1 Установка пакетов

```
import pandas as pd
import matplotlib.pyplot as plt
```

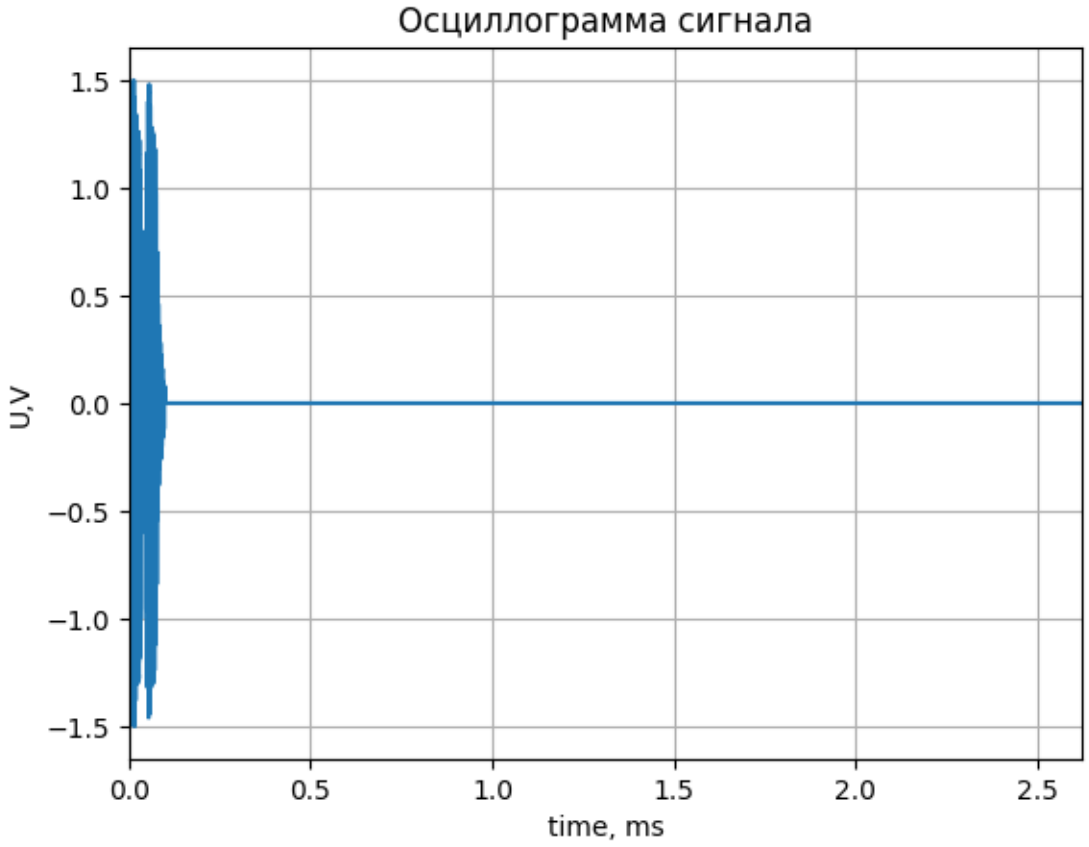
2 Чтение осциллограммы сигнала из *.csv файла

```
format_ver = 0
# file_name = "../data/6кГц/F0004CH1.CSV"  # 1389
# file_name = "../data/9кГц/F0003CH1.CSV"  # 1390
# file_name = "../data/12кГц/F0002CH1.CSV" # 1387
# dds_file = "../ZI/01387.dds"
# file_name = "../data/15кГц/F0001CH1.CSV"  # 1391
file_name = "../data/24кГц/F0000CH1.CSV"
dds_file = "../ZI/01392.dds"
downsampling_factor = 1
inx_start = 1
inx_stop = 2450

from load_and_prepare_data import load_and_prepare_data

t, s, meta_info, meta_df, dt, oversampling_factor, N =
  ↪load_and_prepare_data(
    file_name, format_ver, inx_start, inx_stop, downsampling_factor
)
```

```
Считано 2485 отсчётов
Частота дискретизации = 0  МГц
                 Key               Value
0      Record Length        2.500000e+03
1    Sample Interval        4.000000e-08
2      Trigger Point  7.500000000000e+01
3             Source                 CH1
4     Vertical Units                   V
5     Vertical Scale        5.000000e-01
6    Vertical Offset        0.000000e+00
7   Horizontal Units                   s
8   Horizontal Scale        1.000000e-05
9             Pt Fmt                 ENV
10             Yzero        0.000000e+00
11       Probe Atten        1.000000e+00
12      Model Number            TDS2002B
13     Serial Number             C053511
14  Firmware Version       FV:v22.16 rus
Для анализа берём 65536 отсчётов
```

