МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ»
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)
Институт №8 «Компьютерные науки и прикладная математика»
Кафедра 806: «Вычислительная математика и программирование»

ОТЧЕТ О ВЫПОЛНЕНИИ ЛАБОРАТОРНОЙ РАБОТЫ №4

По курсу: «Цифровая обработка изображений»

Тема: «Фильтрация цифровых сигналов»

Студент: Чернышёв Д.В.
Группа: М8О-107М-22
Вариант: 8
Преподаватель: Гаврилов К.Ю.

Москва
2023

СОДЕРЖАНИЕ

1 Задание к лабораторной работе №4

Часть 1

Сформируйте сигнал в виде суммы трех гармонических колебаний с частотами 100 Гц, 1 кГц, 10 кГц. Длительность сигнала составляет 1 с. Постройте
спектр результирующего сигнала и определите области всех гармонических составляющих. Создайте фильтр, выделяющий только сигнал с частотой 100 Гц. Постройте импульсную и частотную характеристики фильтра. Выполните фильтрацию во временной области путем использования функции свертки сигнала с импульсной характеристикой (ИХ) фильтра. Выполните фильтрацию сигнала в частотной области с помощью операции ДПФ. Сравните результаты фильтрации при использовании обоих способов и проанализируйте сигнал на выходе фильтра. Постройте спектр выходного сигнала.

Часть 2

Дан аудиофайл с записью голоса, на который наложена помеха. Постройте спектр сигнала и определите частотный состав помехи. Постройте режекторный фильтр, подавляющий частоты помехи. С помощью полученного фильтра подавите помехи в заданном сигнале и определите голосовую информацию, содержащуюся заданной аудиозаписи. Постройте спектр отфильтрованного сигнала, сравните его со спектром исходного сигнала и проведите анализ результатов фильтрации. Рекомендация. Для лучшей фильтрации очень сильной помехи, во много раз превосходящей по мощности полезный сигнал, целесообразно в качестве ЧХ фильтра использовать произведение одной и той же ЧХ РФ на себя (2, 3 или более раз).

2 Выполнение лабораторной работы №4

Импортируем необходимые библиотеки

```
from scipy.signal import kaiserord, lfilter, \
    firwin, freqz
import wave as we
import numpy as np
import matplotlib.pyplot as plt

from scipy import signal
from scipy.io import wavfile
from scipy.fft import fft, fftshift, fftfreq,\
    rfft, rfftfreq, irfft,ifft


import scipy.io.wavfile as wav
from scipy import signal

import os

import soundfile as sf
from scipy.signal import filtfilt, iirnotch, freqz, \
    butter,convolve

import librosa

from pydub import AudioSegment
from pydub import AudioSegment
from pydub.playback import play
```

Часть 1

Формирование сигнала

```
# Задаем параметры сигналов
f1 = 100  # частота первого колебания в Гц
f2 = 1000  # частота второго колебания в Гц
f3 = 10000  # частота третьего колебания в Гц
T = 1  # длительность сигнала в секундах
Fs = 5000  # частота дискретизации в Гц

# Создаем временную ось
t = np.linspace(0, T, Fs)

# Генерируем гармонические сигналы
x1 = np.sin(2 * np.pi * f1 * t)
x2 = np.sin(2 * np.pi * f2 * t)
x3 = np.sin(2 * np.pi * f3 * t)

# Складываем сигналы
x = x1  + x2 + x3

# Отображаем график сигнала
plt.figure(figsize=(10,10))

plt.subplot(2, 1, 1)
plt.plot(t, x)
plt.xlim([0, 0.1])
plt.xlabel('Время (с)')
plt.ylabel('Амплитуда')
plt.title('Сложенный сигнал')


plt.subplot(2, 1, 2)
plt.plot(t, x)
plt.xlim([0, 0.6])
```

```
33 plt.xlabel('Время (с)')
34 plt.ylabel('Амплитуда')
35 plt.title('Сложенный сигнал')
36
37 plt.savefig(os.path.join(OUTPUT_DIR,f"ComplexSignalInput"))
```

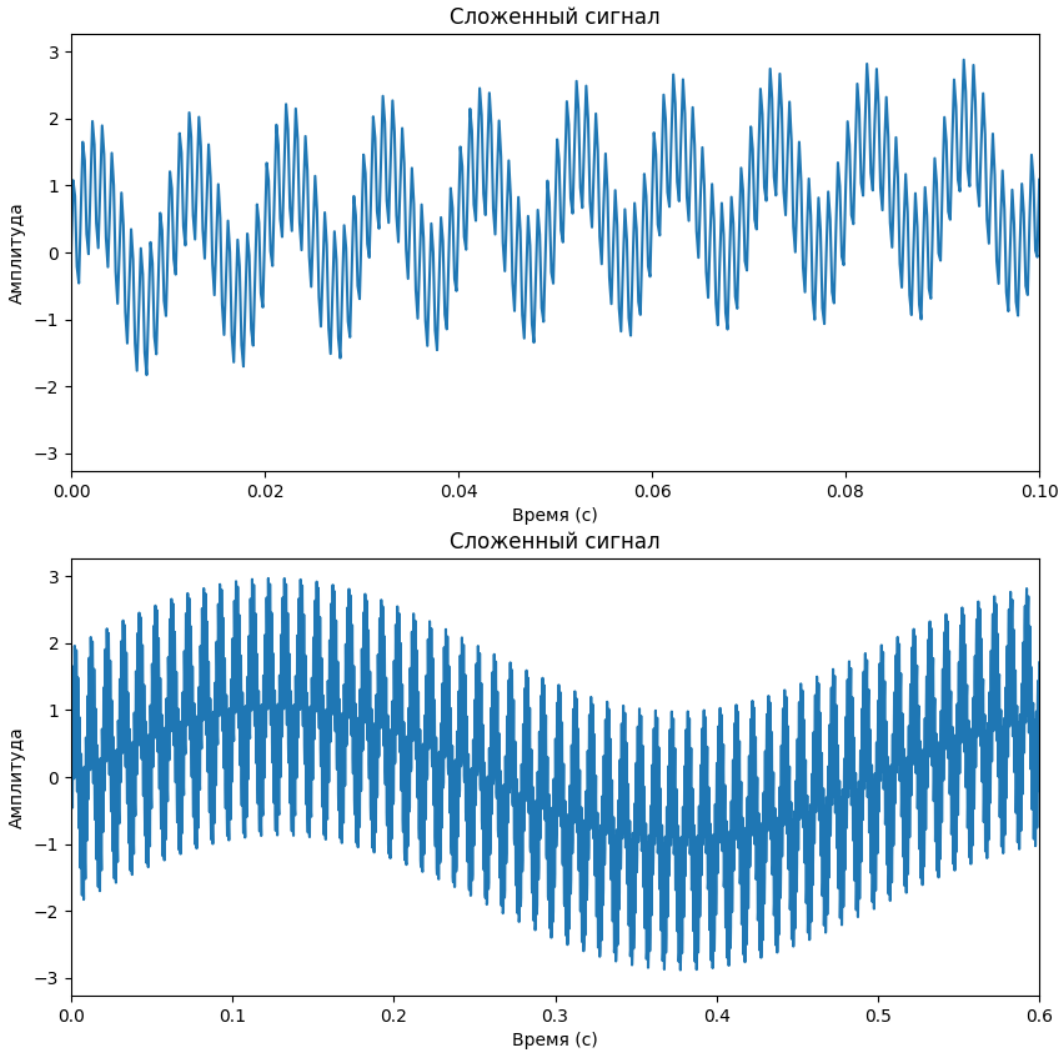


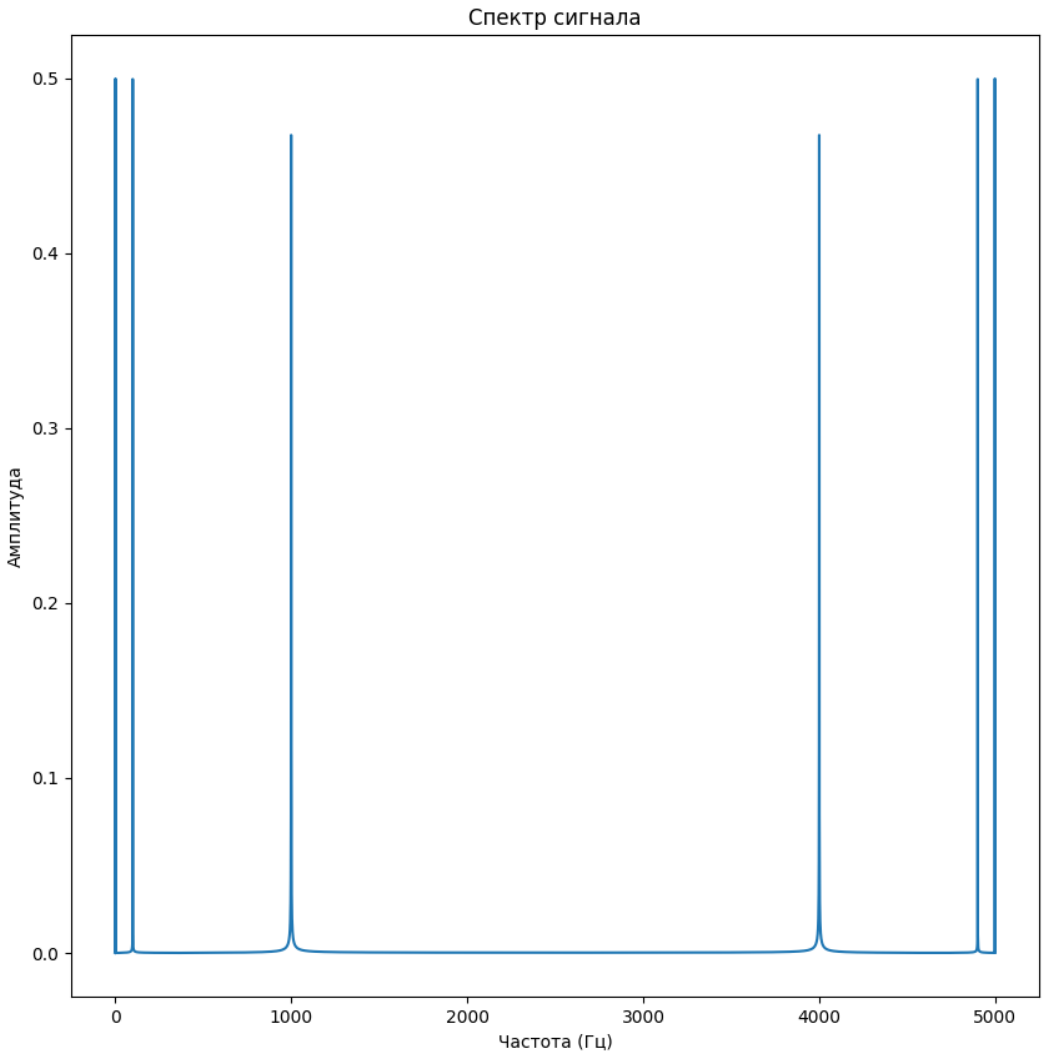
Рис. 1: Спектр сигнала

Рассчитываем спектр

```
N = len(x)  # число отсчетов
X = np.abs(fft(x)) / N  # амплитудный спектр
f = np.linspace(0, N , N)  # частоты

# Отображаем график спектра
plt.figure(figsize=(10,10))
plt.plot(f, X)
# plt.xlim([0, 1000])
plt.xlabel('Частота (Гц)')
plt.ylabel('Амплитуда')
# plt.xticks(
    np.arange(
        f.min(),
        f.max(),
        int((f.max()-f.min())/10) + 1
        )
    )
plt.title('Спектр сигнала')


plt.savefig(os.path.join(OUTPUT_DIR,f"SpectrumInput"))
```



Задаем параметры фильтра

```
fc = 100  # частота среза фильтра в Гц
order = 2  # порядок фильтра

# Создаем фильтр
b, a = butter(order, fc / (Fs / 2), 'low')

# Фильтруем сигнал
filtered_signal = convolve(x, b / a, mode='same')

```

```
# Отображаем график сигнала до и после фильтрации
plt.figure(figsize=(10,10))
# plt.subplot(2, 1, 1)
plt.plot(t, x)
plt.xlim([0, 0.1])
plt.xlabel('Время (с)')
plt.ylabel('Амплитуда')
plt.title('Сигнал до фильтрации')




plt.savefig(os.path.join(OUTPUT_DIR,f"BeforeFiltered"))
```

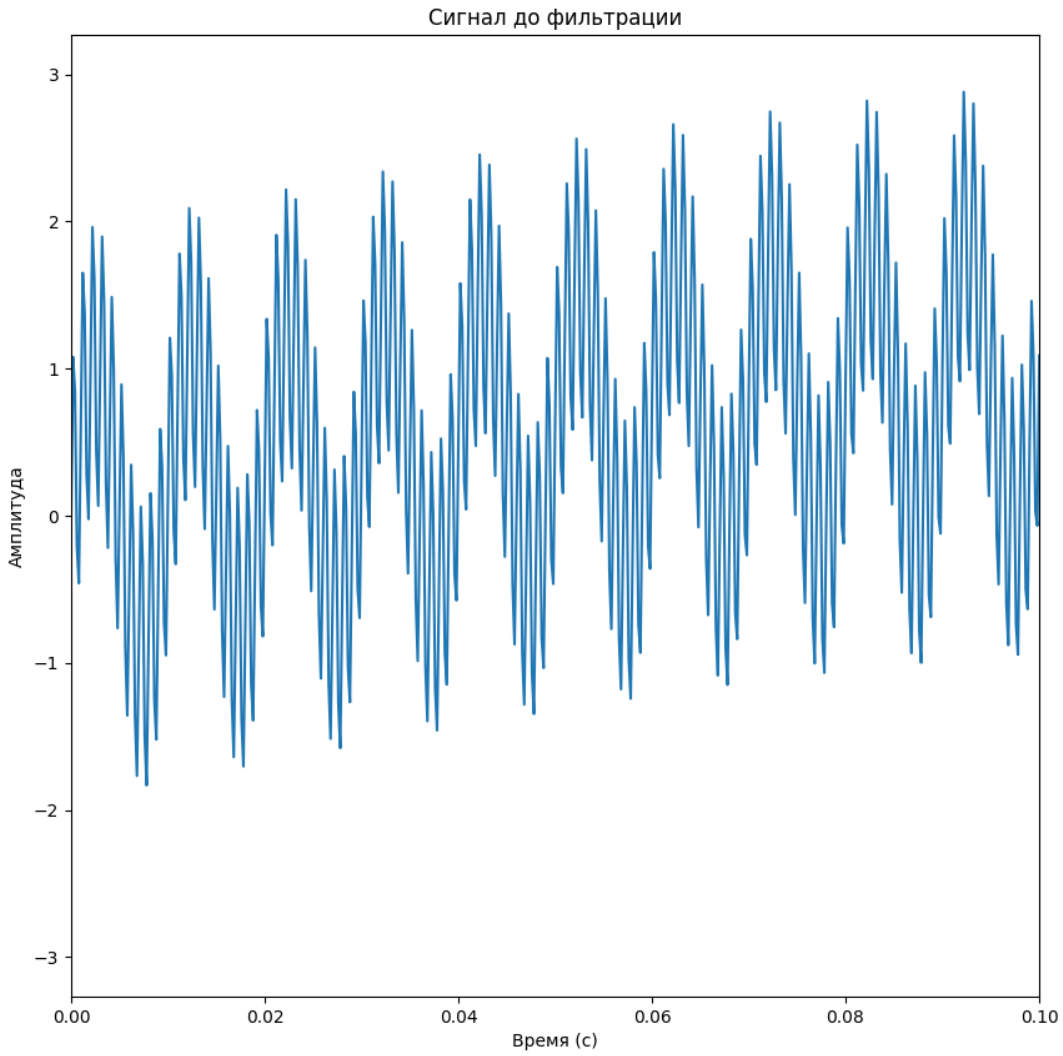
Рассчитываем импульсную характеристику

```
impulse_response = np.zeros(len(t))
impulse_response[:len(b)] = b

# Рассчитываем частотную характеристику
w, h = freqz(b, a, len(t), Fs)

# Отображаем графики импульсной и частотной характеристик
plt.figure(figsize=(10,10))
plt.subplot(2, 1, 1)
plt.stem(impulse_response)
plt.xlabel('Отсчеты')
plt.ylabel('Амплитуда')
plt.title('Импульсная характеристика')

plt.subplot(2, 1, 2)
plt.plot(w, 20 * np.log10(abs(h)))
plt.xlabel('Частота (Гц)')
plt.ylabel('Амплитуда (дБ)')
```

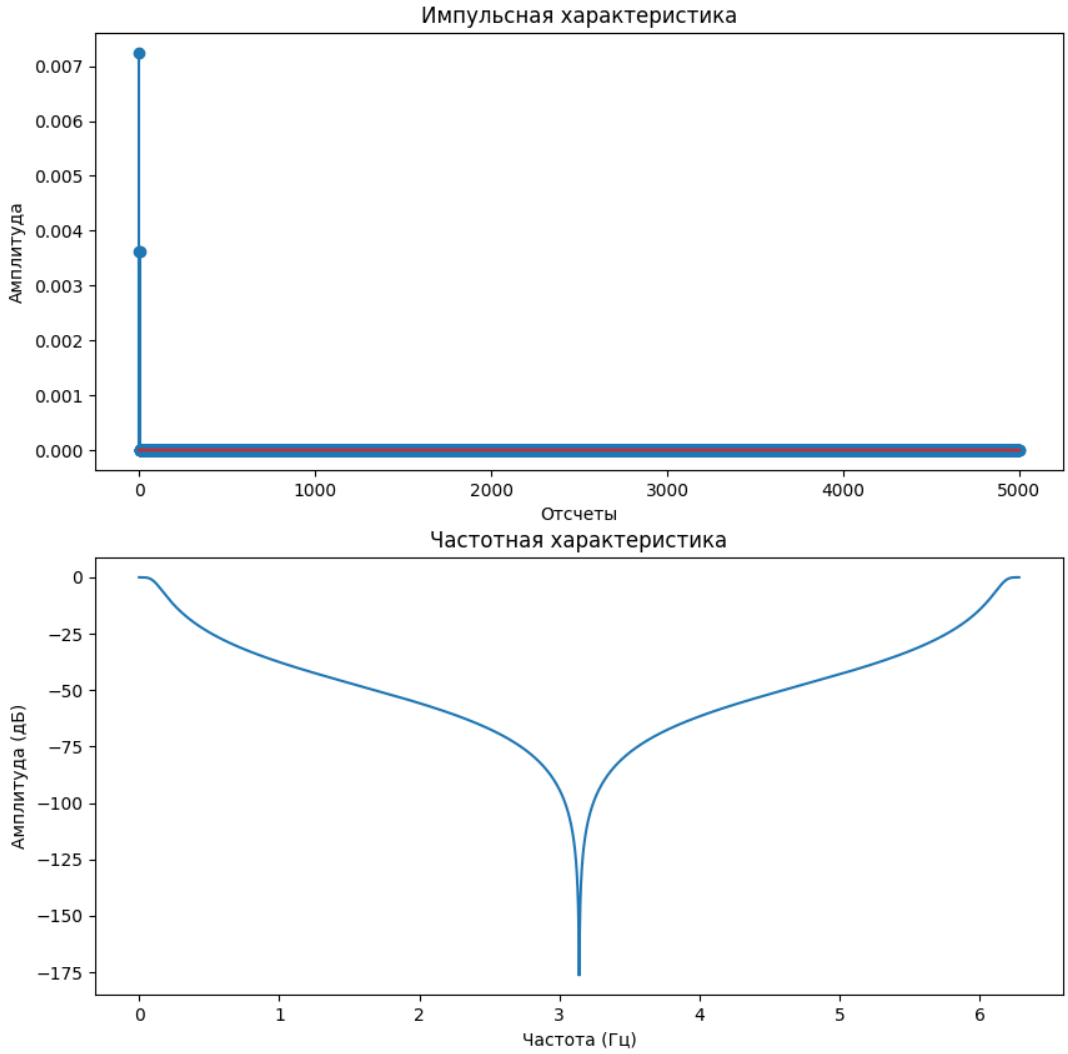



```
19 plt.title('Частотная характеристика')
20
21
22
23 plt.savefig(
24     os.path.join(
25         OUTPUT_DIR,
26         f"Графики_импульсной_и_частотной_характеристик"
27         )
28     )
```

```
29
30 # Свертка сигнала с импульсной характеристикой
31 signal = x
32 # Выполняем фильтрацию сигнала
33 filtered_signal = convolve(
34         signal,
35         impulse_response,
36         mode='same'
37 )
38
39 # Отображаем графики исходного и отфильтрованного сигналов
40 plt.figure(figsize=(10,10))
41 plt.subplot(2, 1, 1)
42 plt.plot(t, signal)
43 plt.xlim([0, 0.1])
44 plt.xlabel('Время (с)')
45 plt.ylabel('Амплитуда')
46 plt.title('Исходный сигнал')
```

Фильтрация через ДПФ Выполнить ДПФ для исходного сигнала

```
1  X = np.fft.fft(x)
2
3  # Получить спектральную плотность мощности
4  Pxx = np.abs(X) ** 2
5
6  # Фильтр
7  filt = np.zeros(len(X), dtype=bool)
8  cutoff_freq = 250  # задаем пороговую частоту
9  filt[:cutoff_freq] = True
10 filt[-cutoff_freq:] = True
11
12 # Обратное ДПФ для отфильтрованного спектра
13 Y = X * filt
```



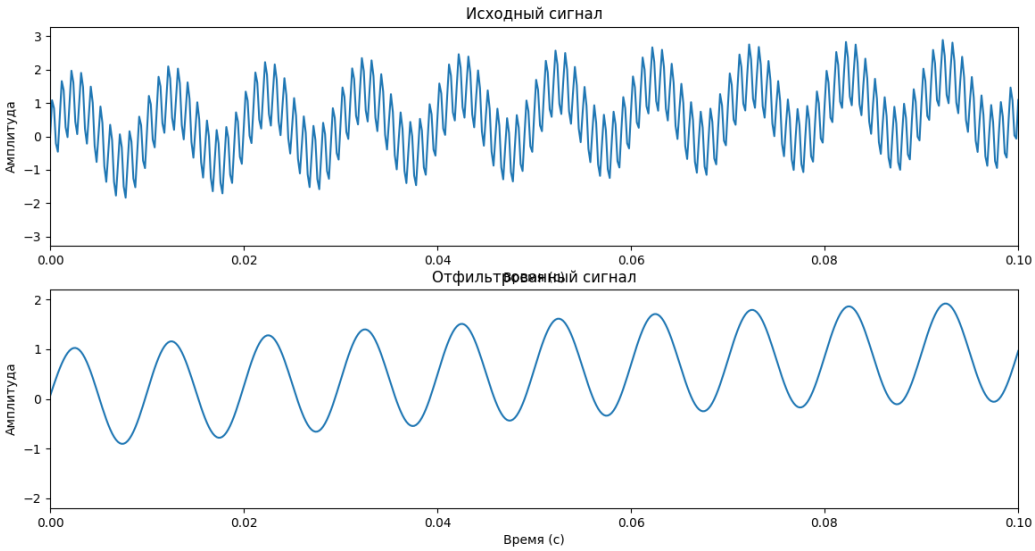

```
14 y = np.real(np.fft.ifft(Y))
15
16 plt.figure(figsize=(14,7))
17 plt.subplot(2, 1, 1)
18 plt.plot(t, x)
19 plt.xlim([0, 0.1])
20 plt.xlabel('Время (c)')
21 plt.ylabel('Амплитуда')
22 plt.title('Исходный сигнал')
23
```

```
plt.subplot(2, 1, 2)
plt.plot(t, y)
plt.xlim([0, 0.1])
plt.xlabel('Время (с)')
plt.ylabel('Амплитуда')
plt.title('Отфильтрованный сигнал')



plt.savefig(os.path.join(OUTPUT_DIR,f"Фильтрация_ДПФ"))

```



Рассчитываем спектр выходного сигнала

```
N = len(y)  # число отсчетов
X = np.abs(np.fft.fft(y)) / N  # амплитудный спектр
f = np.arange(0, N) * Fs / N  # частоты

# Отображаем график спектра
plt.figure(figsize=(10,10))
plt.plot(f, X)
```

```
plt.xlim([0, 500])
plt.xlabel('Частота (Гц)')
plt.ylabel('Амплитуда')
plt.xticks(np.arange(f.min(),500,100))
plt.title('Спектр выходного сигнала')

plt.show()


plt.savefig(os.path.join(OUTPUT_DIR,f"Результат_Фильтрация_ДПФ"))
```

Часть 2

Для начала считаем исходный файл методом ‘wavfile.read‘(метод возвращает частоту дискретизации (в сек) и данные из файла LPCM WAV).

```
fs, Audiodata = wavfile.read(SOURCE_FILE)
# Возвращает частоту дискретизации
# (в сэмплах/сек) и данные из файла LPCM WAV.
print(fs)
# Plot the audio signal in time
print(type(Audiodata))
plt.figure(figsize=(16,8))
plt.plot(Audiodata)
plt.xlim(0,10000)
plt.title('Audio signal in time',size=15)
plt.savefig(os.path.join(OUTPUT_DIR,f"Audio_signal_in_time"))
```

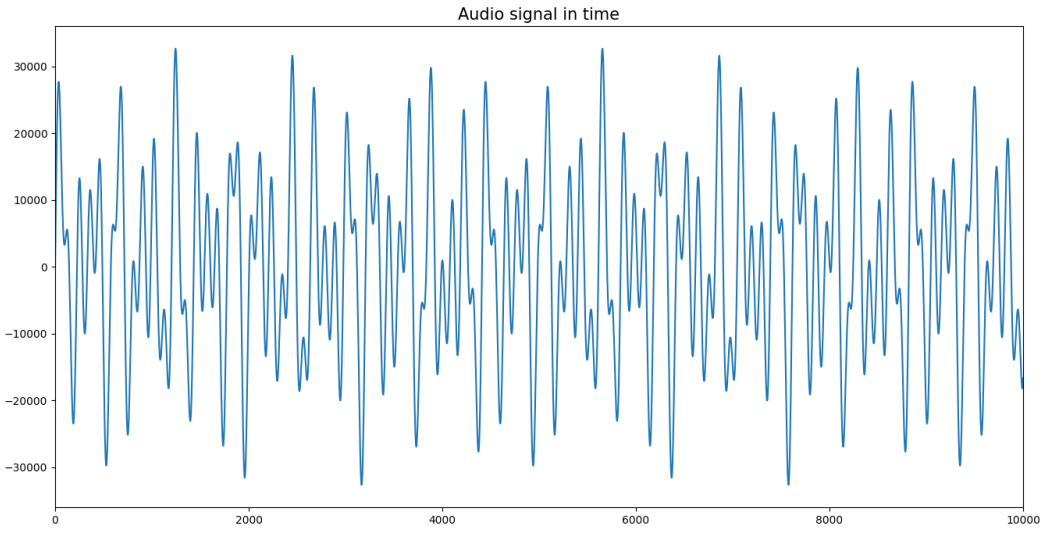


Рис. 2: Исходный аудиосигнал

Построение спектра исходного сигнала

```
 freq, original_spectrum = signal.periodogram(Audiodata, fs=fs)
# Построение спектра исходного сигнала
plt.figure(figsize=(16, 8))
plt.plot(
    freq[1:],
    (10 * np.log10(original_spectrum))[1:],
    linewidth=0.55
    )
plt.xlabel('Frequency (Hz)')
plt.ylabel('Power Spectral Density (dB/Hz)')
plt.title('Original Signal Spectrum')
plt.xticks(np.arange(0, 24000, step=1500))
plt.yticks(
        np.arange(
            np.min(
            (10 * np.log10(original_spectrum))[1:]),
            np.max( (10 * np.log10(original_spectrum))[1:])+10,
            step=10)
            )
plt.grid(True)
plt.show()

```

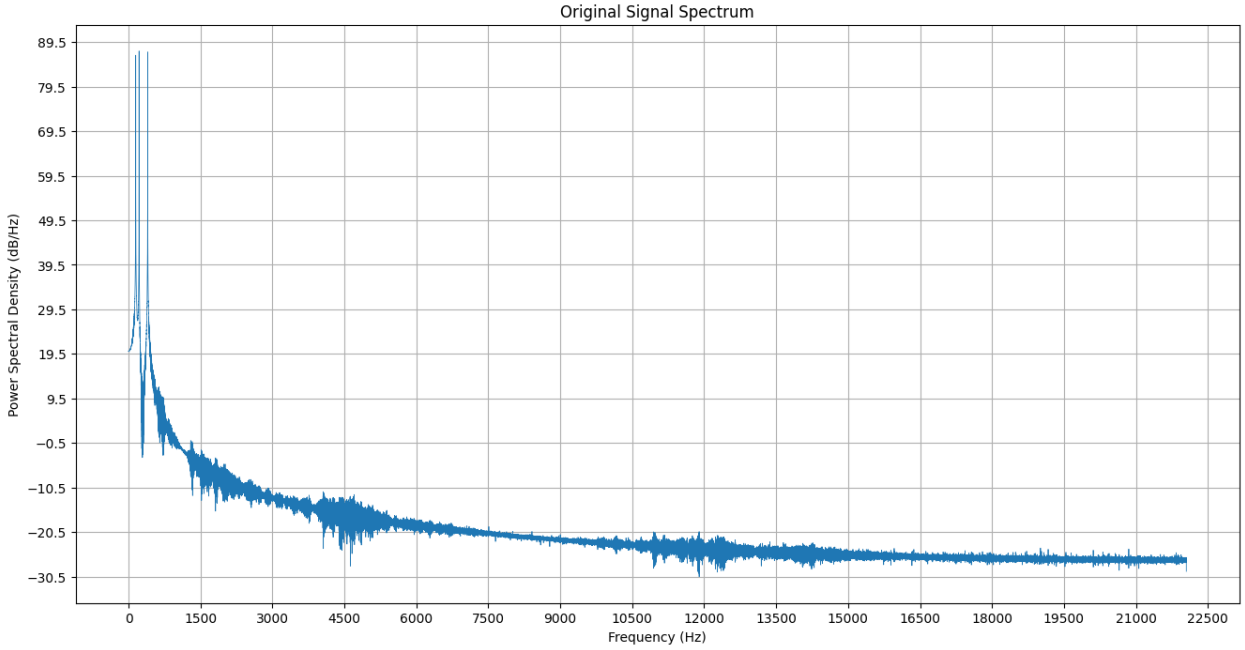


Рис. 3: Исходный аудиосигнал (Спектр)

Автоматическое нахождение частот и ширин помех методом спектрального анализа можно выполнить с помощью алгоритма поиска пиков в спектре сигнала. Вот пример кода для такого автоматического нахождения:

```
from scipy.signal import find_peaks

def find_noise_frequency_spectrum(signal, fs):
    # Вычисление спектра сигнала
    spectrum = np.abs(fft(signal))

    # Определение частотной оси
    freq_axis = fftfreq(len(signal), 1 / fs)

    # Нахождение пиков в спектре
    peaks, _ = find_peaks(spectrum, height=0)

    # Определение частоты и ширины помехи
    peak_freqs = freq_axis[peaks]
    noise_frequency = peak_freqs[np.argmax(spectrum[peaks])]
```

```
    noise_bandwidth = np.sum(
            spectrum[peaks] > 0.5 * np.max(spectrum[peaks])
            )

    return noise_frequency, noise_bandwidth
```

Создание и применение многополосных фильтров Заготовка

```
filtered_audio = Audiodata.copy()
list_of_audios_iter = [filtered_audio]
list_of_titles = ["Input"]

T_list = [1,1]

```

Итерационно обрабатываем сигнал, при этом сохраняя промежуточный вывод

```
ITERATIONS = 4#5
# assert ITERATIONS== len(T_list)
for I in range(ITERATIONS):
  audio_i = list_of_audios_iter[-1]/np.max(
              np.abs(list_of_audios_iter[-1])
              )
  noise_frequency, noise_bandwidth =
       find_noise_frequency_spectrum(
                  audio_i, fs
          )

  print(f"Шаг итерации:{I+1}
    +f"\n\tЧастота помехи:{round(noise_frequency,4)}"
   +f"\n\tШирина помехи:{noise_bandwidth}"
  )

```

```
  # Нормализация частоты и ширины помехи
  normalized_center_freq = noise_frequency / (fs / 2)
  normalized_bandwidth = noise_bandwidth / (fs / 2)

  # Создание полосового фильтра (bandstop filter)
  b, a = signal.iirnotch(
        normalized_center_freq,
        normalized_bandwidth
  )
  # Применение фильтра к сигналу
  # for j in range(T_list[I]):
    audio_i = signal.lfilter(b, a, audio_i)

  list_of_audios_iter.append(audio_i)

  list_of_titles.append(f"Filter_iteration_{I}")

```

Визуализируем

```
plt.figure(figsize=(16, 8))
for i in range(len(list_of_audios_iter)):

    audio,title = list_of_audios_iter[i],list_of_titles[i]
    freq_i, spectrum_i = signal.periodogram(
            list_of_audios_iter[i],
            fs=fs
            )
    plt.plot(
        freq_i[1:],
        (10 * np.log10(spectrum_i))[1:],
        label=title,
        linewidth=1
        )
```

```
15
16  # plt.plot(
17      filtered_freq,
18      10 * np.log10(filtered_spectrum),
19      label='Filtered Signal'
20      )
21  plt.xlabel('Frequency (Hz)')
22  plt.ylabel('Power Spectral Density (dB/Hz)')
23  plt.legend()
24  plt.grid(True)
25  plt.show()
26
27  # plt.savefig(os.path.join(
28      OUTPUT_DIR,
29      f"Comparison_of_Original_and_Filtered_Signal_Spectrums")
30      )
```

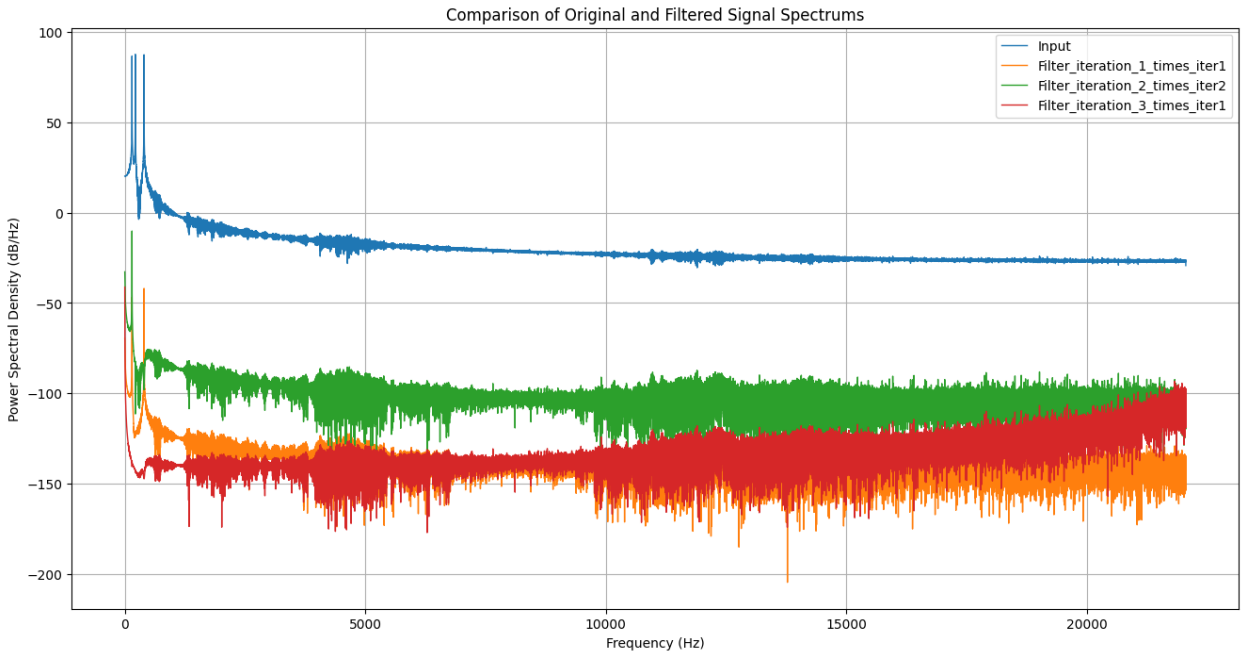


Рис. 4: Сравнение спектров на каждом шаге итерации

Сохраняем

```
 pathes = []
for i in range(len(list_of_audios_iter)):
    audio,title = list_of_audios_iter[i],list_of_titles[i]

    # Задайте путь к файлу и параметры дискретизации
    filename = os.path.join(OUTPUT_DIR,f"{title}.wav")
    pathes.append(filename)
    sample_rate = fs

    # Сохранение отфильтрованного сигнала в файл
    sf.write(
        filename,
        audio/np.max(np.abs(audio)),
        sample_rate, 'PCM_32')

```

Из итогового файла можно узнать что :

На полке в ряд стояли книги разного цвета: 3 красные, 5 синих, 2 зеленые

3 Выводы лабораторной работы №4

Выполнив данную работу были изучены методы методы цифровых и сигналов, обработан зашумленный сигнал, из которого получилось извлечь информацию.