Индивидуальное задание 2
“Дискретизация сигнала и его спектральный анализ”
Вариант 7

Илья Мурадьян, группа 4.1

4 апреля 2018 г.

Мой вариант предполагал работу с сигналом $x_0, x_1, ..., x_{N-1}, N = 512$ с частотой дискретизации $h \approx 0.018104$ из файла `Sa7.tx`. Ввиду того, что файл содержит 513 строк, я его здесь не привожу.

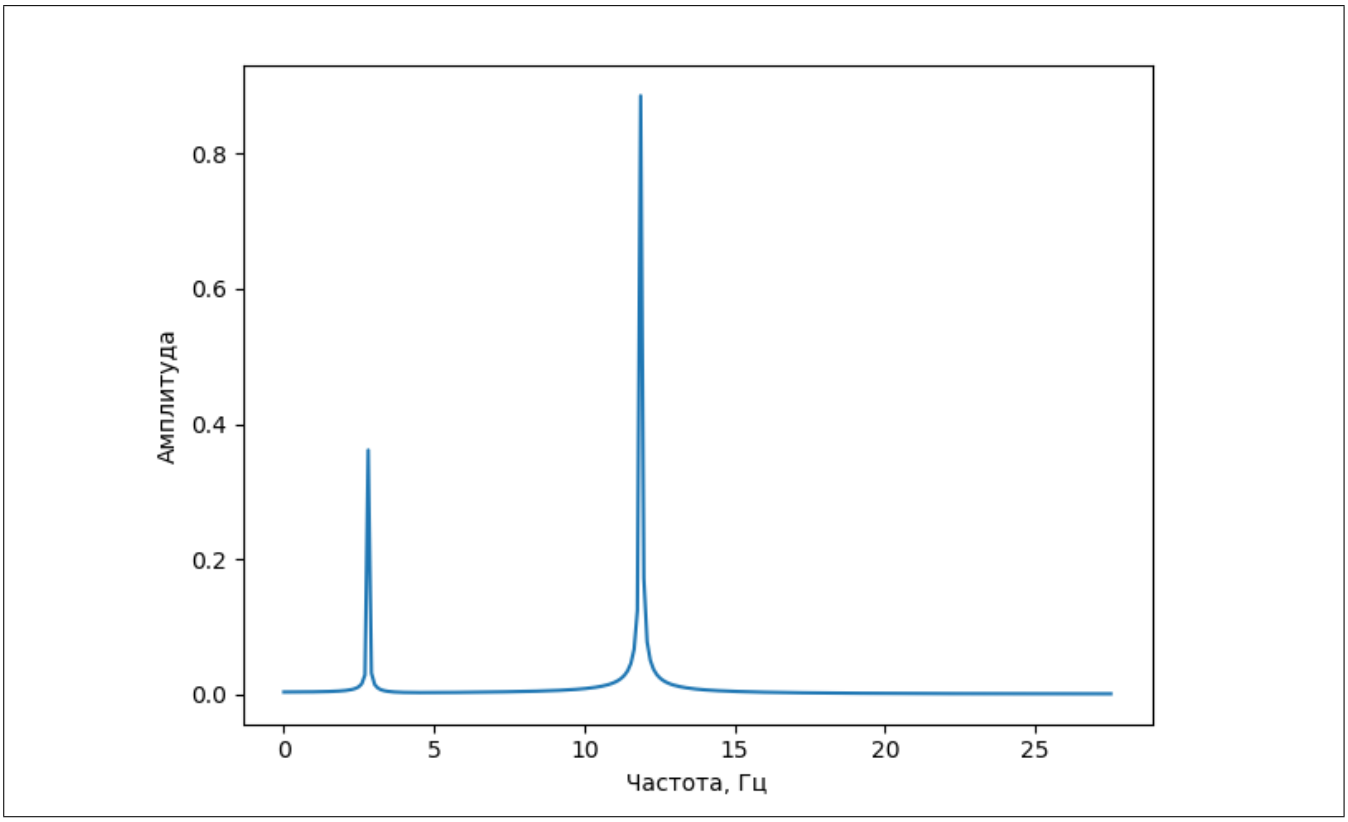

Рис. 1: БПФ исходного сигнала

Было проведено преобразование Фурье $X_0, X_1, ..., X_{N-1}$ входного сигнала, график функции $A_i = \frac{2|X_i|}{N}, i = 0, 1, ..., \frac{N}{2} - 1$ приведён на рисунке 1. Нормировка вещественного сигнала понадобилась для получения реальных амплитуд. Частоты в герцах вычислялись через индексы следующим образом:

$$\nu_i = \frac{i}{Nh} \tag{1}$$

Для выделения гармоник была написана функция `extract_harmonics` (см. листинг 1). Она выделяет амплитуды выше некоторого предела, устанавливаемого вручную (я всюду брал $threshold = 0, 2$), а затем в выделенных кусках находит максимумы и соответствующие им индексы.

```
def extract_harmonics(frequencies, amplitudes, threshold = None):
        th = threshold if threshold is not None else CONFIG['threshold']
        labels, num_labels = ndimage.label(amplitudes > th)
        unique_labels = np.unique(labels)
        idx = np.array(ndimage.maximum_position(amplitudes, labels, unique_labels[1:]))
            .reshape((num_labels, ))
        return frequencies[idx], amplitudes[idx]
```

Листинг 1: Функция extract_harmonics

По найденным частотам в герцах круговые частоты восстанавливались по формуле:

$$\omega_i = 2\pi\nu_i. \tag{2}$$

Это преобразование и печать данных о выделенных гармониках осуществляет функция `print_components`, приведённая в листинге 2.

```
def print_components(title, frequencies, amplitudes):
        break_out = '='*60
        c_f = 2 * np.pi * frequencies
        print('\n{}:'.format(title))
        print(break_out)
        print('Circular freq\t\t\tFreq\t\t\t\tA')
        for i in range(frequencies.shape[0]):
                print('{}\t\t{}\t\t{}'.format(c_f[i], frequencies[i], amplitudes[i]))
                print(break_out)
                print('')
```

Листинг 2: Функция print_components

В листинге 3 приведены ещё две вспомогательные функции, осуществляющие чтение из файла, сохранение графиков в файл и их вывод на экран.

```
def get_data():
    file_name = CONFIG['file_name']
    with open(file_name, 'r') as f:
        first_line = f.readline()
        h = float(first_line.split(':')[1])
        s_val = f.readlines()
        data = np.ndarray((len(s_val),), dtype=np.float64)
        for i, val in enumerate(s_val):
            data[i] = float(val)
        return h, data

def show_save(x, y, title:str, x_label:str='', y_label:str=''):
    plt.close()
    plt.plot(x, y)
    # plt.title(title)
    plt.tight_layout(2)
    plt.xlabel(x_label)
    plt.ylabel(y_label)
    plt.savefig(title+'.png')
    #plt.show()
```

Листинг 3: Функции ввода-вывода

Наконец, в листинге 4 все указанные функции были использованы для выделения гармоник из исходного сигнала.

```
h, data = get_data()
n = data.shape[0]
n_2 = n // 2
```

```
t_all = n * h

freq_line = np.arange(n) / t_all
freq_line_2 = freq_line[:n_2]
data_t = np.fft.fft(data, n)
data_t_amplitudes = (np.abs(data_t) / n * 2)[:n_2]

show_save(freq_line_2, data_t_amplitudes, 'Figure1', 'Frequency, Hz', 'Amplitude')
harmonics_1 = extract_harmonics(freq_line_2, data_t_amplitudes)
print_components('Clear signal components', *harmonics_1)
```

Листинг 4: Обработка исходного сигнала

В таблице 1 приведены амплитуды и частоты выделенных гармоник.

№	Круговая частота	Частота, Гц	Амплитуда
1	17.6239196013666	2.8049339212116404	0.36153886289036397
2	74.56273677501255	11.867028128203094	0.8856231380539284

Таблица 1: Чистый сигнал

Определение 1. Шумом амплитуды A будем называть сигнал x, для которого $x_i = A \cdot rand(-1, 1)$. $rand(a, b)$ здесь означает случайную величину, равномерно распределенную на отрезке $[a, b]$.

Поскольку шум носит случайный характер, указать на точную границу амплитуды, при которой из сигнала ещё можно выделить исходные гармонические составляющие, сложно. Но указать примерные границы можно. Получилось, что при амплитуде $A_1 = 1,5$ исходные гармоники ещё выделяются (см. рисунок 2, таблицу 2), а при амплитуде $A_2 = 2$ – уже нет (см. рисунок 3, таблицу 3).