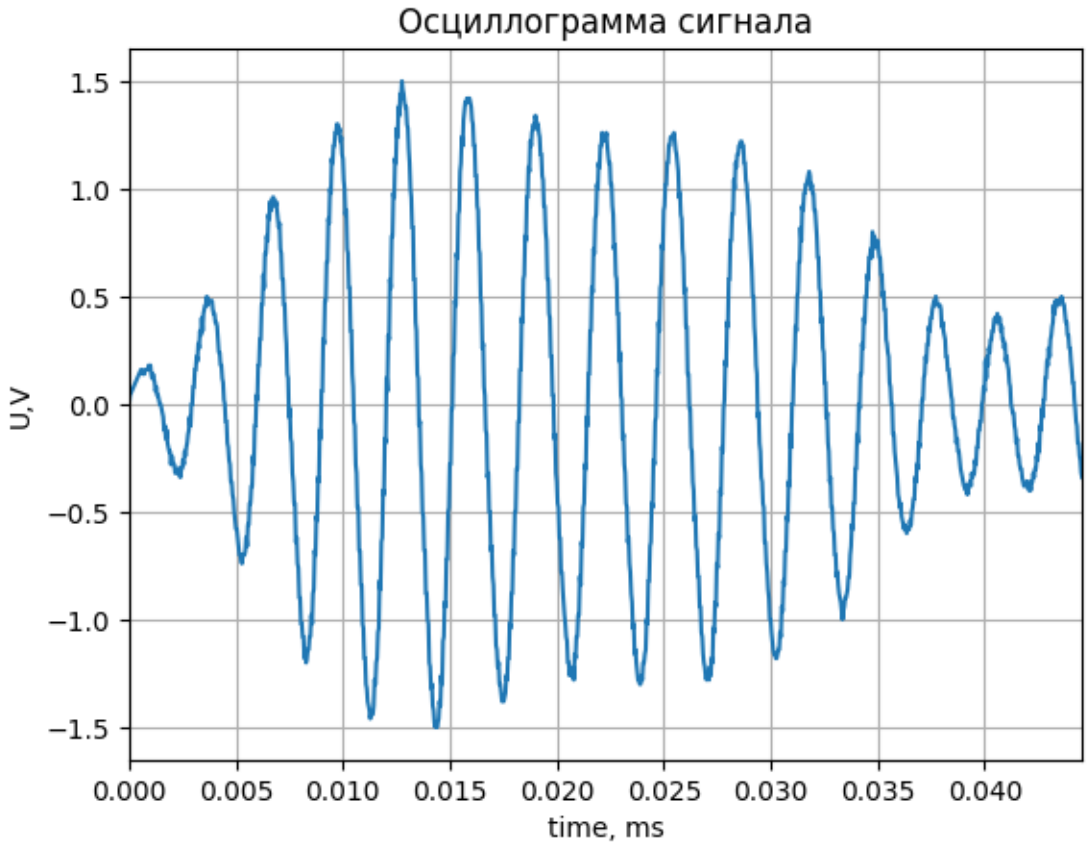
```
Частота дискретизации (новая) = 25  МГц
```

```
import numpy as np

t_ms = np.array(t) * 1000
s_arr = np.array(s)

plt.figure()
plt.plot(t_ms, s_arr)
plt.xlabel("time, ms")
plt.ylabel("U,V")
plt.title("Осциллограмма сигнала")
plt.grid(True)
plt.xlim(0, t_ms[-1])
plt.show()
```




```
plt.figure()
plt.plot(t_ms, s_arr)
plt.xlabel("time, ms")
```

```
plt.ylabel(”U,V”)
plt.title(”Осциллограмма сигнала”)
plt.grid(True)
plt.xlim(0, t_ms[-1] / oversampling_factor / 2.2)
plt.show()
```



3 Спектральный анализ

3.1 БПФ (fft)

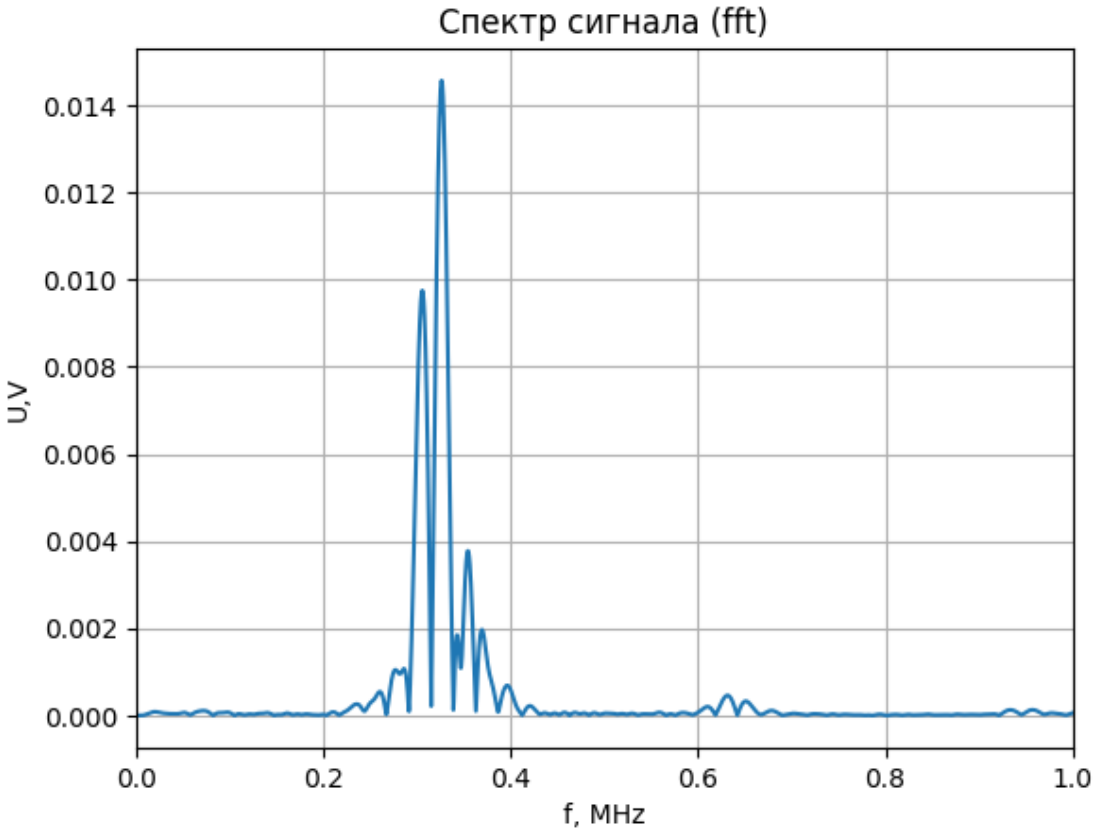
Применим к сигналу функцию fft(). В результает получем комплексный вектор, который содержит информацию как об амплитуде спектральных составляющих сигнала, так и о фазе.

```
from fft_signal import fft_signal

fft_s, nsamp, fs, df, freq_vec = fft_signal(s, t);
```

```
Частота дискретизации = 25  МГц
Разрешение по частоте = 381  Гц
```

