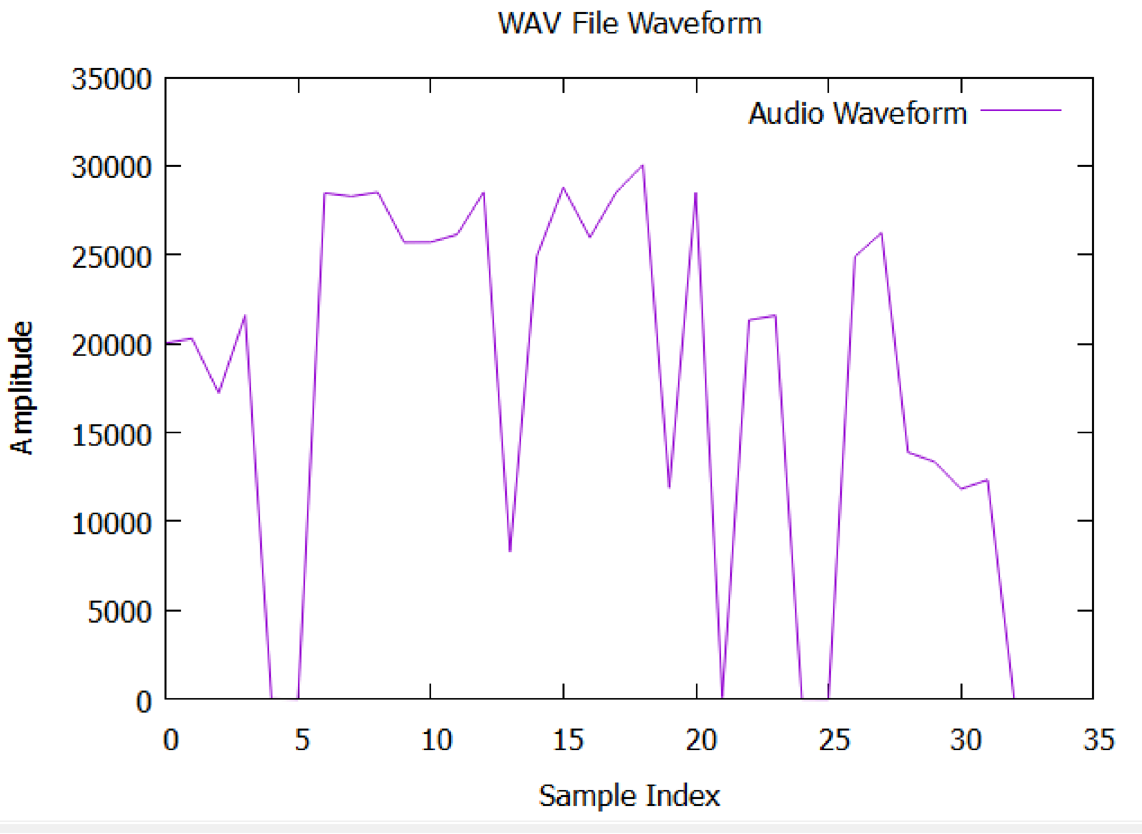
- The filter adjusts the weight **w(n)** using the LMS update rule:

