Федеральное агентство связи

СибГУТИ

Кафедра телекоммуникационных сетей и

вычислительных средств (ТС и ВС)

Дисциплина

Сети ЭВМ и телекоммуникации 2.0

Лабораторная работа №2

Цифровая обработка сигналов. Дискретизация.

Выполнил: студент группы ИА-832

Тиванов.Д.Е

Проверил: преподаватель

Ахпашев Р.В

Новосибирск 2021

Задание

1) Сформировать аккорд\мелодию длительностью до 5 секунд (в каждого уникальная);
2) Проверить влияние частоты дискретизации на качество воспроизводимой мелодии. Сделать выводы, описать;
3) Вывести каждый вариант (при разной дискретизации) в виде графика функции;
4) Изучить принцип и произвести преобразование Фурье (прямое и обратное) для исходного сигнала;
5) Сделать отчет с описанием и результатами и выводами.

Ход работы

Создание мелодии. Сигналы с нужными частотами хранятся в массиве SIGNALS. Представлены в виде формулы колебаний

$$s(t)=A\cdot sin(f\cdot t+\phi),$$

где: *A* - амплитуда синусоиды (в большинстве случаев это мощность сигнала),*f* - частота колебаний, ф = 0 – начальная фаза.

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

duration = 1  # длительность сигнала в секундах
amplitude = 0.3  # амплитуда  (в пределах: +-1.0)
frequency = 100  # частота сигнала в [Гц]
# т.к. невозможно программно организовать аналоговый сигнал, необходимо
обозначить
# количество временных отчетов, т.е. частоту дискретизации
fs = 80000  # 80 тыс. временных отсчетов в 1 секунду

timeSamples = np.arange(np.ceil(duration * fs)) / fs
timeSamples1 = np.arange(np.ceil(2 * fs)) / fs
print(len(np.arange(np.ceil(2 * fs))))
print(np.arange(np.ceil(2 * fs)))
SIGNALS = np.array(
    [
        amplitude * np.sin(2 * np.pi * 361.87 * timeSamples),
        amplitude * np.sin(2 * np.pi * 35.30 * timeSamples),
        amplitude * np.sin(2 * np.pi * 261.63 * timeSamples),
        amplitude * np.sin(2 * np.pi * 180.26 * timeSamples),
        amplitude * np.sin(2 * np.pi * 93.88 * timeSamples),
        amplitude * np.sin(2 * np.pi * 162.35 * timeSamples),
        amplitude * np.sin(2 * np.pi * 280.00 * timeSamples),
        amplitude * np.sin(2 * np.pi * 55.06 * timeSamples),
        amplitude * np.sin(2 * np.pi * 12.05 * timeSamples)
```

```
    ], float
                )

seconds = 3
s1 = amplitude * np.sin(2 * np.pi * 361.87 * timeSamples1)
s = []
for i in range(len(timeSamples)):
    s.append(amplitude * np.sin(2 * np.pi * 361.87 * timeSamples[i]))
for i in range(len(timeSamples), len(timeSamples)*2):
    s.append(amplitude * np.sin(2 * np.pi * 35.30 * timeSamples[i-
len(timeSamples)]))
for i in range(len(timeSamples)*2, len(timeSamples)*3):
    s.append(amplitude * np.sin(2 * np.pi * 261.63 * timeSamples[i-
len(timeSamples)*2]))
for i in range(len(timeSamples)*3, len(timeSamples)*4):
    s.append(amplitude * np.sin(2 * np.pi * 180.26 * timeSamples[i-
len(timeSamples)*3]))
for i in range(len(timeSamples)*4, len(timeSamples)*5):
    s.append(amplitude * np.sin(2 * np.pi * 93.88 * timeSamples[i-
len(timeSamples)*4]))
for i in range(len(timeSamples)*5, len(timeSamples)*6):
    s.append(amplitude * np.sin(2 * np.pi * 162.35 * timeSamples[i-
len(timeSamples)*5]))
for i in range(len(timeSamples)*6, len(timeSamples)*7):
    s.append(amplitude * np.sin(2 * np.pi * 280.00 * timeSamples[i-
len(timeSamples)*6]))
for i in range(len(timeSamples)*7, len(timeSamples)*8):
    s.append(amplitude * np.sin(2 * np.pi * 55.06 * timeSamples[i-
len(timeSamples)*7]))
for i in range(len(timeSamples)*8, len(timeSamples)*9):
    s.append(amplitude * np.sin(2 * np.pi * 12.05 * timeSamples[i-
len(timeSamples)*8]))

