Санкт-Петербургский Государственный Электротехнический

Университет "ЛЭТИ"

кафедра физики

Задание №4 по дисциплине

"Физические основы информационных технологий"

Название: Фильтрация звукового сигнала

Фамилия И.О.:	Бутыло Е. А.
группа:	1303
Преподаватель:	Альтмарк А. М.
Итоговый балл:	
Крайний срок сдачи:	5.12.23

Санкт-Петербург 2023

Условие задания

На входе приёмника получен звуковой сигнал в двоичном коде (рис.1.). Необходимо перевести двоичный код в десятичный и затем провести над аналоговым сигналом процедуру фильтрации от высокочастотных помех. Для фильтрации необходимо использовать пассивные фильтры (фильтры без дополнительного источника питания), которые могут в себя включать, резисторы, конденсаторы и катушки индуктивности.

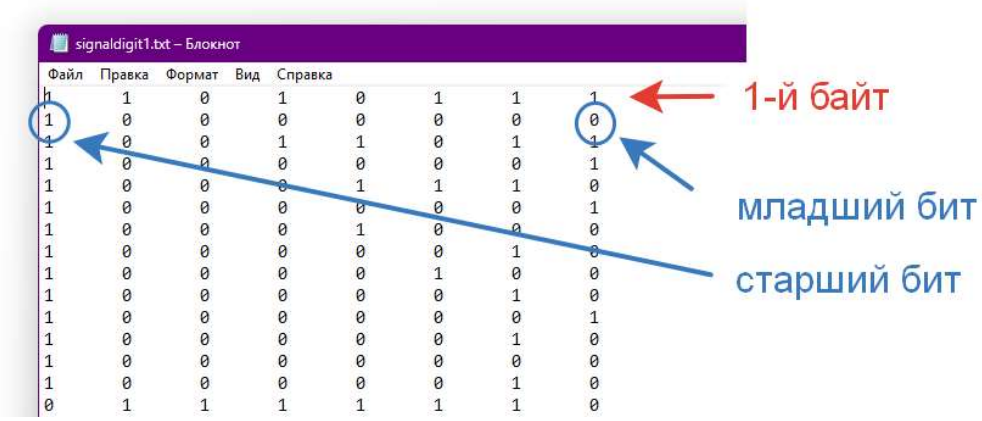

Рис.1. Структура данных в текстовом файле с сигналом

Исходные данные нужно взять в файле FOIT_IDZ4.xlsx. В отчет нужно включить график сигнала во временной области и его спектр, схему фильтра и АЧХ его передаточной функции, спектр фильтрованного сигнала, а также график выходного сигнала во временной области. Файл IDZ4.txt должен содержать ответ на вопрос, который записан в звуком сигнале.

Помимо текстового файла IDZ4.txt в папке IDZ4 должен находиться Word-файл с отчетом, а также файл с кодом (Python, Mathcad, Mathematica). Для лучшего понимания отчетности смотрите папку “Пример организации яндекс-папки студентов”.

Исходные данные

Вар	длительность сигнала, с	Файл с сигналом
3	4	Signaldigit3.txt

Теоретические сведения

В качестве фильтра низких частот в работе использовался фильтр Баттерворда. Фильтр Баттерворта проектируется так, чтобы его амплитудно-частотная характеристика была максимально гладкой на частотах полосы пропускания. Он позволяет эффективно подавить высокие частоты шума и распознать сообщение. Изображение фильтра представлено на рисунке 2.