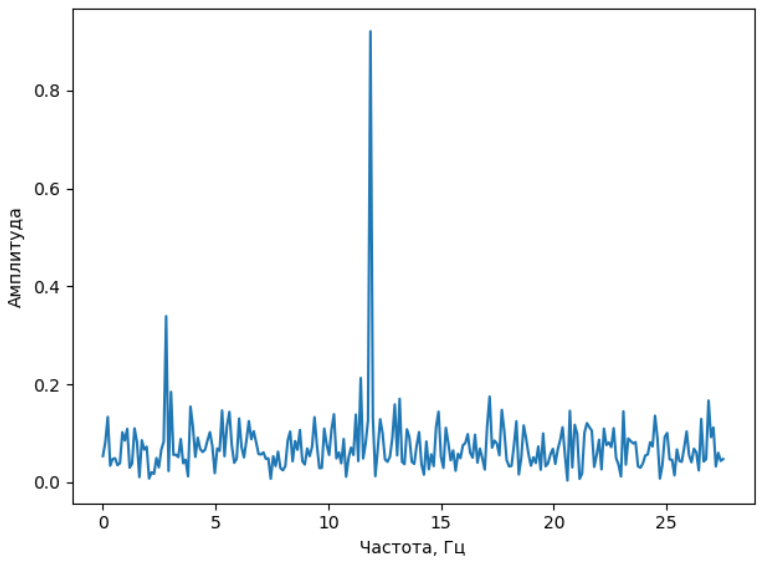

Рис. 2: Несильно зашумленный сигнал

№	Круговая частота	Частота, Гц	Амплитуда
1	17.6239196013666	2.8049339212116404	0.33924080902743353
2	71.85136452864846	11.435499832632074	0.21289770452023127
3	74.56273677501255	11.867028128203094	0.9206363487398033

Таблица 2: Гармоники несильно зашумленного сигнала

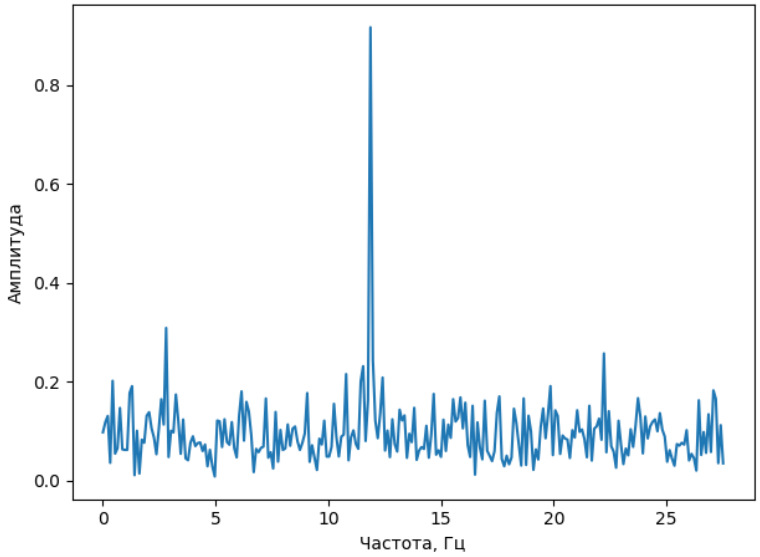

Рис. 3: Сильно зашумленный сигнал

№	Круговая частота	Частота, Гц	Амплитуда
1	2.7113722463640926	0.4315282955710216	0.20181606302237273
2	17.6239196013666	2.8049339212116404	0.30872862676602403
3	67.78430615910231	10.78820738927554	0.21540254430716102
4	72.52920759023948	11.543381906524829	0.23139725890845966
5	74.56273677501255	11.867028128203094	0.9165057088333591
6	77.95195208296767	12.406438497666871	0.2084104876153394
7	139.63567068775077	22.223707221907613	0.2571670581225341