sd.play(s, fs)
time.sleep(9)
```

На данном этапе воспроизводится мелодия.

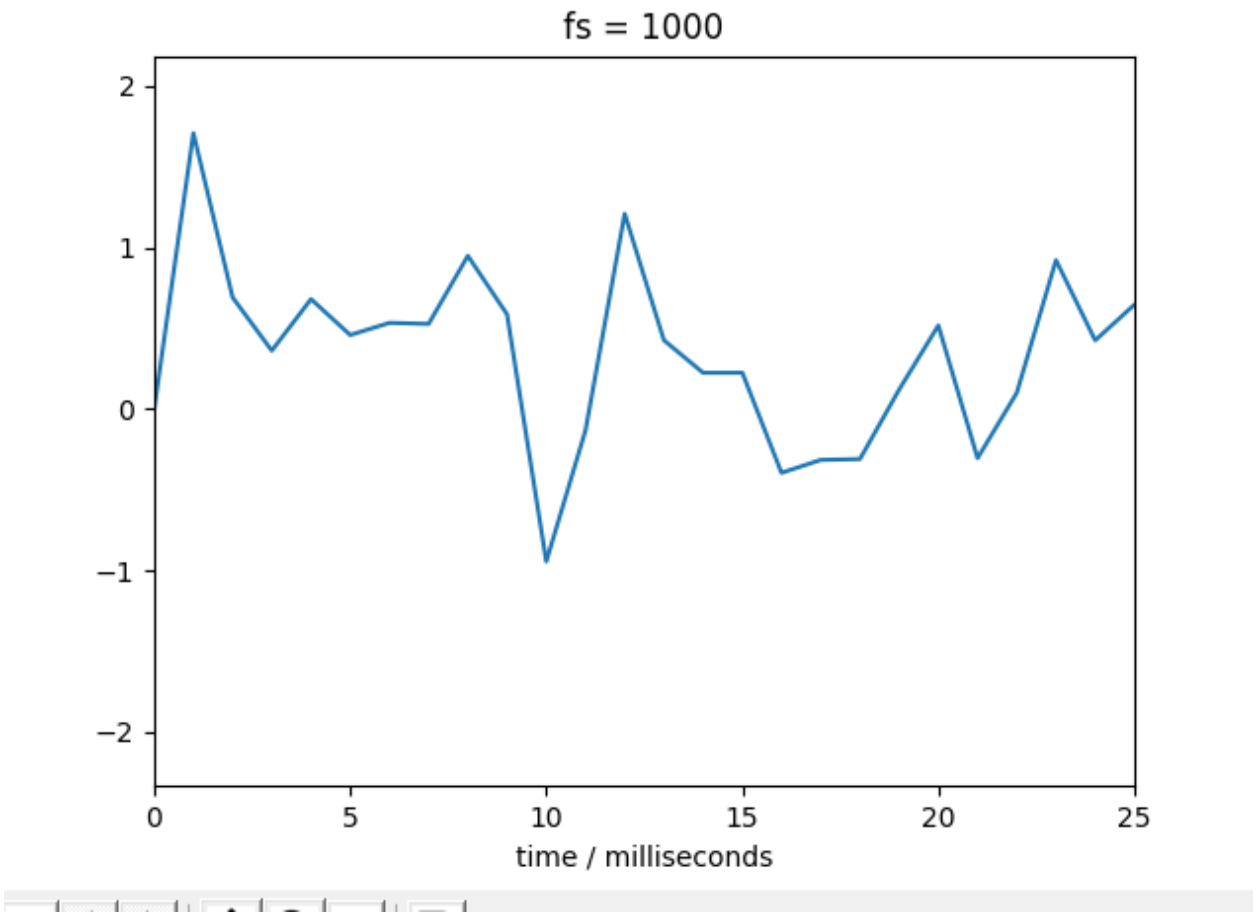
Частота дискретизации - частота, с которой происходит оцифровка, хранение, обработка или конвертация сигнала из аналога в цифру. Частота дискретизации, согласно Теореме Котельникова, ограничивает максимальную частоту оцифрованного сигнала до половины