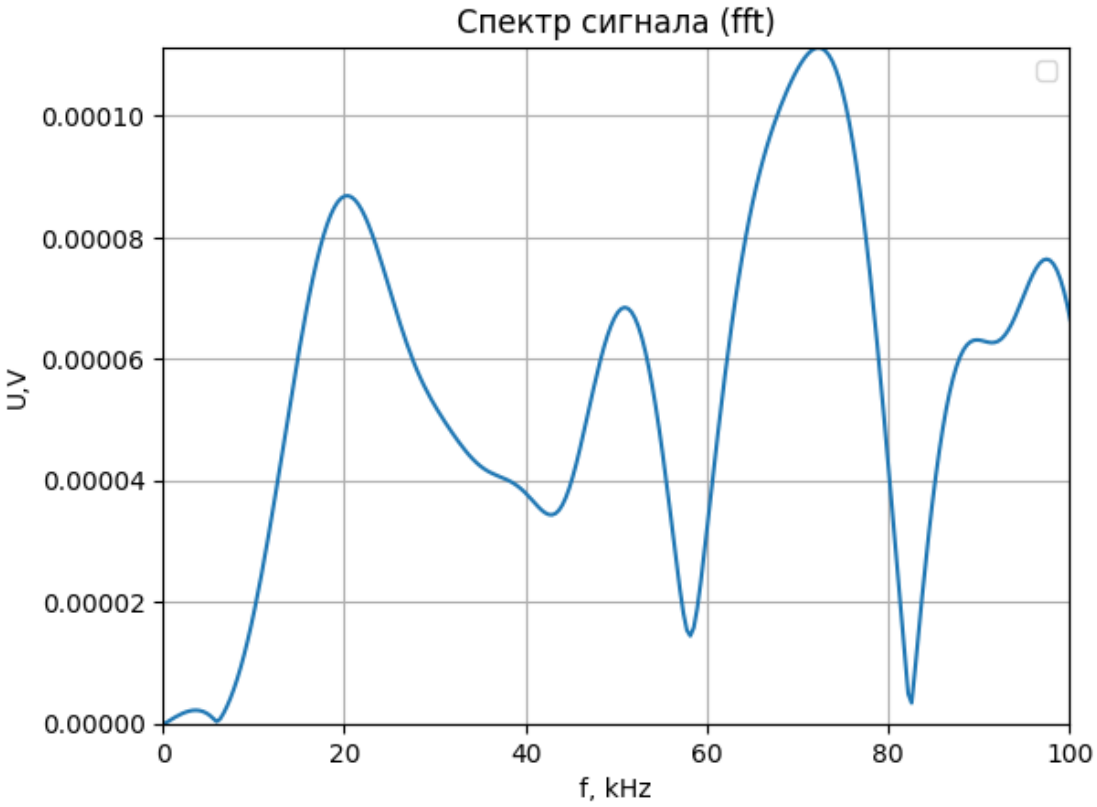
строи спектр сигнала (fft)

```
plt.figure()
plt.plot(freq_vec / 1e6, np.abs(fft_s))
plt.xlabel("f, MHz")
plt.ylabel("U,V")
plt.title("Спектр сигнала (fft)")
plt.xlim(0, 1)
plt.grid(True)
plt.show()
```

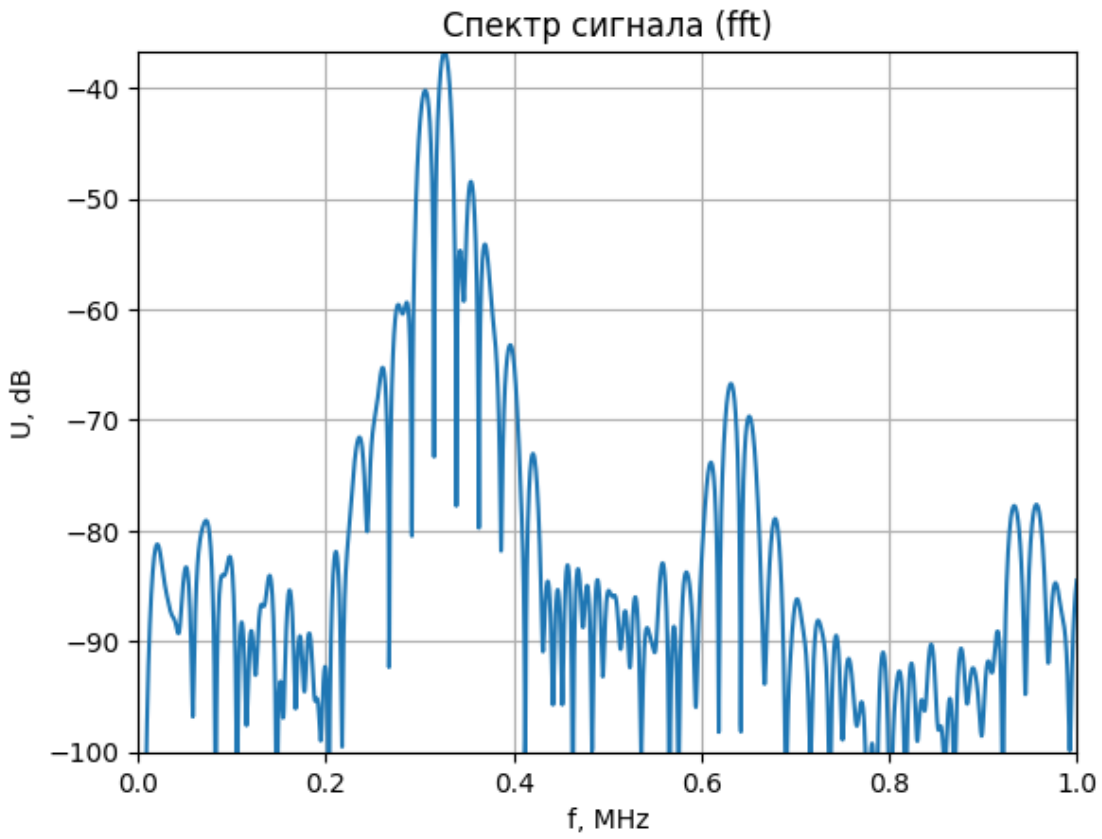



```
inds = np.where((freq_vec >= 0) & (freq_vec <= 100_000))[0]
max_y = np.max(np.abs(fft_s[inds]))

plt.figure()
plt.plot(freq_vec / 1e3, np.abs(fft_s))
plt.xlabel("f, kHz")
plt.ylabel("U,V")
plt.title("Спектр сигнала (fft)")
plt.legend([])
```

```
plt.xlim(0, 100)
plt.ylim(0, max_y)
plt.grid(True)
plt.show()
```




```
inds = np.where((freq_vec >= 0) & (freq_vec <= 1e6))[0]
max_y = np.max(10 * np.log10(np.abs(fft_s[inds]) ** 2))

plt.figure()
plt.plot(freq_vec / 1e6, 10 * np.log10(np.abs(fft_s) ** 2))
plt.xlabel("f, MHz")
plt.ylabel("U, dB")
plt.title("Спектр сигнала (fft)")
plt.xlim(0, 1)
plt.ylim(-100, max_y)
plt.grid(True)
plt.show()
```