$$w(n+1) = w(n) + \mu \cdot e(n) \cdot x(n)$$

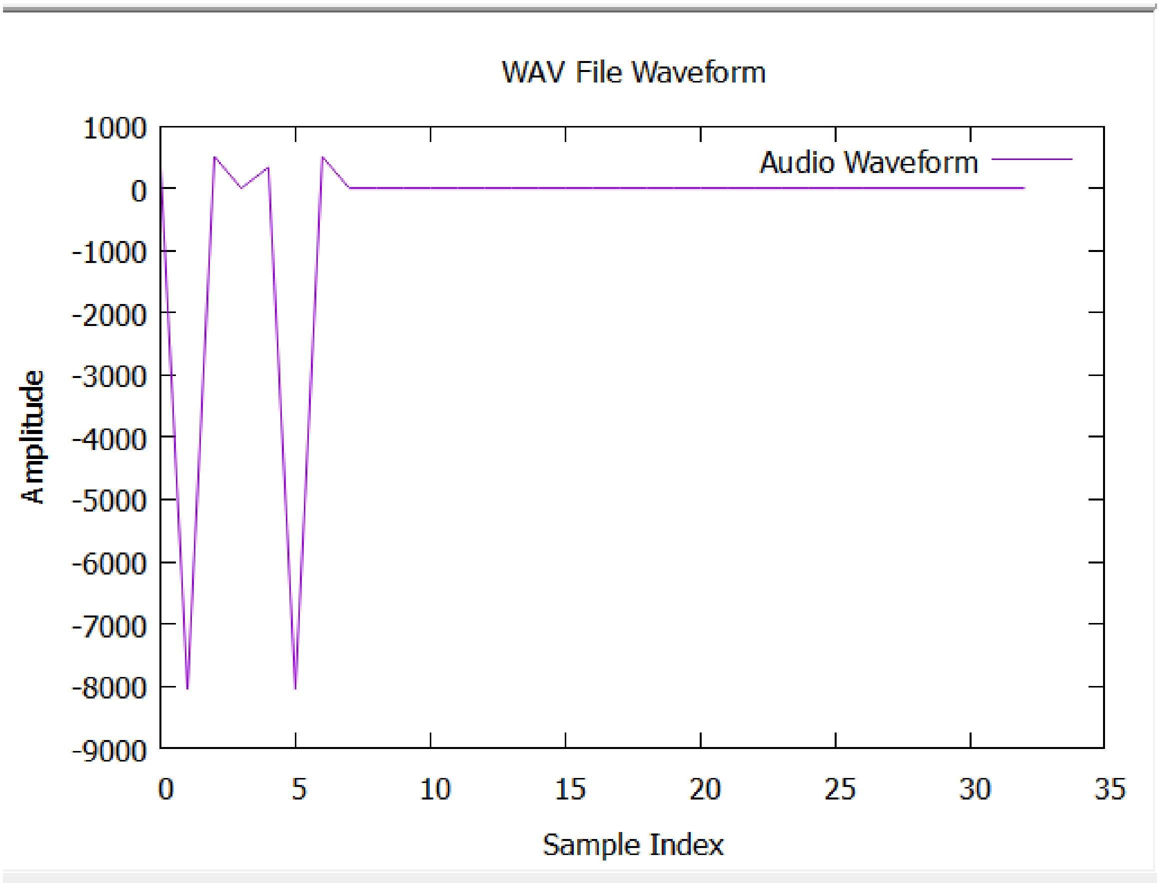
Using of predictive filtering to filter out noises from noisy sources.