своей величины. Чем выше частота дискретизации, тем более качественной будет оцифровка. Как следует из теоремы Котельникова для того чтобы однозначно восстановить исходный сигнал, частота дискретизации должна превышать наибольшую необходимую частоту сигнала в два раза.

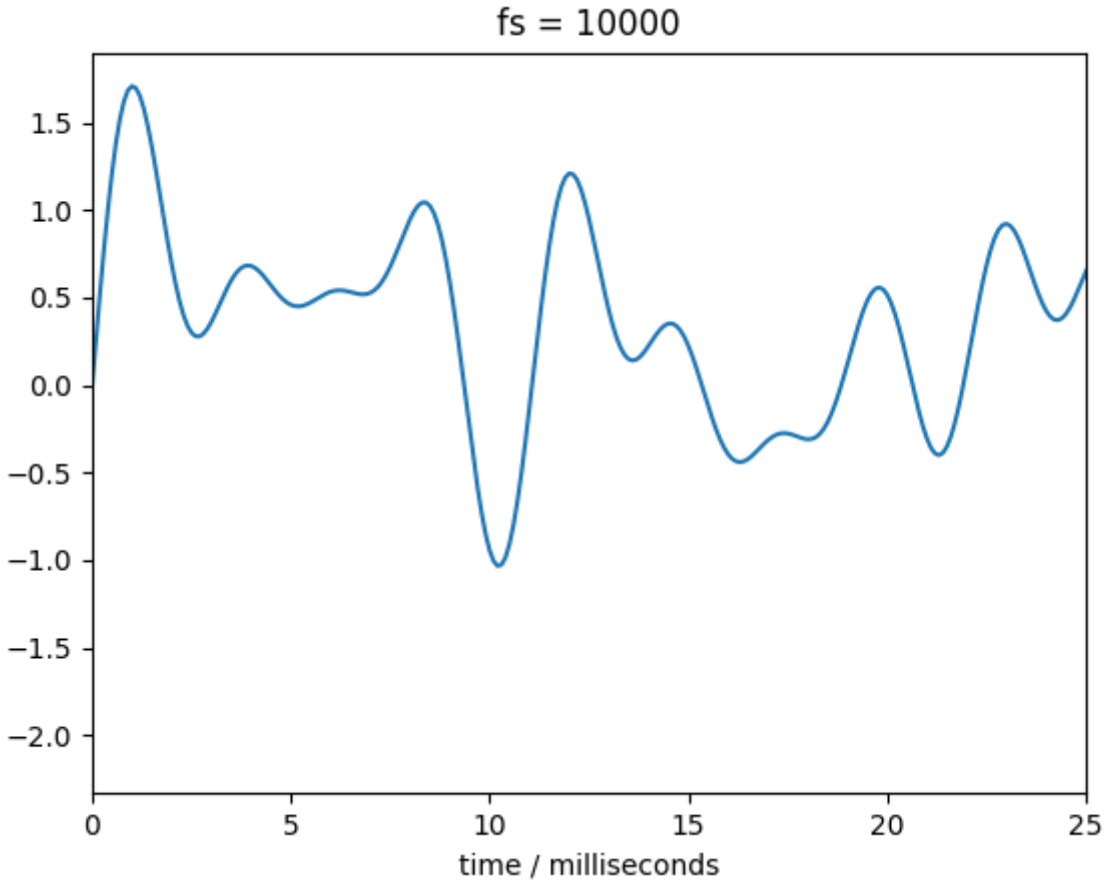