3.2 Периодограмма

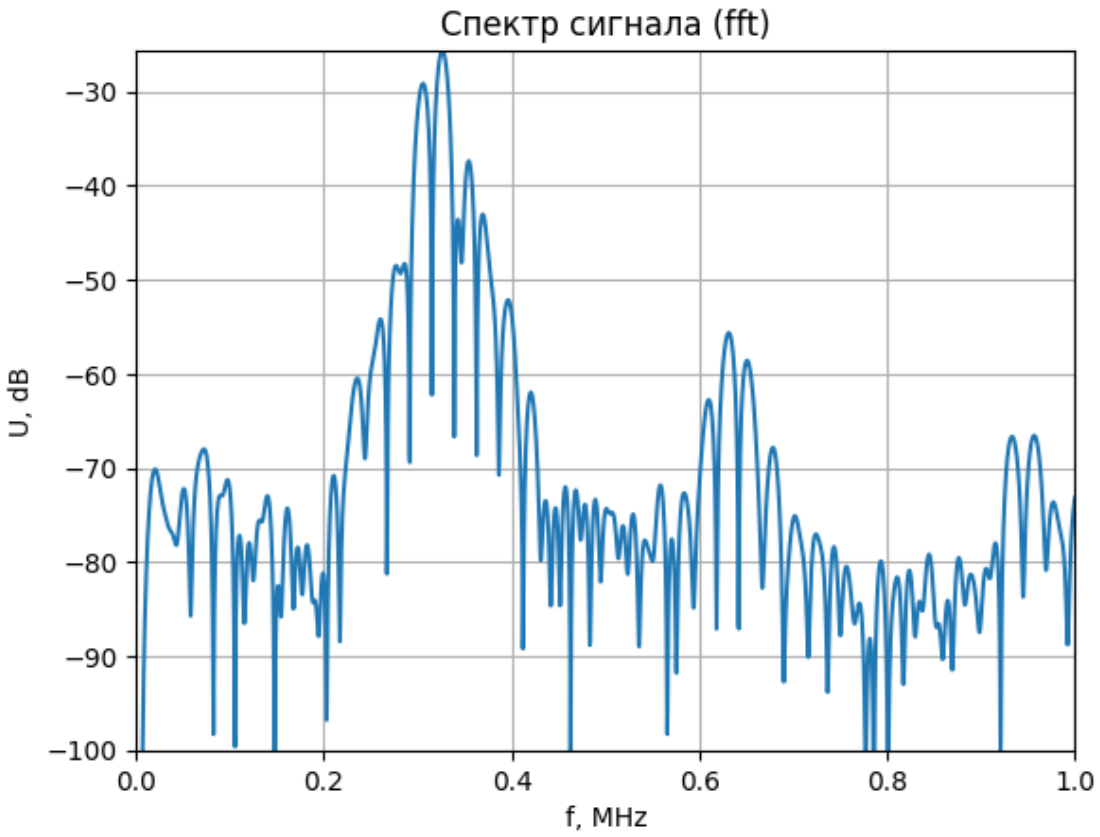
Периодограмма — это метод оценки **спектральной плотности мощности (СПМ)** сигнала на основе его **дискретного преобразования Фурье (ДПФ)**. На практике вычисляется через **БПФ**.

Алгоритм:

1. Берётся отрезок сигнала (например, 1024 отсчёта).
2. Вычисляется БПФ этого отрезка.
3. Амплитудный спектр возводится в квадрат и нормируется на длину выборки.
4. Получается график мощности по частотам.

```
# !pip install scipy

from scipy.signal import periodogram
```

```
f_periodogram, P_periodogram = periodogram(s_arr, fs=fs, nfft=nsamp,
  ↪scaling='density', return_onesided=True)
max_y = np.max(10 * np.log10(N * P_periodogram))

plt.figure()
plt.plot(f_periodogram / 1e6, 10 * np.log10(N * P_periodogram))
plt.xlabel("f, MHz")
plt.ylabel("U, dB")
plt.title("Спектр сигнала (fft)")
plt.xlim(0, 1)
plt.ylim(-100, max_y)
plt.grid(True)
plt.show()
```



3.3 Метод Уэлча

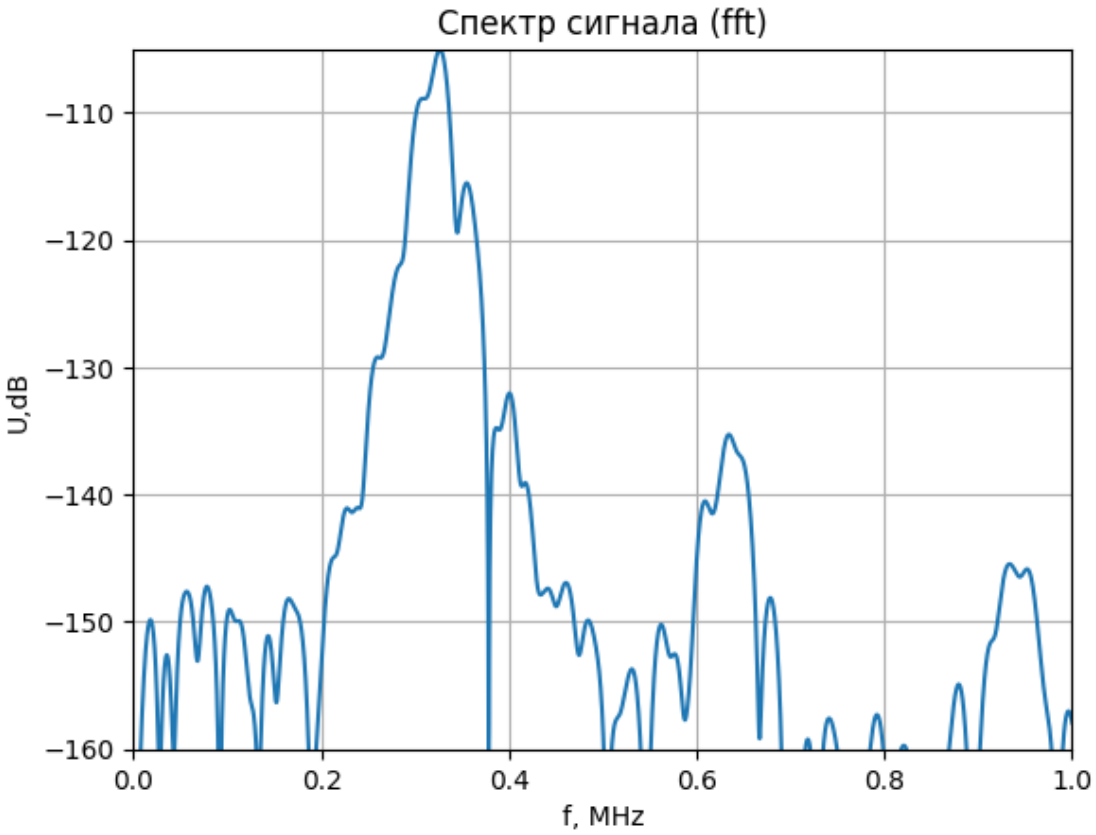
Метод Уэлча - это модификация периодограммы, в которой:

- Сигнал разбивается на перекрывающиеся отрезки.
- К каждому отрезку применяется оконная функция (например, Ханна, Хэм-

минга) - это уменьшает эффект утечки спектра.