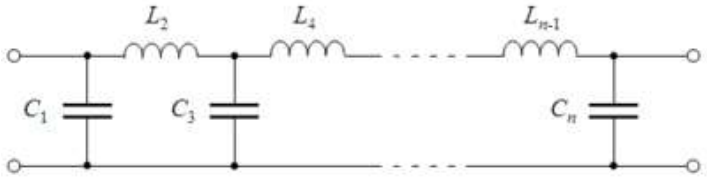

Рис. 2. Фильтр Баттерворта

ПРИЛОЖЕНИЕ А
ПРОГРАММА MAIN.PY

```
import matplotlib.pyplot as plt
import sounddevice as sd
from scipy.fft import fft, ifft
import numpy as np


def play_sound(signal):
    sd.play(np.real(signal), 45000)
    sd.wait()


def draw_plot(*args, title) -> None:
    if len(args) == 2:
        plt.plot(args[0], args[1])
        plt.title(title)
        plt.show()


def initialization(filename: str, dur_time: float) -> tuple[list[int],
list[float], float]:
    with open(filename, 'r') as signal_file:
        content = signal_file.read()

    content = ''.join(content.split('\t'))
    content = content.split('\n')

    signal = [int(elem, 2) for elem in content]
    n = len(signal)

    delta_t = dur_time / n
    timeline = [i * delta_t for i in range(n)]

    dt = 1 / n

    return signal, timeline, dt


def filter_freq(freq_range):
    coefficient = 2
    L1 = 12.4 * pow(10, -3) * coefficient
    L2 = 14.4 * pow(10, -3) * coefficient
    L3 = 12 * pow(10, -3) * coefficient
    L4 = 8.3 * pow(10, -3) * coefficient
    L5 = 3.7 * pow(10, -3) * coefficient
    C1 = 5.9 * pow(10, -6) * coefficient
    C2 = 5.4 * pow(10, -6) * coefficient
    C3 = 4.1 * pow(10, -6) * coefficient
    C4 = 2.4 * pow(10, -6) * coefficient
```

```
    C5 = 497.9 * pow(10, -9) * coefficient
    R = 50

    def ZL(L, omega):
        return 1j * omega * L

    def ZC(C, omega):
        return 1 / (1j * omega * C)

    def H(omega):
        Z5par = 1 / ((1 / R) + 1 / (ZC(C5, omega)))
        Z4par = 1 / ((1 / ZC(C4, omega)) + 1 / (Z5par + ZL(L5, omega)))
        Z3par = 1 / (1 / (ZC(C3, omega)) + 1 / (Z4par + ZL(L4, omega)))
        Z2par = 1 / (1 / (ZC(C2, omega)) + 1 / (Z3par + ZL(L3, omega)))
        Z1par = 1 / (1 / (ZC(C1, omega)) + 1 / (Z2par + ZL(L2, omega)))
        ZL1 = ZL(L1, omega)
        Zall = ZL1 + Z1par
        Iin = 1 / Zall
        Upar1 = Iin * Z1par
        I1top = Upar1 / (Z2par + ZL(L2, omega))
        Upar2 = I1top * Z2par
        I2top = Upar2 / (Z3par + ZL(L3, omega))
        Upar3 = I2top * Z3par
        I3top = Upar3 / (Z4par + ZL(L4, omega))
        Upar4 = I3top * Z4par
        I4top = Upar4 / (Z5par + ZL(L5, omega))
        Upar5 = I4top * Z5par
        return Upar5

    return H(freq_range)


def main() -> None:
    duration: float = 4
    file: str = 'signaldigit3.txt'

    signal, timeline, dt = initialization(file, duration)
    draw_plot(timeline, signal, title='Input signal')

    sign_ampl = fft(signal)
    n = len(sign_ampl)
    freq = np.fft.fftfreq(n, dt)[1:]
    draw_plot(freq, np.abs(sign_ampl[1:]), title='Input spectrum')

    freq_range = np.linspace(1, n, n - 1)
    filtered = np.abs(filter_freq(freq_range))
    plt.xlim(0, 10000)
    draw_plot(freq_range, filtered, title='Filter frequency response')

    H = filtered
    F = sign_ampl[1:] * H
    draw_plot(freq, np.abs(F), title='Result spectrum')
```

```
    changed_signal = ifft(np.concatenate(([0], F)))
    filtered_time_axis = np.linspace(0, 4, len(changed_signal))
    draw_plot(filtered_time_axis, np.real(changed_signal),
title='Result signal')

    # play_sound(changed_signal)


if __name__ == "__main__":
    main()
```