Исправив в коде частоту дискретизации fs на 1600, немного большую, чем в 2 раза от максимальной частоты, мелодия была воспроизведена без помех. Далее, увеличивая fs во много раз, заметного улучшения качества звука не последовало. Но, уменьшая значение от 1600 до минимального

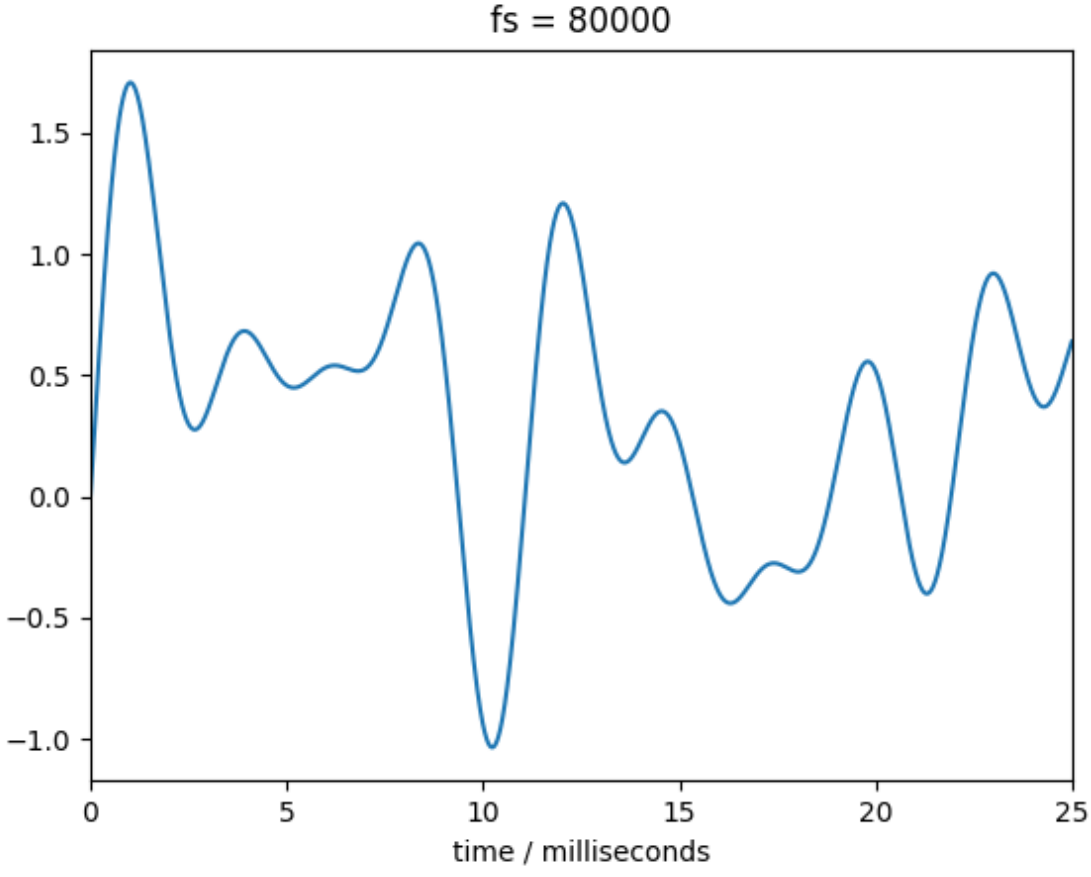
предела, качество записи ухудшалось прямо пропорционально уменьшению частоты дискретизации.