- Для каждого отрезка считается периодограмма.
- Результаты усредняются → снижается дисперсия.

```
from scipy.signal import welch

f_welch, Pxx_welch = welch(np.array(s), fs=fs, nperseg=nsamp,
  ↪scaling='density', return_onesided=True)
max_y = np.max(10 * np.log10(Pxx_welch))

plt.figure()
plt.plot(f_welch / 1e6, 10 * np.log10(Pxx_welch))
plt.xlabel("f, MHz")
plt.ylabel("U,dB")
plt.title("Спектр сигнала (fft)")
plt.xlim(0, 1)
plt.ylim(-160, max_y)
plt.grid(True)
plt.show()
```

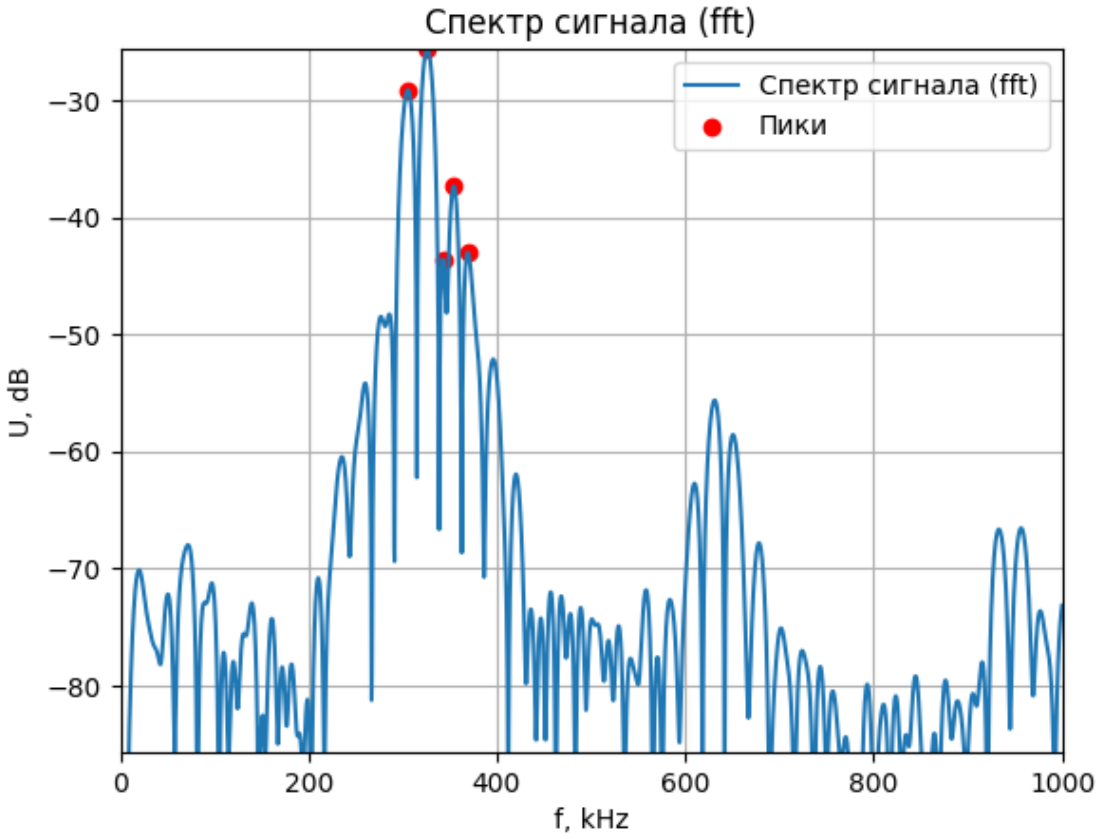


4 Определение основной частоты сигнала

Остановимся на оценке спектра методом Уэлча и проанализируем область низких частот (от 0 до 1500 Гц). Найдём **пики** в спектре - они соответствуют гармоникам сигнала. В этом нам поможет функция findpeaks1d из соответствующего пакета. Выделим первые шесть гармоник, и отобразим их на графике:

```
from plot_peaks_periodogram import plot_peaks_periodogram

p_test = P_periodogram
f_test = f_periodogram

plot_peaks_periodogram(N*p_test, f_test, freq_limit=1_000_000,
  ↪height=-45)
```

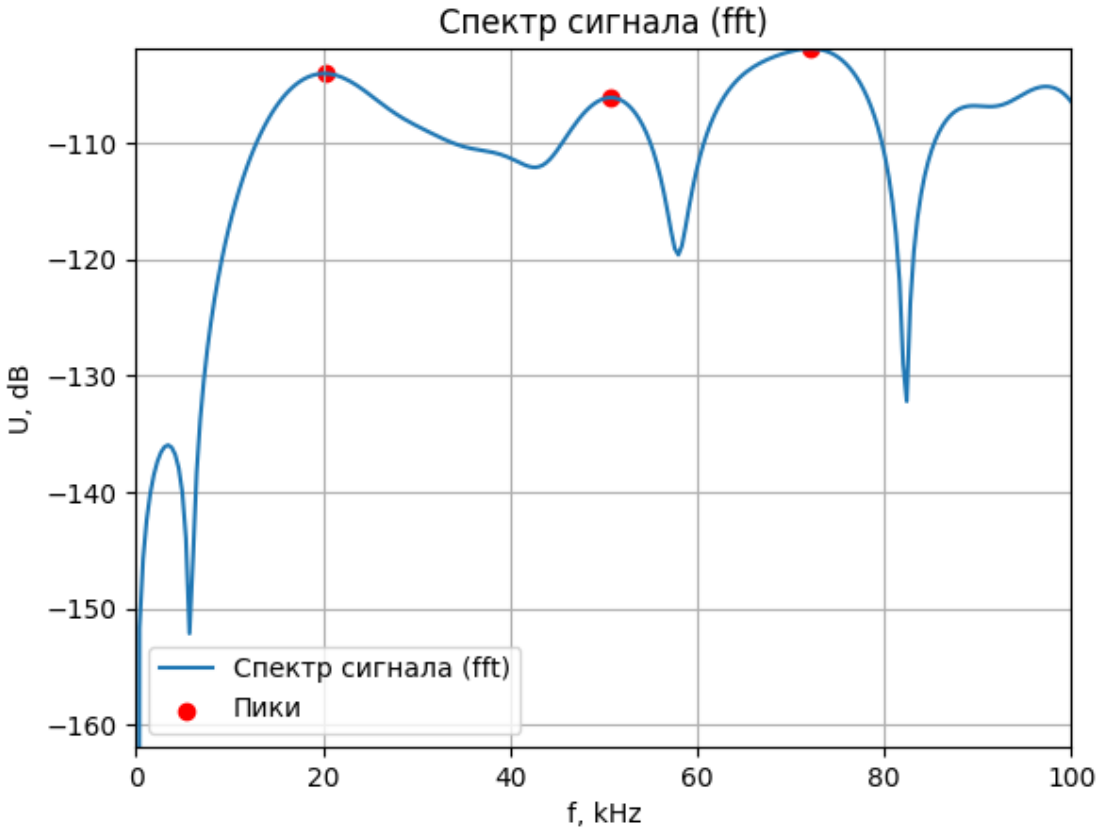
```
Индексы пиков: [801 855 899 929 968]
Частоты пиков: [305.55725098 326.15661621 342.94128418 354.38537598
369.26269531] кГц
Амплитуды пиков: [-29.13693325 -25.64784317 -43.60254371 -37.36996384
-43.03219691] дБ
```



Рассмотрим более внимательно спектр сигнала в рабочей области ППФ (в низких частотах)

```
plot_peaks_periodogram(p_test, f_test, freq_limit=100_000, height=-120)
```

```
Индексы пиков: [ 53 133 189]
Частоты пиков: [20.21789551 50.73547363 72.09777832] кГц
Амплитуды пиков: [-104.03028153 -106.09528253 -101.88315321] дБ
```



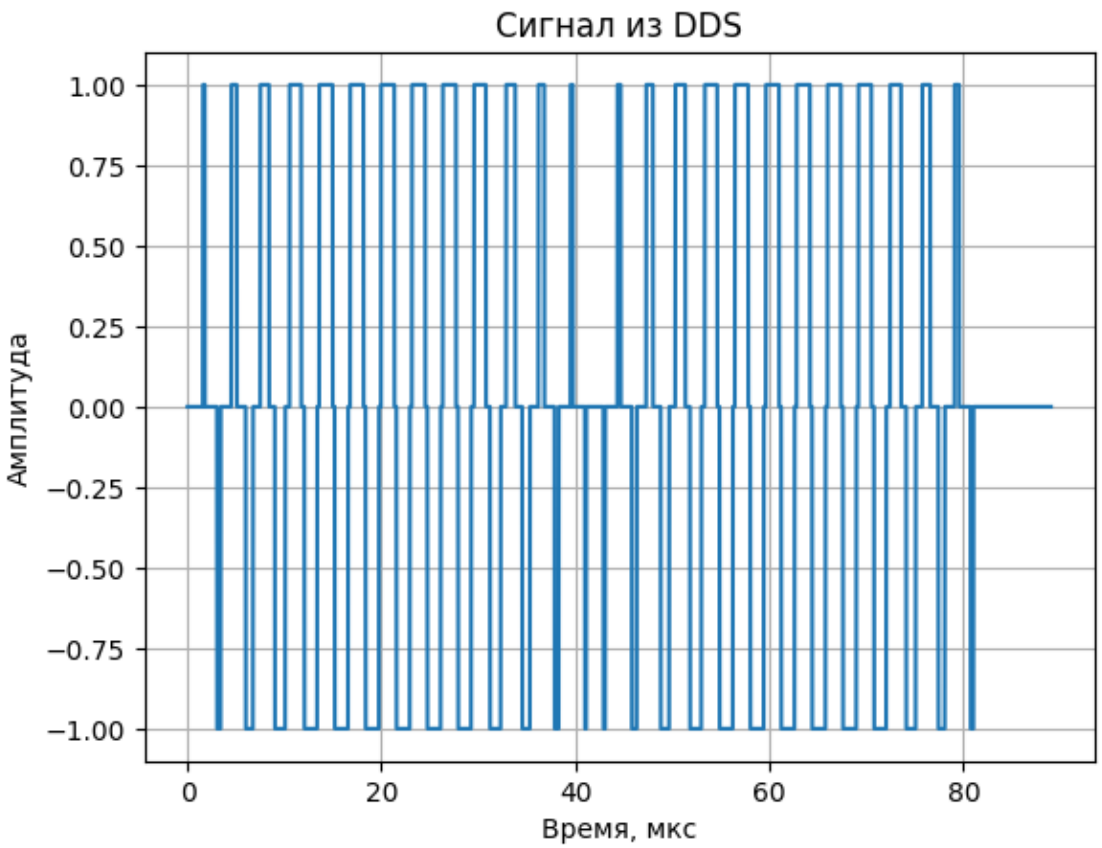
5 Считывание параметров ЗИ из файла *.dds

Параметры ЗИ записаны в файле *.dds. Формат файла описан руководстве системного программиста Гидра_DDS.14.pdf (стр.61):

```
from reader_dds import reader_dds, print_info, plot_signal

fclk = 125e6  # частота тактирования ЦАП

mode = reader_dds(dds_file, fclk)
```

```
print_info(mode)
plot_signal(mode)
```

```
sig1: 0x55aa0f0f
sig2: 0xff00caac
version: 0x100
rec_type: 0x201
size_file: 366
Code_ZI: 1392
Mode: 5
Mode1: 1
TimeZ: 26
NumPrd: 0
TimeR: 77
TimeI: 38
CRC: 0
Prm0: 0
Prm1: 0
Ver: 0
Type: 0
Name_RU: 2F_315кГц_81мкс
Name_EN: 2F_315kHz_81us
StartFreq: 24000.0
EndFreq: 24000.0
TimeZI: 8.104000000000001e-05
Fs: 78125.0
SampleType: 0
Rzv2: 0
NumCC: 103
Rzv3: 0
```