Таблица 3: Гармоники, выделенные программой в плохом сигнале

В листинге 5 приведена работа с зашумленными сигналами: генерация шума, сложение сигналов, вычисление их БПФ и выделение гармоник.

```
noise_configs = [
        (1.5, 'Figure2', 'OK noise'),
        (2, 'Figure3', 'Not-OK noise'),
]

for noise_a, figure_title, log_title in noise_configs:
        noise = np.random.rand(n) * (noise_a * 2) - noise_a

        noised_signal = data + noise
        noised_t = np.fft.fft(noised_signal, n)
        noised_t_amplitudes = (np.abs(noised_t) / n * 2)[:n_2]

        show_save(freq_line_2, noised_t_amplitudes, figure_title, 'Frequency, Hz', '
            Amplitude')
        harmonics_2 = extract_harmonics(freq_line_2, noised_t_amplitudes)
        print_components(log_title, *harmonics_2)
```

Листинг 5: Работа с зашумленным сигналом

Прибавим теперь к нашему сигналу следующий сигнал:

$$\widehat{x}(t) = A\cos(\Omega_1 t) + 2A\cos(\Omega_2 t) \tag{3}$$

Амплитуду A выберем равной $0,4$, чтобы амплитуды составляющих сигнал $\widehat{x}$ гармоник оказались между найденными нами амплитудами гармоник исходного сигнала (которые составляют примерно $0,36$ и $0,89$). Шаг дискретизации оставим таким же.

Частота Найквиста для используемого нами шага дискретизации составляет:

$$\nu_N = \frac{1}{2h} \approx 27,61781. \tag{4}$$

Круговая частота Найквиста составляет, соответственно:

$$\omega_N = \frac{\pi}{h} \approx 173,5278. \tag{5}$$

Необходимо выбрать частоты сигнала $\widehat{x}$ так, чтобы выполнялось $\Omega_1 < \Omega_2 < \omega_N$, или $\frac{\Omega_1}{2\pi} < \frac{\Omega_2}{2\pi} < \nu_N$. Поэтому возьмём $\nu_1 = \frac{\Omega_1}{2\pi} = 25$, $\nu_2 = \frac{\Omega_2}{2\pi} = 27$. Для полученного после сложения с исходным сигнала проведём преобразование Фурье. График можно видеть на рисунке 4.

