Цифровая обработка сигналов. Дискретизация.

Ruslan V. Akhpashev - 2021, e-mail: ruslan.akhpashev@gmail.com

В данной лабораторной работе мы научимся:

- Генерировать базовые периодические сигналы и их сумму,
- Выводить характеристики сигнала на графике с помощью библиотеки **matplotlib**,
- Воспроизводить сгенерированные сигналы,
- Проводить анализировать вияния частоты дискретизации на качество воспроизводимого звука.

Сигналы

Первым делом необходимо сгенерировать и проиграть в аудио обычную синусоиду. Что такое синусоида? Синусоида - обычный периодическая функция с определенными свойствами (на матанализе вы это проходили). Выглядит она следующим образом:

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

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

Для генерации сигнала необходимо заранее определиться с параметрами сигнала, которые были описаны выше.

Для генерации синусоидального сигнала необходимо знать следующие параметры (характеристики) функции:

Для начана нам необходимо получить временные отсчеты, в которых будет существовать наша функция:

In [1]:
```
duration = 4  # длительность сигнала в секундах
amplitude = 0.3  # амплитуда  (в пределах: +-1.0)
frequency = 400  # частота сигнала в [Гц]
# т.к. невозможно программно организовать аналоговый сигнал, необходимо обозначить
# количество временных отчетов, т.е. частоту дискретизации (об этом чуть позже)
fs = 80000 # 80 тыс. временных отсчетов в 1 секунду
```

In [2]:
```
import numpy as np

timeSamples = np.arange(np.ceil(duration * fs)) / fs
timeSamples[1]
```

Out[2]: 1.25e-05

Вы можете заметить, что функция **np.arange** генерирует массив размером **duration * fs** (с округлением произведения вниз, для получения целого числа отсчетов), затем каждый элемент массива делится **fs**, получая при этом небольшие дискретные отсчеты по времени (потенциально).

Теперь получим значения функции синуса:

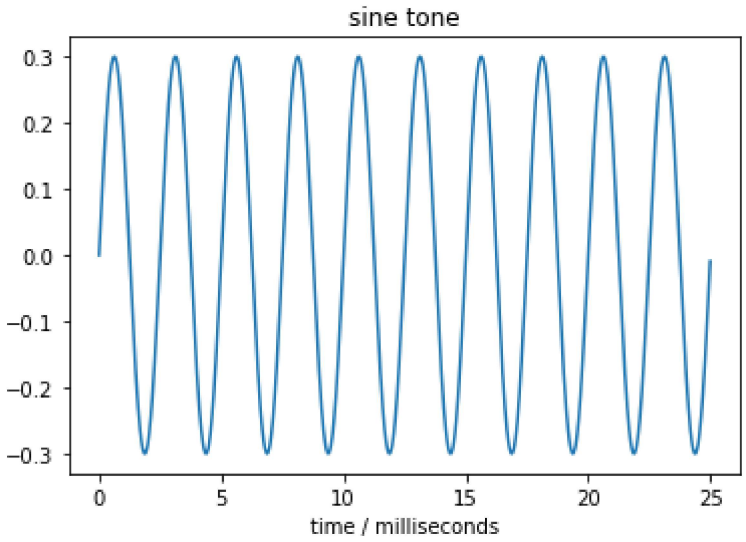
In [3]:
```
signal = amplitude * np.sin(2 * np.pi * frequency * timeSamples)
signal
```

Out[3]:
```
array([ 0.        ,  0.00942323,  0.01883716, ..., -0.02823249,
       -0.01883716, -0.00942323])
```

In [4]:
```
import matplotlib
import matplotlib.pyplot as plt

#для удобства отрезаме первые 2000 элементов каждого массива времени и амплитуды
plt.plot(timeSamples[:2000] * 1000, signal[:2000])
plt.title("sine tone")
plt.xlabel("time / milliseconds")
#plt.ylim(-1.1, 1.1);
```

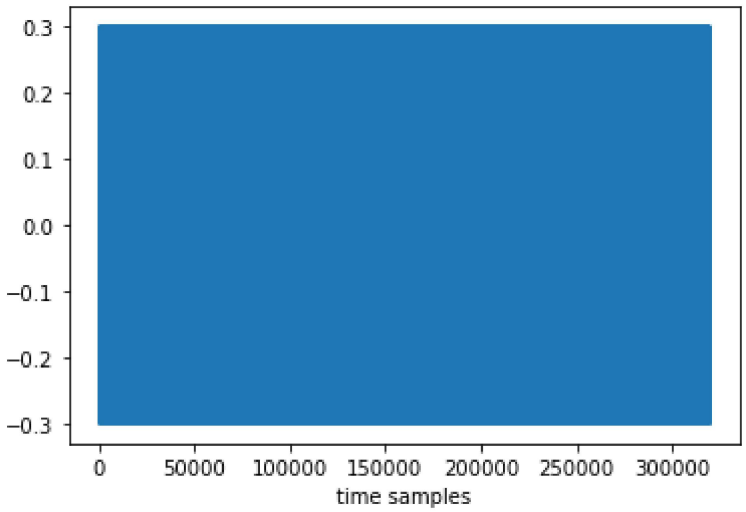
Out[4]:
```
Text(0.5, 0, 'time / milliseconds')
```