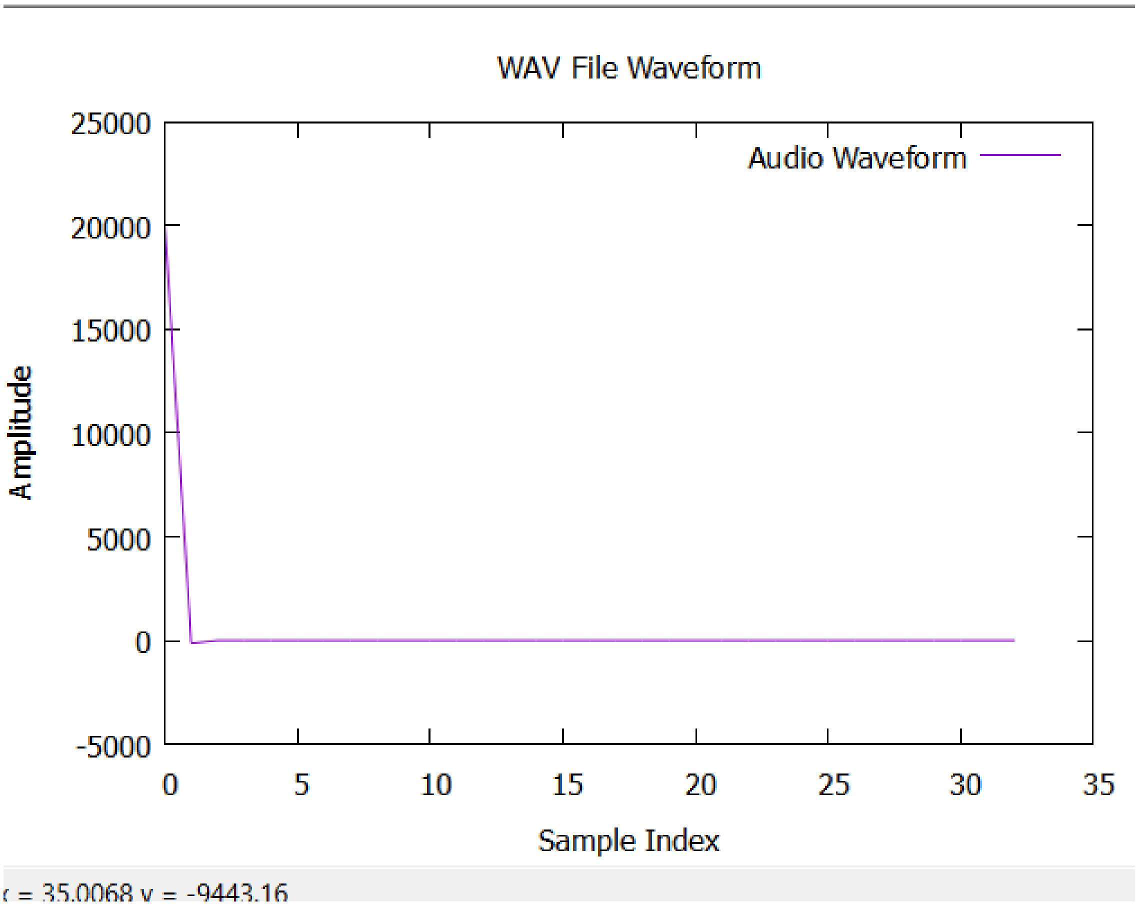
Original noisy input signal from the source



