Санкт-Петербургский Государственный Электротехнический Университет "ЛЭТИ"

кафедра физики

Задание №4 по дисциплине

"Физические основы информационных технологий"

Название: Фильтрация звукового сигнала

|  |  |
| --- | --- |
| Фамилия И.О.: | Куклина Ю.Н. |
| группа: | 1303 |
| Преподаватель: | Альтмарк А.М. |
| Итоговый балл: |  |
|  |  |

Крайний срок сдачи: 5.12.23

.

Санкт-Петербург 2023

**Условие задания**

На входе приемника 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.

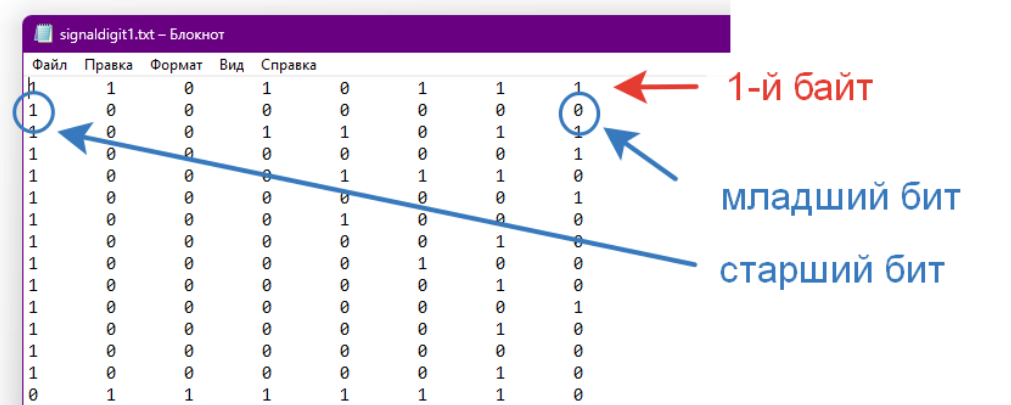


Рис.1. Структура данных в текстовом файле с сигналом

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”.

Пример содержания файла IDZ4.txt:

25

Вариант 7

t = 2.25

**Основные теоретические положения**

Для фильтрации сигнала в работе был использован фильтр Баттерворта. Так как он позволяет достаточно точно обрезать высокие частоты шума и распознать сообщение. Фильтр представлен на рисунке 2.

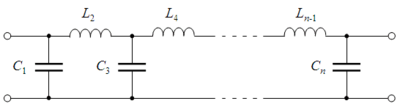


Рис.2. Фильтр Баттерворта низких частот

**ПРИЛОЖЕНИЕ А**

**ИСХОДНЫЙ КОД ПРОГРАММЫ**

import numpy as np

import matplotlib.pyplot as plt

from scipy.fft import fft, ifft

t = 2.25

file\_path = "C:\\Users\\dns\\Desktop\\Учеба\\3 курс\\5 семестр\\ФОИТ\\signaldigit7.txt"

with open(file\_path, 'r') as file:

discret\_signal = [line.strip().split('\t') for line in file]

analog\_signal = [int(''.join(b), 2) for b in discret\_signal]

dt = t / len(discret\_signal)

sample\_rate = 44100

time\_axis = np.linspace(0, t, len(analog\_signal))

plt.plot(time\_axis, analog\_signal)

plt.title('Analog Signal')

plt.xlabel('Time (s)')

plt.ylabel('Amplitude')

plt.show()

fsig = fft(analog\_signal)

out\_n = len(fsig)

df = 1 / dt

freq\_axis = np.fft.fftfreq(out\_n, dt)[1:]

plt.plot(freq\_axis, np.abs(fsig[1:]))

plt.title('Spectrum')

plt.xlabel('Frequency (Hz)')

plt.ylabel('Amplitude')

plt.show()

R1 = 50

C1 = 0.000005

C2 = 0.0000045

C3 = 0.0000035

C4 = 0.000002

C5 = 497.9 \* 10\*\*-9

L1 = 0.01

L2 = 0.012

L3 = 0.01

L4 = 0.007

L5 = 0.003

def butterworth\_filter(w, Uin):

Zpar5 = 1 / (1j \* w \* C5 + 1 / R1)

Zpar4 = 1 / (1j \* w \* C4 + 1 / (1j \* w \* L5 + Zpar5))

Zpar3 = 1 / (1j \* w \* C3 + 1 / (1j \* w \* L4 + Zpar4))

Zpar2 = 1 / (1j \* w \* C2 + 1 / (1j \* w \* L3 + Zpar3))

Zpar = 1 / (1j \* w \* C1 + 1 / (1j \* w \* L2 + Zpar2))

I1 = Uin / (1j \* w \* L1 + Zpar)

Upar = I1 \* Zpar

I2 = Upar / (1j \* w \* L2 + Zpar2)

Upar2 = I2 \* Zpar2

I3 = Upar2 / (1j \* w \* L3 + Zpar3)

Upar3 = I3 \* Zpar3

I4 = Upar3 / (1j \* w \* L4 + Zpar4)

Upar4 = I4 \* Zpar4

I5 = Upar4 / (1j \* w \* L5 + Zpar5)

Upar5 = I5 \* Zpar5

Uout = Upar5

return Uout / Uin

Uin\_value = 1

freq\_range = np.linspace(1, out\_n/10, out\_n - 1)

filter\_response = np.abs(butterworth\_filter(freq\_range, Uin\_value))

plt.plot(freq\_range, filter\_response)

plt.title('Normalized Filter Frequency Response')

plt.xlabel('Frequency (Hz)')

plt.ylabel('Amplitude')

plt.show()

H\_list = filter\_response

F\_new = fsig[1:] \* H\_list

plt.plot(freq\_axis, np.abs(F\_new))

plt.title('Filtered Spectrum')

plt.xlabel('Frequency (Hz)')

plt.ylabel('Amplitude')

plt.show()

changed\_signal = ifft(np.concatenate(([0], F\_new)))

filtered\_time\_axis = np.linspace(0, t, len(changed\_signal))

plt.plot(filtered\_time\_axis, np.real(changed\_signal))

plt.title('Filtered Signal')

plt.xlabel('Time (s)')

plt.ylabel('Amplitude')

plt.show()

import sounddevice as sd

sd.play(np.real(changed\_signal), sample\_rate)

sd.wait()