Вот как выглядит сгенерированная функция для всего количества отсчетов **(важно отметить, что это отсчеты(индексы элементов массива, значит, в теории, их можно и нужно масштабировать по времени))**:

In [5]:
```
plt.plot(signal)
plt.xlabel("time samples")
#plt.ylim(-1.1, 1.1);
```

Out[5]:
```
Text(0.5, 0, 'time samples')
```



Воспроизведение сигнала

In [6]:
```
import sounddevice as sd

sd.play(signal, fs) # функция воспроизведения массива signal, с частотой дискретизац
```

Попробуем сформировать несколько сигналов с разной частотой и отпрвить в динамики в виде суммы всех сигналов:

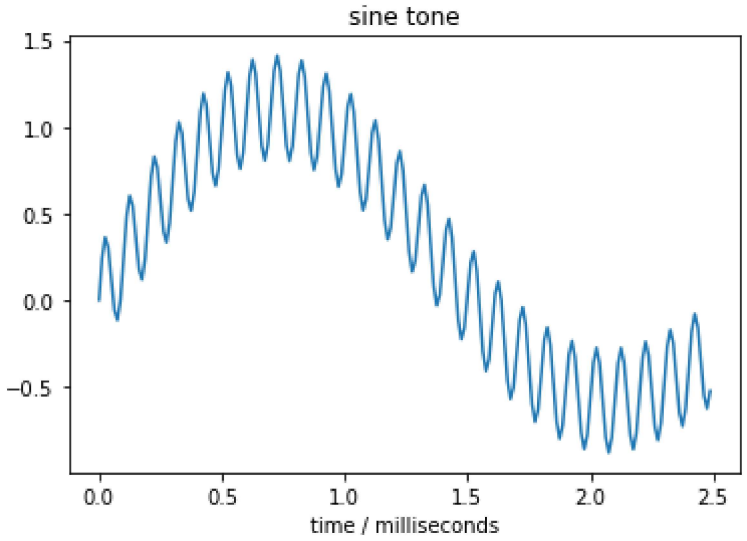
In [8]:
```
signal2 = amplitude * np.sin(2 * np.pi * 200 * timeSamples)
signal3 = amplitude * np.sin(2 * np.pi * 300 * timeSamples)
signal4 = amplitude * np.sin(2 * np.pi * 400 * timeSamples)
signal5 = amplitude * np.sin(2 * np.pi * 10000 * timeSamples)

signalSumm = signal + signal2 + signal3 + signal4 + signal5
```

На графике му получим:

In [9]:
```
plt.plot(timeSamples[:200] * 1000, signalSumm[:200])
plt.title("sine tone")
plt.xlabel("time / milliseconds")
```

Out[9]:
```
Text(0.5, 0, 'time / milliseconds')
```



In [10]:
```
sd.play(signalSumm, fs)
```

Задание на лабораторную работу №2

1) Сформировать аккорд\мелодию длительностью до 5 секунд (в каждого уникальная);

2) Проверить влияние частоты дискретизации(1000; 50000; 5000++) 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.

Установка доп. модулей

NumPy

Для удобства работы с числами, массивами, математическими операциями будем использовать популярную библиотуку NumPy. - https://numpy.org/

https://pythonworld.ru/numpy/1.html - так же можно посмотреть базовый функционал библиотеки.

In [1]:
```
import numpy as np
```

as **np** - сокращает написание текста при вызове функций из данной библиотеки.

Если у вас **нет** встоенного пакета **numpy**, необходимо установить. Можно сделать двумя способами:

1. С помощью командной строки, например, прописав **pip install numpy**
2. Из ядра jupyter (примеры смотрите ниже):

In [12]:
```
import sys

#!{sys.executable} -m pip install numpy

#или

#!conda install --yes --prefix {sys.prefix} numpy
```

In [6]:
```
array1= np.arange(10) # - быстрое создание непустого массива.
print("array1 = ", array1)
array2 = np.arange(0)
print("array2 = ", array2) # - создание пустого массива
```

```
array1 =  [0 1 2 3 4 5 6 7 8 9]
array2 =  []
```

Matplotlib - билбиотека для визуализации данных

Очень подробно про данную библиотеку описано - https://pythonworld.ru/novosti-mira-python/scientific-graphics-in-python.html

Официальная документация модуля - https://matplotlib.org/

Установка модуля схожа с установкой NumPy

SoundDevice - библиотека для воспроизведения\записи звука

Официальная документатция - https://python-sounddevice.readthedocs.io/en/0.4.1/

In [ ]:
```
```