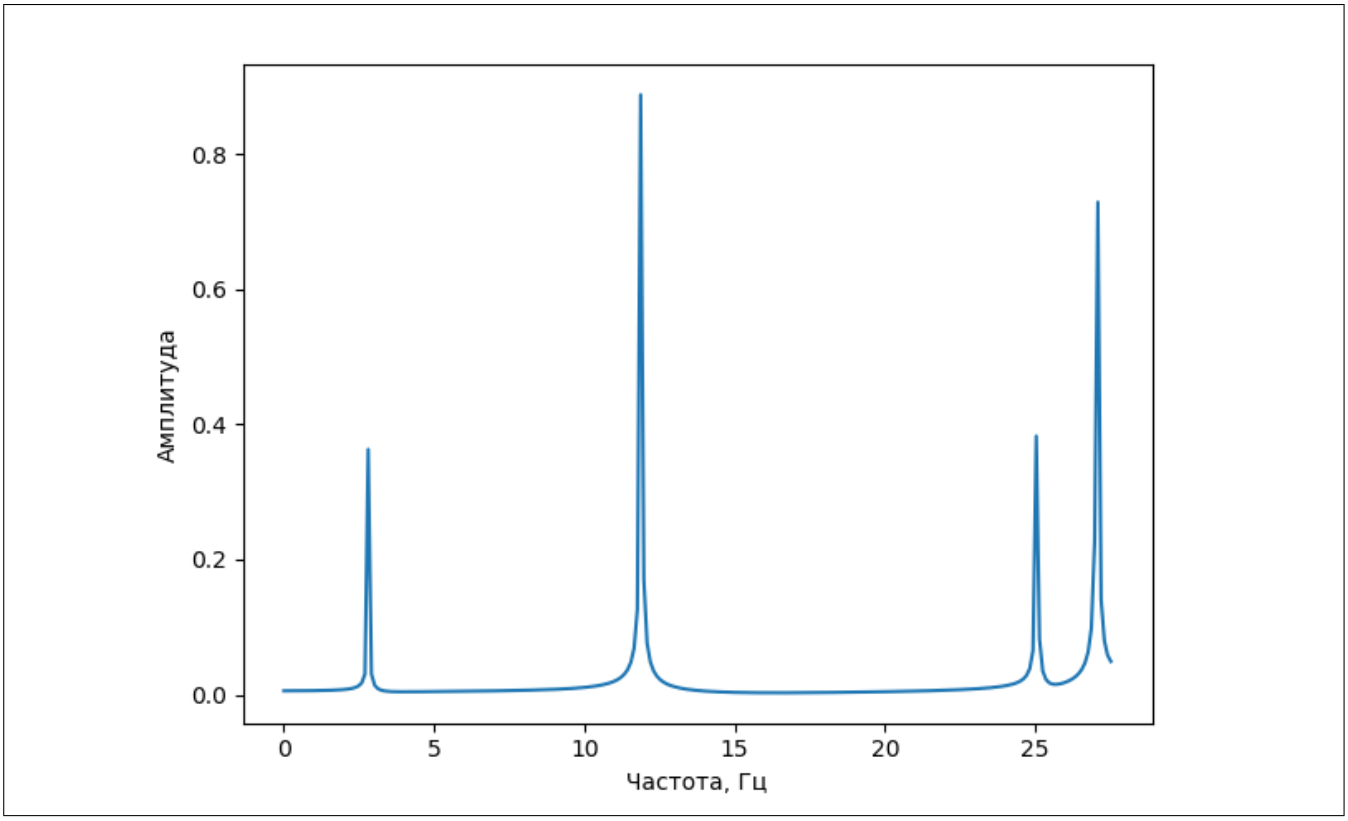

Рис. 4: Модифицированный сигнал

В таблице 4 можно видеть, что программа распознала все 4 частотных пика.

№	Круговая частота	Частота, Гц	Амплитуда
1	17.6239196013666	2.8049339212116404	0.3628595862359613
2	74.56273677501255	11.867028128203094	0.8875908052529861
3	157.25959028911737	25.028641143119252	0.3826367613808222
4	170.1386084593468	27.078400547081607	0.7286697992946649

Таблица 4: Гармоники модифицированного сигнала

Если $\Omega_2 > \omega_N$, возникает алиасинг. При дальнейшем увеличении Ω_2 пики добавляемых гармоник меняются местами. В теории это должно происходить, когда $\Omega_2 > \omega_N + (\omega_N - \Omega_1) = 2\omega_N - \Omega_1 \approx 189,976$. На практике приходится брать значения ещё больше, чтобы получилась хорошая картинка. Было взято значение $\Omega_2 = 33 \cdot 2\pi \approx 207,345$. Получились график 5 и таблица 5.