Sampled sound input - after changing the frequency values using c programming

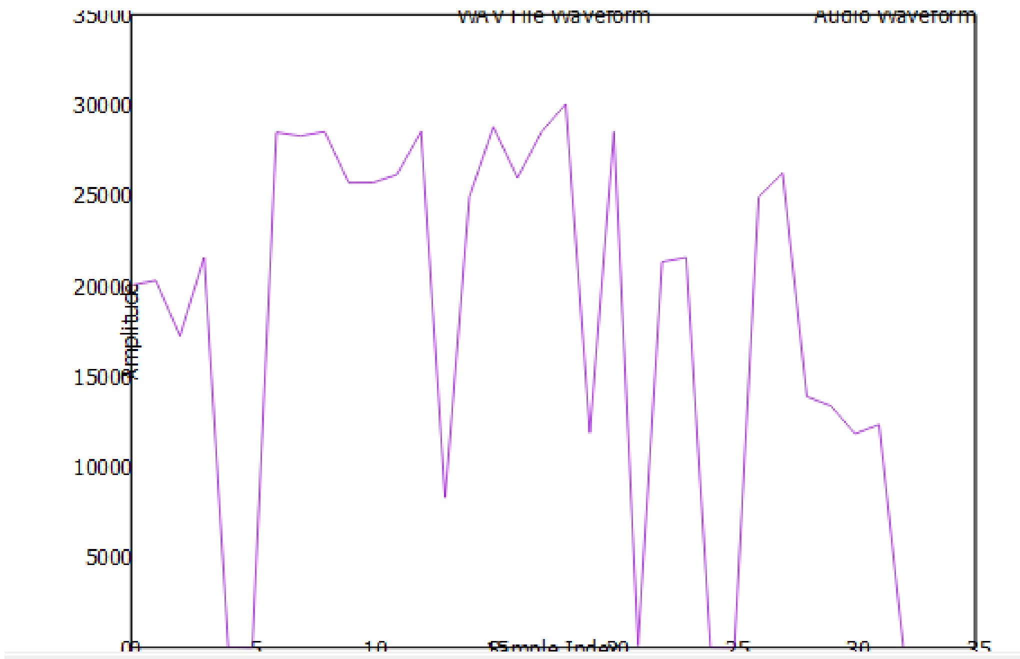


Adaptive LMS - Variable step size implementation

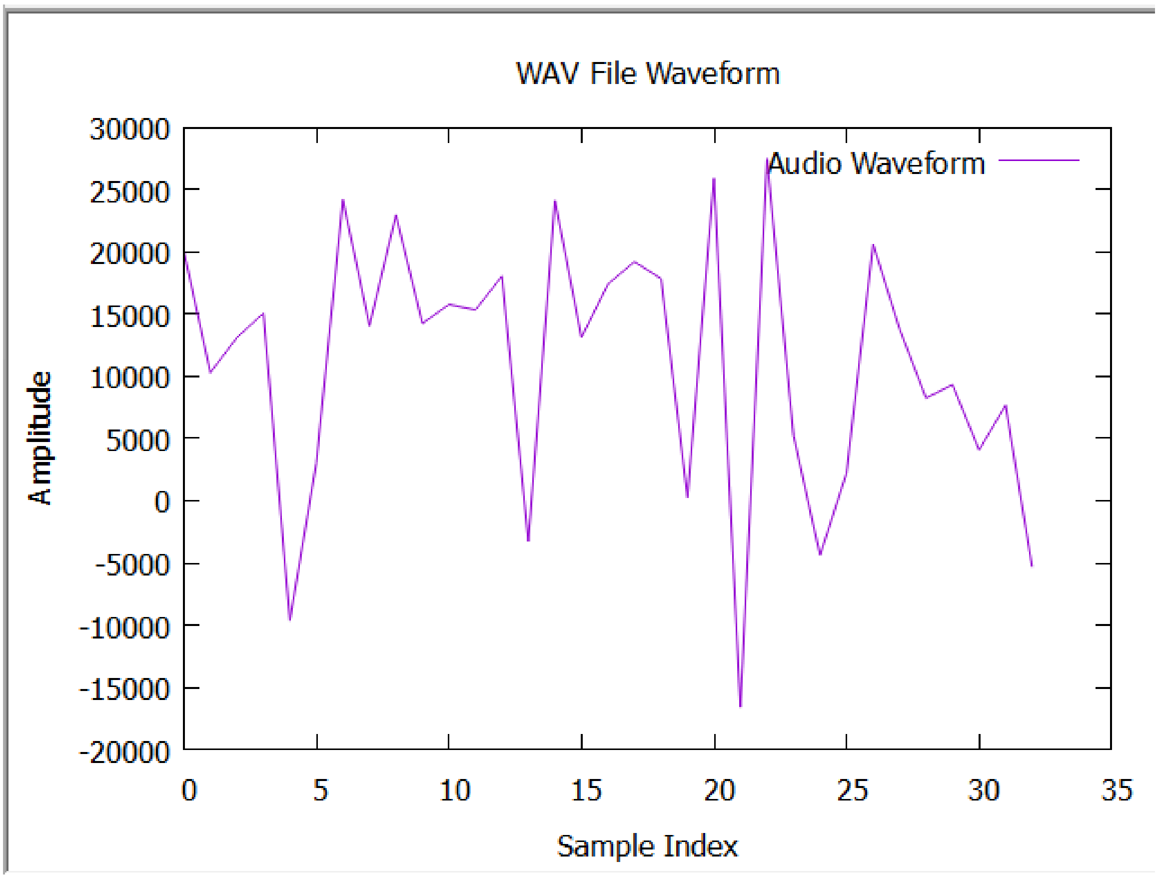


Normal LMS

Clean output obtained after keeping samped sound as the main signal and the noisy audio as the reference background noise to be removed from the source.