```
# попробуем сформировать несколько сигналов с разной частотой и отобразить в
виде суммы всех сигналов:
signalSum = 0
for i in range(len(SIGNALS)):
    signalSum = signalSum + SIGNALS[i]
#sd.play(SIGNALS, fs)


plt.plot(timeSamples[:2000] * 1000, signalSum[:2000])
plt.title("fs = " + str(fs))
plt.xlabel("time / milliseconds")
plt.xlim(0, 25)
plt.show()
```




fs = 10000
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
0
5
10
15
20
25
time / milliseconds


fs = 80000
1.5
1.0
0.5
0.0
−0.5
−1.0
0
5
10
15
20
25
time / milliseconds

Дискретное преобразование Фурье

Для использования в компьютерах, как для научных расчетов, так и для цифровой обработки сигналов, необходимо иметь функции x_k, которые определены на *дискретном* множестве точек вместо непрерывной области, снова периодическом или ограниченном. В этом случае используется дискретное преобразование Фурье (DFT), которое представляет x_k как сумму синусоид:

$$X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-i\,2\pi\,k\,n\,/\,N}$$

,

где f_j — амплитуды Фурье. Хотя непосредственное применение этой формулы требует $\mathrm{O}(n^2)$ операций, этот расчет может быть сделан за $\mathrm{O}(n \log n)$ операций используя алгоритм быстрого преобразования Фурье (БПФ, FFT) (см. О-большое), что делает преобразование Фурье практически важной операцией на компьютере.

Обратное дискретное преобразование Фурье (ОДПФ):

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k e^{i\,2\pi\,k\,n\,/\,N}$$

В библиотеке numpy содержится всё что нужно для дискретного преобразования Фурье. Всё это лежит в numpy.fft. Вот эти функции.

fft(a, n=None, axis=-1) — прямое одномерное ДПФ.

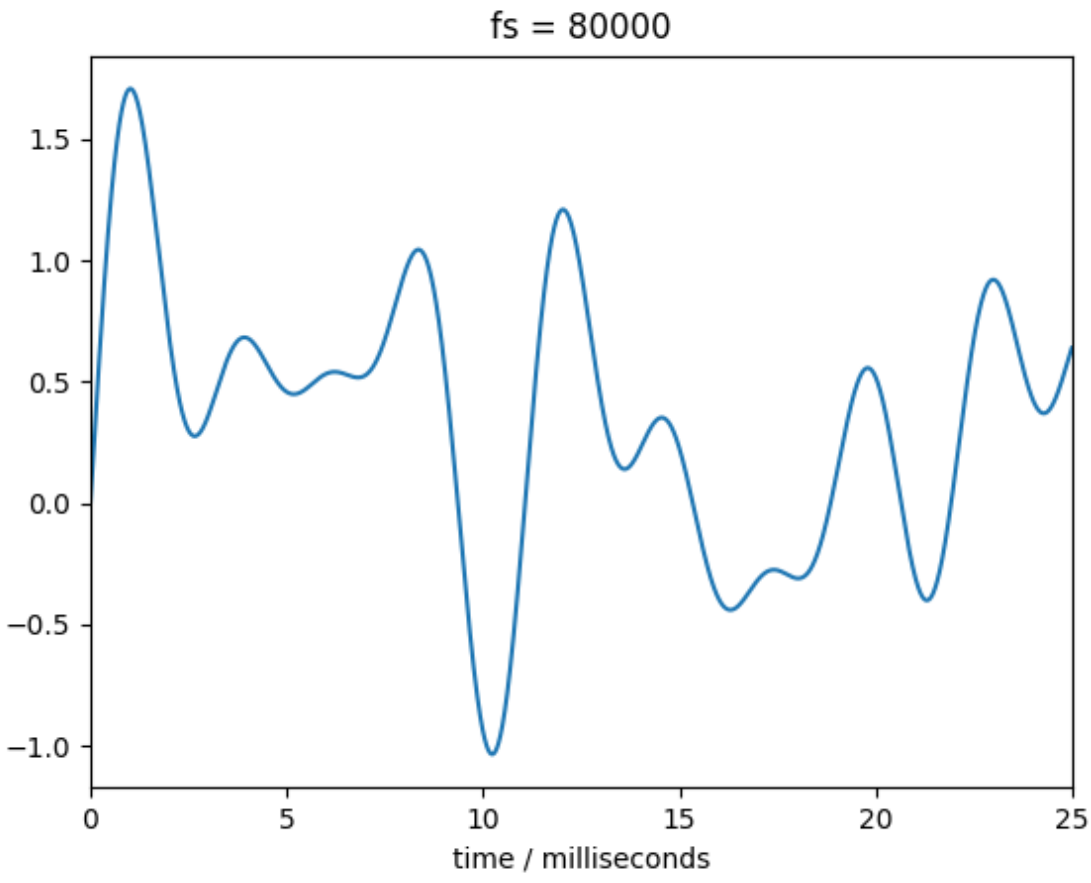
ifft(a, n=None, axis=-1) — обратное одномерное ДПФ.

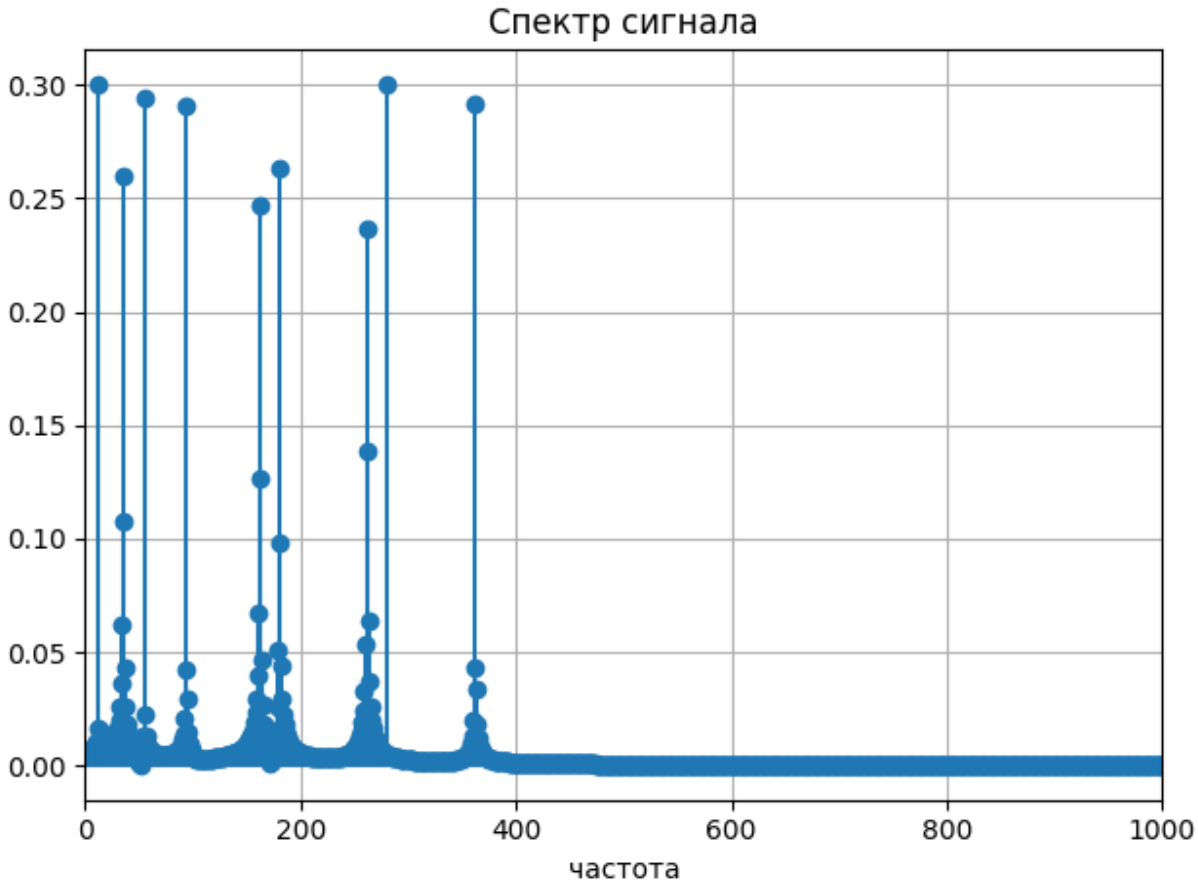
a — "сигнал", входной массив (массив numpy, array или даже питоновский список или кортеж, если в нём только числа). Массив может быть и многомерным, тогда будет вычисляться много ОДНОМЕРНЫХ ПФ по строкам (по умолчанию) или столбцам, в зависимости от параметра axis. n —

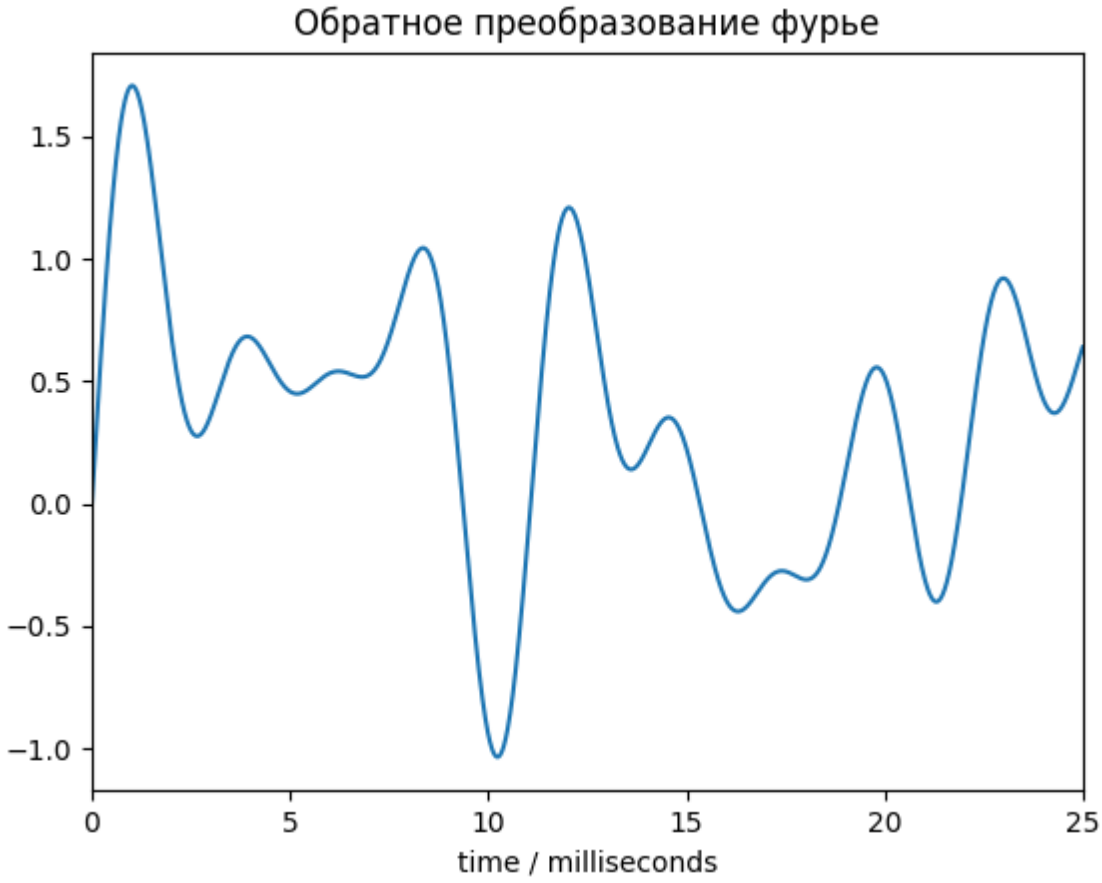
сколько элементов массива брать. Если меньше длины массива, то обрезать, если больше, то дополнить нулями, по умолчанию len(a).

```
fourier = fft(signalSum)
signalFFTabs = 2*np.abs(fourier) / fs
plt.title('Спектр сигнала')
plt.stem(signalFFTabs, basefmt='C0')
plt.xlim([0, 1000])
plt.xlabel('частота')
plt.grid()
plt.tight_layout()
plt.show()

fourierBack = ifft(fourier)
plt.plot(timeSamples[:2000] * 1000, fourierBack[:2000])
plt.title("Обратное преобразование фурье")
plt.xlabel("time / milliseconds")
plt.xlim(0, 25)
plt.show()
```




Спектр сигнала
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0
200
400
600
800
1000
частота


Обратное преобразование фурье
1.5
1.0
0.5
0.0
−0.5
−1.0
0
5
10
15
20
25
time / milliseconds