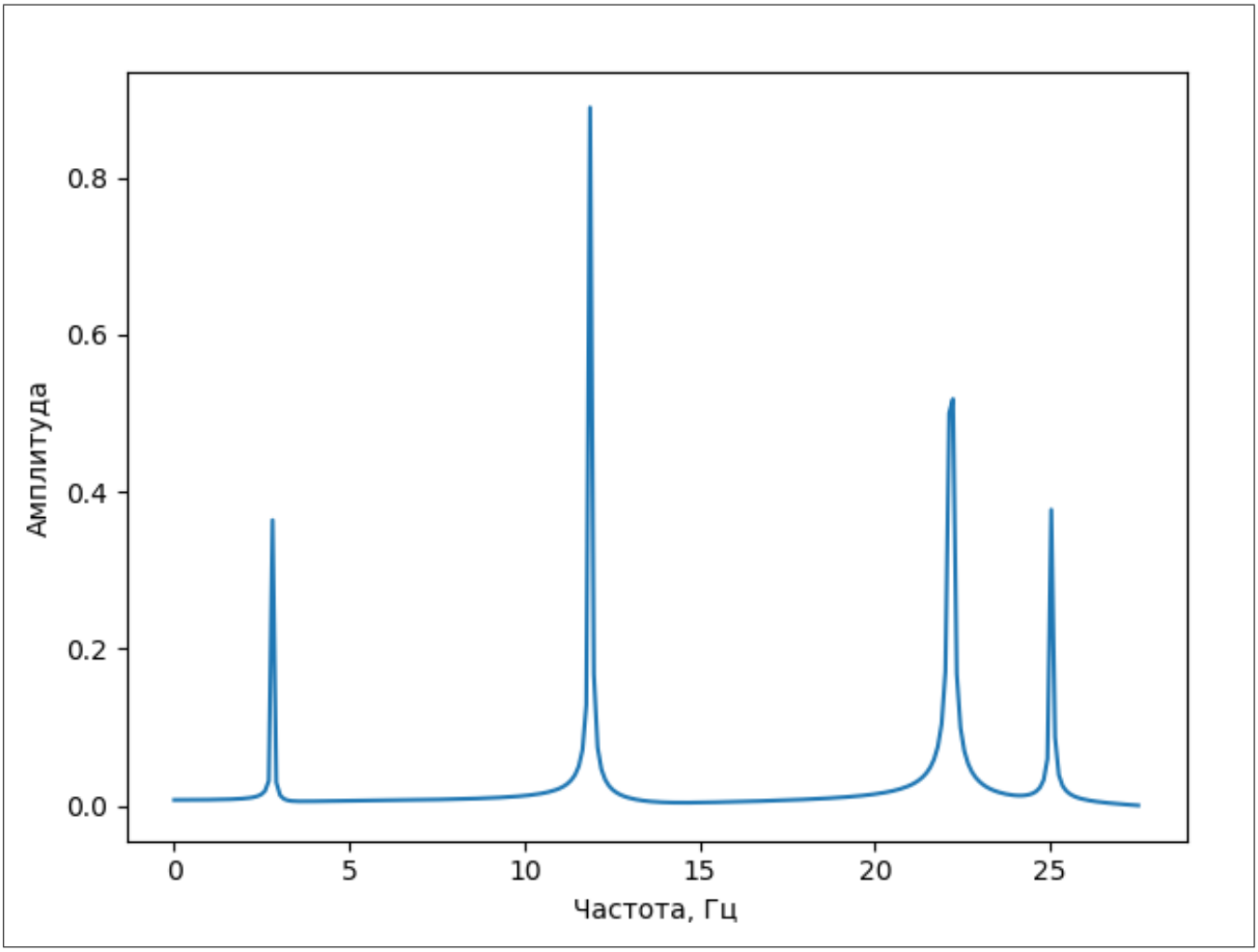

Рис. 5: Алиасинг

№	Круговая частота	Частота, Гц	Амплитуда
1	17.6239196013666	2.8049339212116404	0.3639636485096471
2	74.56273677501255	11.867028128203094	0.8897287103590086
3	139.63567068775077	22.223707221907613	0.5182384798759567
4	157.25959028911737	25.028641143119252	0.3772253427030885

Таблица 5: Гармоники при алиасинге

Если уменьшить шаг дискретизации до $h' = \frac{h}{2}$ и взять $\Omega_1 = 20 \cdot 2\pi, \Omega_2 = 25 \cdot 2\pi$ (то есть взять такие круговые частоты, с которыми при шаге дискретизации h алиасинга не возникало), то возникнет алиасинг. Соответствующие график и таблица приведены ниже.

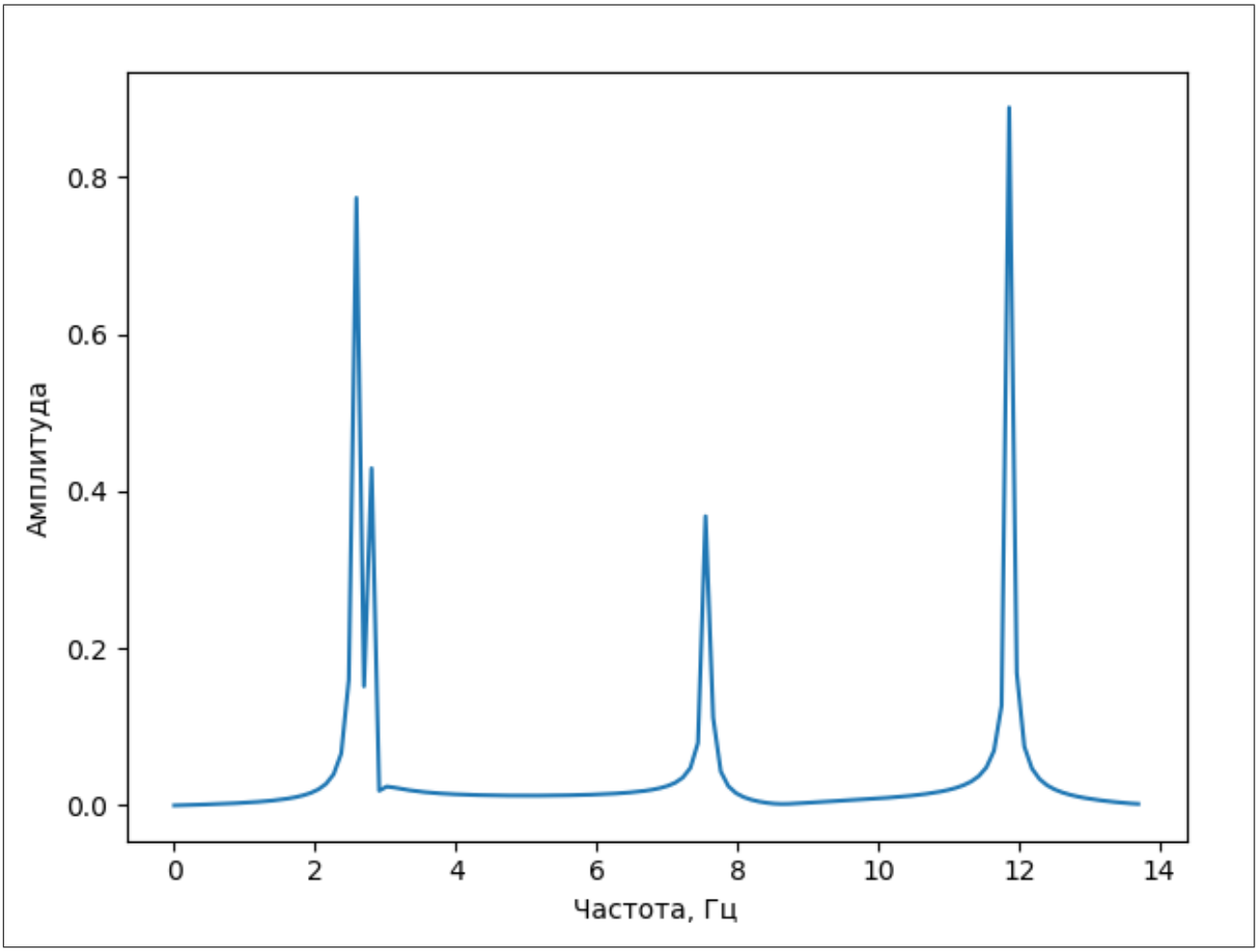

Рис. 6: Алиасинг при маленьком шаге дискретизации

№	Круговая частота	Частота, Гц	Амплитуда
1	16.26823347818456	2.58916977342613	0.7734754902584449
2	17.6239196013666	2.8049339212116404	0.4292633260699508
3	47.44901431137162	7.551745172492878	0.36820718500252175
4	74.56273677501255	11.867028128203094	0.8885122557004277

Таблица 6: Гармоники огрублённого сигнала, алиасинг

Остаток программы, который рисует три последние картинки/таблицы, приведён в листинге 6.

```
add_amplitude = CONFIG['add_amplitude']
add_configs = [
        (25, 27, 'Figure4', 'Modified signal (no aliasing)'),
        (25, 33, 'Figure5', 'Modified signal (aliasing + strange behavior)'),
]

for s1, s2, figure_title, log_title in add_configs:
        add_signal = modified_signal(time_line, s1, s2, add_amplitude)
        mod_sig = data + add_signal
        mod_t = np.fft.fft(mod_sig, n)
        mod_t_a = (np.abs(mod_t) / n * 2)[:n_2]

        show_save(freq_line_2, mod_t_a, figure_title, 'Frequency, Hz', 'Amplitude')
        harmonics_4 = extract_harmonics(freq_line_2, mod_t_a)
        print_components('csv'+figure_title, log_title, *harmonics_4)

add_signal = modified_signal(time_line, 20, 25)
```

```
mod_sig2 = add_signal[::2] + data[::2]
freq_line_4 = (np.arange(n//2) / (n * h))[:n//4]
mod_t2 = np.fft.fft(mod_sig2)
mod_t_a2 = (np.abs(mod_t2) / (n//2) * 2)[:n_2//2]

show_save(freq_line_4, mod_t_a2, 'Figure6', 'Frequency, Hz', 'Amplitude')
harmonics_6 = extract_harmonics(freq_line_4, mod_t_a2)
print_components('reduced','Modified signal components (only even samples)', *
    harmonics_6)
```

Листинг 6: Работа с модифицированным сигналом

Работа была выполнена мною лично, без чьей-либо помощи, без использования нелицензионных программ.