Output obtained after passing input to RLS filter.



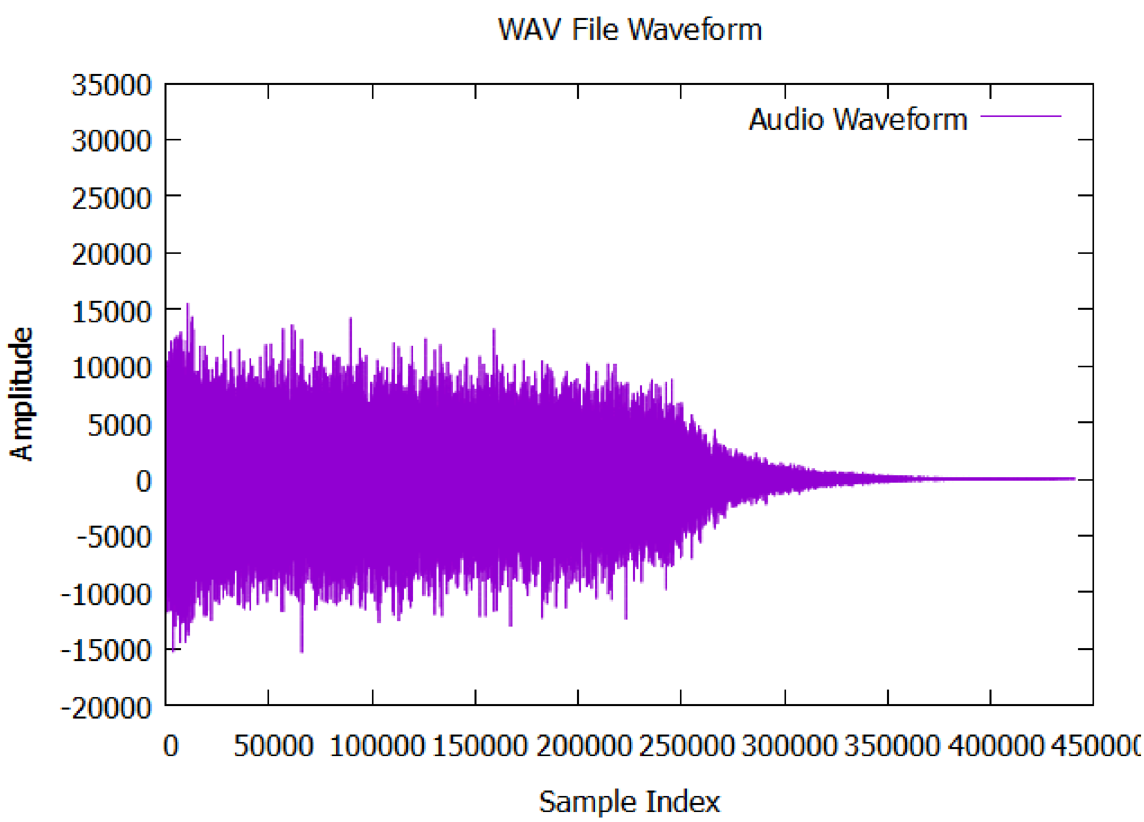
Output obtained by applying predictive analysis to the noisy input audio signal