5.0.1 Фильтрация цифрового сигнала

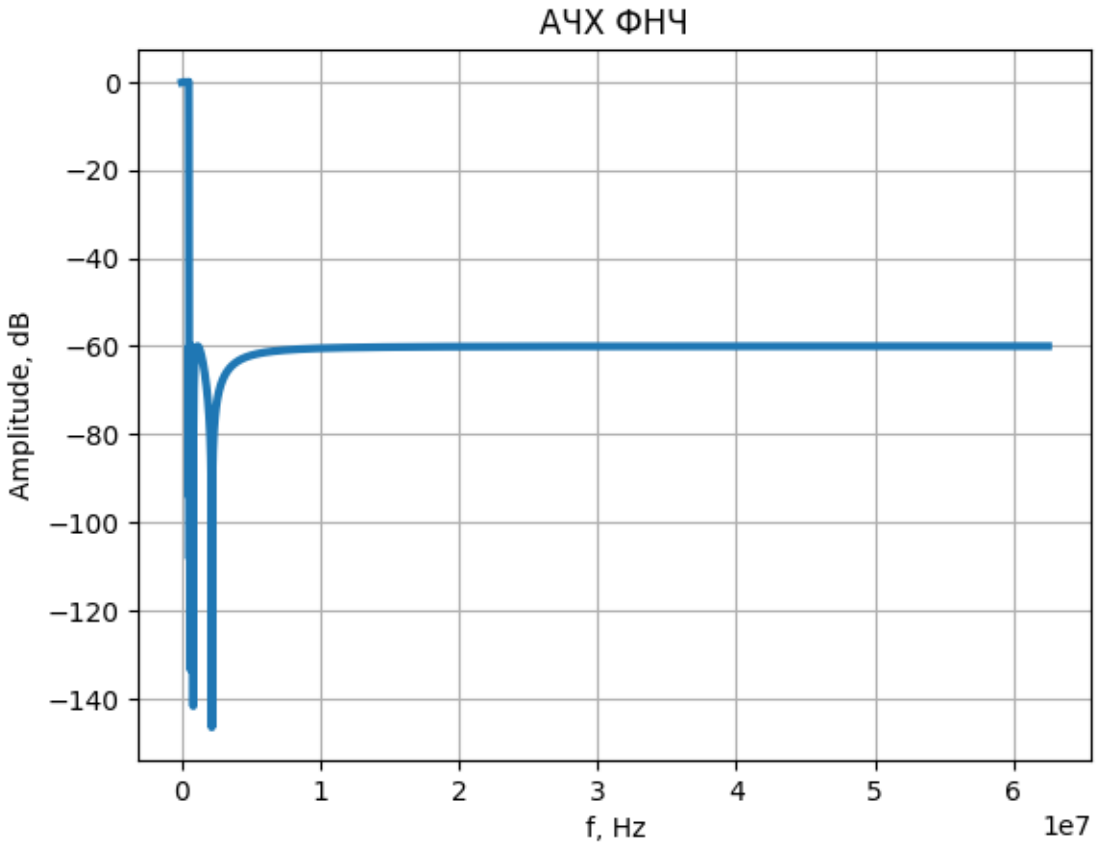
Синтезируем простой эллиптический фильтр нижних частот (ФНЧ), подавляющий частоты выше 360 Гц. Это позволит нам отсечь высшие гармоники. Отфильтруем исходный сигнал и прослушаем результат:

```
from scipy.signal import ellip, sosfilt, sosfreqz

f_cutoff = 500_000

# Создание эллиптического фильтра нижних частот
sos = ellip(N=10,               # порядок фильтра
            rp=0.1,             # неравномерность в полосе пропускания
 ↪(дБ)
            rs=60,              # подавление в полосе задерживания (дБ)
            Wn=f_cutoff / (fclk / 2),  # нормированная частота среза
            btype='low',
            output='sos')
```

```
# # Пример применения фильтра к сигналу
# s_after_flt = sosfilt(sos, s_arr)
```

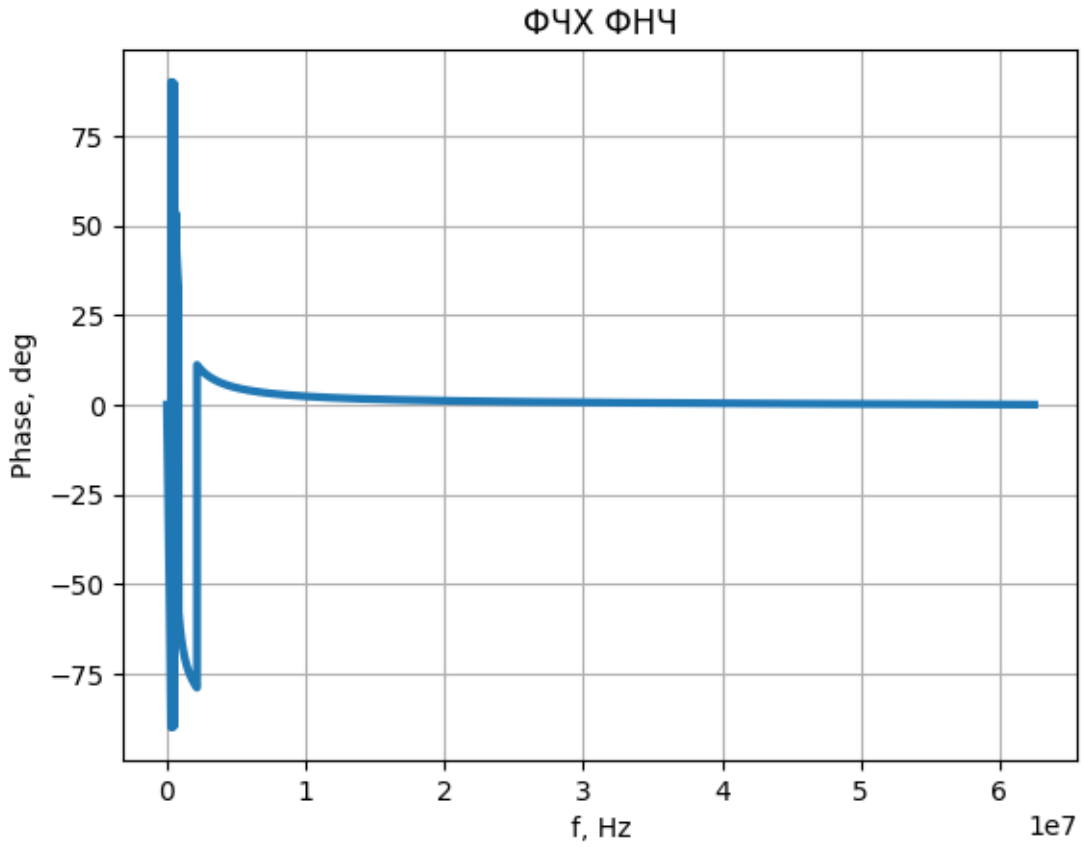
Полезно оценить результат синтеза коэффициентов, визуализировав амплитудно-частотную (АЧХ) и фазо-частотную (ФЧХ) характеристики фильтра, а также его импульсную характеристику:

```
from scipy.signal import sosfreqz

w, h = sosfreqz(sos, worN=f_cutoff, fs=fclk)
plt.figure()
plt.plot(w, 20 * np.log10(np.abs(h)), linewidth=3)
plt.title("АЧХ ФНЧ")
plt.xlabel("f, Hz")
plt.ylabel("Amplitude, dB")
plt.grid(True)
plt.show()
```




```
from scipy.signal import sosfreqz
```

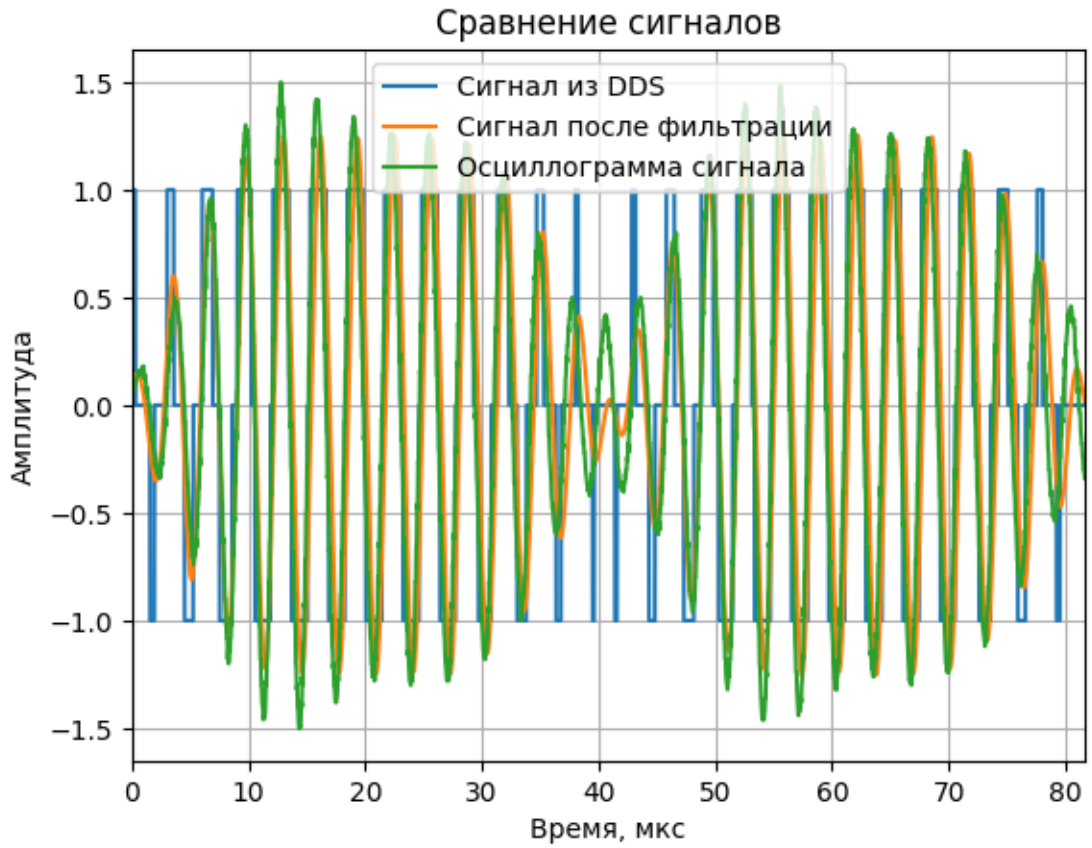
```
w, h = sosfreqz(sos, worN=f_cutoff, fs=fclk)
phi = np.angle(h)

plt.figure()
plt.plot(w, np.rad2deg(phi / 2), linewidth=3)
plt.title("ФЧХ ФНЧ")
plt.xlabel("f, Hz")
plt.ylabel("Phase, deg")
plt.grid(True)
plt.show()
```



Отфильтруем сигнал функцией filt, послушаем результат:

```
from scipy.signal import sosfilt

s_after_flt = sosfilt(sos, mode.s)

plt.figure()
plt.plot(np.array(mode.t) * 1e6 - 1.5, mode.s, label="Сигнал из DDS")
```

```
plt.plot(np.array(mode.t) * 1e6 - 3, s_after_flt, label="Сигнал после
  ↪фильтрации")
plt.plot(np.array(t) * 1e6, s, label="Осциллограмма сигнала")
plt.xlabel("Время, мкс")
plt.ylabel("Амплитуда")
plt.title("Сравнение сигналов")
plt.legend()
plt.grid(True)
plt.xlim(0, np.array(t)[-1] * 1e6 / oversampling_factor / 1.2)
plt.show()
```



Сравним спектры исходного и отфильтрованного сигнала:

```
# # Welch periodogram for both signals
# f_welch_mode, Pxx_welch_mode = welch(mode.s, fs=fclk,
  ↪nperseg=60000, scaling='density', return_onesided=True)
# f_welch_flt, Pxx_welch_flt = welch(s_after_flt, fs=fclk,
  ↪nperseg=60000, scaling='density', return_onesided=True)

# Standard periodogram for both signals
```

```
f_pgram_mode, P_mode = periodogram(mode.s, fs=fclk, nfft=65000,
  ↪scaling='density', return_onesided=True)
f_pgram_flt, P_flt = periodogram(s_after_flt, fs=fclk, nfft=65000,
  ↪scaling='density', return_onesided=True)
f_osc, P_osc = periodogram(s_arr, fs=fs, nfft=nsamp,
  ↪scaling='density', return_onesided=True)

plt.figure()
plt.plot(f_pgram_mode / 1e6, 10 * np.log10(P_mode), label="Исходный
  ↪сигнал")
plt.plot(f_pgram_flt / 1e6, 10 * np.log10(P_flt), linewidth=3,
  ↪label="После фильтрации")
plt.plot(f_osc / 1e6, 10 * np.log10(P_osc) + 14, linewidth=3,
  ↪label="Реальный сигнал")
plt.xlabel("f, MHz")
plt.ylabel("U, dB")
plt.title("Спектр сигнала (fft)")
plt.xlim(0, 1)
plt.ylim(-100, -40)
plt.grid(True)
plt.legend()
plt.show()
```