challenges faced:

```
Biancaa. R@LAPTOP-K1QM67OU UCRT64 /d/Downloads/noise_cancellation_c/lms_audio
$ ./adaptive_noise_cancellation converted_audio.wav noisy_audio.wav cleaned_audio.wav
Noise cancellation completed. Output saved to cleaned_audio.wav

Biancaa. R@LAPTOP-K1QM67OU UCRT64 /d/Downloads/noise_cancellation_c/lms_audio
$ ./plot_wav cleaned_audio.wav

Biancaa. R@LAPTOP-K1QM67OU UCRT64 /d/Downloads/noise_cancellation_c/lms_audio
$ ./plot_wav converted_audio.wav

Biancaa. R@LAPTOP-K1QM67OU UCRT64 /d/Downloads/noise_cancellation_c/lms_audio
$ ./plot_wav noisy_audio.wav

Biancaa. R@LAPTOP-K1QM67OU UCRT64 /d/Downloads/noise_cancellation_c/lms_audio
$ ./adaptive_noise_cancellation noisy_audio.wav white_noise.wav cleaned_audio.wav
Error: Mismatched file sizes!
```

```
D:\Downloads\noise_cancellation_c\lms_audio>ffprobe -i noisy_mono.wav -show_entries format=duration -of compact=p=0:nk=1
ffprobe version 7.1-essentials_build-www.gyan.dev Copyright (c) 2007-2024 the FFmpeg developers
  built with gcc 14.2.0 (Rev1, Built by MSYS2 project)
  configuration: --enable-gpl --enable-version3 --enable-static --disable-w32threads --disable-autodetect --enable-fontconfig --enable-iconv --enable-gnutls
 --enable-libxml2 --enable-gmp --enable-bzlib --enable-lzma --enable-zlib --enable-libsrt --enable-libssh --enable-libzmq --enable-avisynth --enable-sdl2 --
enable-libwebp --enable-libx264 --enable-libx265 --enable-libxvid --enable-libaom --enable-libopenjpeg --enable-libvpx --enable-mediafoundation --enable-lib
ass --enable-libfreetype --enable-libfribidi --enable-libharfbuzz --enable-libvidstab --enable-libvmaf --enable-libzimg --enable-amf --enable-cuda-llvm --en
able-cuvid --enable-dxva2 --enable-d3d11va --enable-d3d12va --enable-ffnvcodec --enable-libvpl --enable-nvdec --enable-nvenc --enable-vaapi --enable-libgme
--enable-libopenmpt --enable-libopencore-amrwb --enable-libmp3lame --enable-libtheora --enable-libvo-amrwbenc --enable-libgsm --enable-libopencore-amrnb --e
nable-libopus --enable-libspeex --enable-libvorbis --enable-librubberband
  libavutil      59. 39.100 / 59. 39.100
  libavcodec     61. 19.100 / 61. 19.100
  libavformat    61.  7.100 / 61.  7.100
  libavdevice    61.  3.100 / 61.  3.100
  libavfilter    10.  4.100 / 10.  4.100
  libswscale      8.  3.100 /  8.  3.100
  libswresample   5.  3.100 /  5.  3.100
  libpostproc    58.  3.100 / 58.  3.100
Input #0, wav, from 'noisy_mono.wav':
  Metadata:
    comment         : Downloaded from Samplefocus.com
    encoder         : Lavf61.7.100
  Duration: 00:00:10.00, bitrate: 705 kb/s
  Stream #0:0: Audio: pcm_s16le ([1][0][0][0] / 0x0001), 44100 Hz, 1 channels, s16, 705 kb/s
10.000000
```

```
D:\Downloads\noise_cancellation_c\lms_audio>ffprobe -i white_mono.wav -show_entries format=duration -of compact=p=0:nk=1
ffprobe version 7.1-essentials_build-www.gyan.dev Copyright (c) 2007-2024 the FFmpeg developers
  built with gcc 14.2.0 (Rev1, Built by MSYS2 project)
  configuration: --enable-gpl --enable-version3 --enable-static --disable-w32threads --disable-autodetect --enable-fontconfig --enable-iconv --enable-gnutls
 --enable-libxml2 --enable-gmp --enable-bzlib --enable-lzma --enable-zlib --enable-libsrt --enable-libssh --enable-libzmq --enable-avisynth --enable-sdl2 --
enable-libwebp --enable-libx264 --enable-libx265 --enable-libxvid --enable-libaom --enable-libopenjpeg --enable-libvpx --enable-mediafoundation --enable-lib
ass --enable-libfreetype --enable-libfribidi --enable-libharfbuzz --enable-libvidstab --enable-libvmaf --enable-libzimg --enable-amf --enable-cuda-llvm --en
able-cuvid --enable-dxva2 --enable-d3d11va --enable-d3d12va --enable-ffnvcodec --enable-libvpl --enable-nvdec --enable-nvenc --enable-vaapi --enable-libgme
--enable-libopenmpt --enable-libopencore-amrwb --enable-libmp3lame --enable-libtheora --enable-libvo-amrwbenc --enable-libgsm --enable-libopencore-amrnb --e
nable-libopus --enable-libspeex --enable-libvorbis --enable-librubberband
  libavutil      59. 39.100 / 59. 39.100
  libavcodec     61. 19.100 / 61. 19.100
  libavformat    61.  7.100 / 61.  7.100
  libavdevice    61.  3.100 / 61.  3.100
  libavfilter    10.  4.100 / 10.  4.100
  libswscale      8.  3.100 /  8.  3.100
  libswresample   5.  3.100 /  5.  3.100
  libpostproc    58.  3.100 / 58.  3.100
Input #0, wav, from 'white_mono.wav':
  Metadata:
    encoder         : Lavf61.7.100
  Duration: 00:00:10.00, bitrate: 705 kb/s
  Stream #0:0: Audio: pcm_s16le ([1][0][0][0] / 0x0001), 44100 Hz, 1 channels, s16, 705 kb/s
10.000000
```

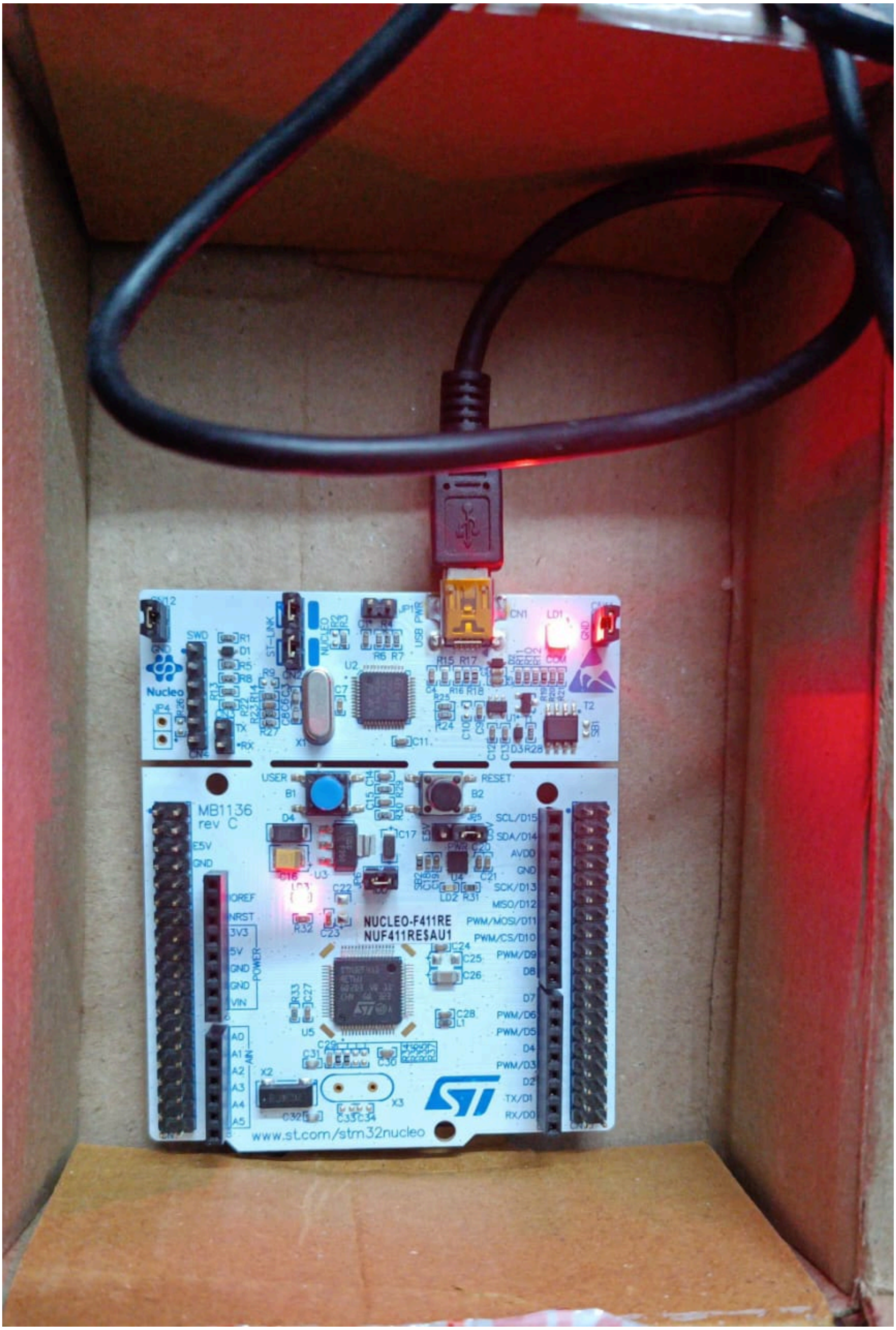


Converted noisy input signal - after formatting

Ouput from STM32:

```
Input: 0.81 --> output: 0.70439
                                Input: 0.59 --> output: 0.509708
                                                                 Input: 0.31 -->
 output: 0.26688
                 Input: 0 --> output: 0
                                        Input: -0.31 --> output: -0.26662
                                                                          Input:
 -0.59 --> output: -0.50696
                            Input: -0.81 --> output: -0.69357
                                                              Input: -0.95 --> o
utput: -0.80811
                Input: -1 --> output: -0.84296
                                               Input: -0.95 --> output: -0.79281

 Input: -0.81 --> output: -0.66987
                                   Input: -0.59 --> output: -0.48473
                                                                     Input: -0.3
1 --> output: -0.2538
                      Input: 0 --> output: 0
                                             Input: 0.31 --> output: 0.253558
                                                                             In
put: 0.59 --> output: 0.482115
                               Input: 0.81 --> output: 0.659583
                                                                Input: 0.95 -->
output: 0.768509
                 Input: 1 --> output: 0.801656
                                               Input: 0.95 --> output: 0.753958
                                                                              I
nput: 0.81 --> output: 0.637046
                                Input: 0.59 --> output: 0.460977
                                                                 Input: 0.31 -->
 output: 0.241365
                  Input: 0 --> output: 0
                                         Input: -0.31 --> output: -0.24113
                                                                           Input
: -0.59 --> output: -0.45849
                             Input: -0.81 --> output: -0.62726
                                                               Input: -0.95 -->
output: -0.73085
                 Input: -1 --> output: -0.76237
                                                Input: -0.95 --> output: -0.7170
1
  Input: -0.81 --> output: -0.60583
                                    Input: -0.59 --> output: -0.43839
                                                                      Input: -0.
31 --> output: -0.22954
```

External setup connected:


MB1136
rev C
NUCLEO-F411RE
NUF411RE$AU1
www.st.com/stm32nucleo

Noise input:

1. https://www.audiocheck.net/testtones_whitenoise.php
2. https://en.wikipedia.org/wiki/DBFS

Clipper:

https://www.aconvert.com/audio/split/

File: