ThinkDSP. Лабораторная 10. Линейные стационарные системы.

Шерепа Никита

14 мая 2021 г.

Содержание

Список иллюстраций

Листинги

1 Упражнение 10.1

1. Задание

   Измените пример в `chap10.ipynb` и убедитесь, что дополнение нулями устраняет лишнюю ноту в начале фрагмента.

2. Ход работы

   Усечём оба сигнала до 2^{16} элементов, а затем обнуляем их до 2^{17}. Использование степени двойки делает алгоритм ДПФ наиболее эффективным.

   Построим импульсный отклик

```
response = read_wave('res/180960__kleeb__gunshot.wav')

start = 0.12
response = response.segment(start=start)
response.shift(-start)

response.truncate(2**16)
response.zero_pad(2**17)

response.normalize()
response.plot()
decorate(xlabel='Time (s)')
```

Листинг 1: Импульсный отклик

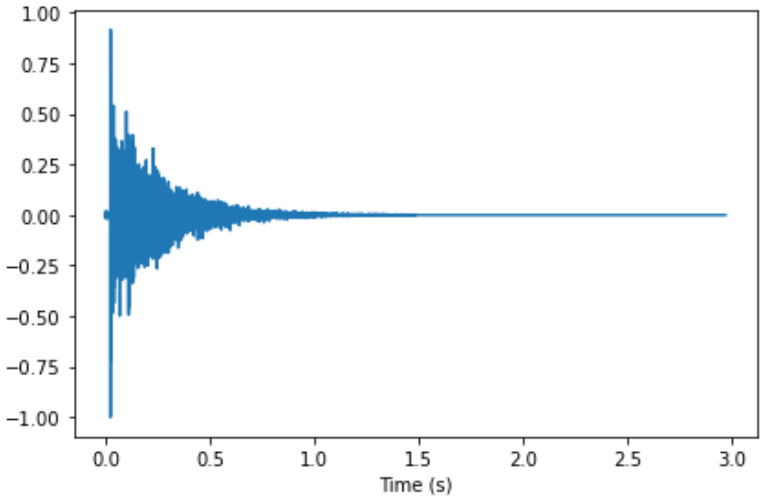

Рис. 1: Импульсный отклик

Теперь построим его спектр

```
transfer = response.make_spectrum()
transfer.plot()
decorate(xlabel='Frequency (Hz)', ylabel='Amplitude')
```

Листинг 2: Спектр импульсного отклика

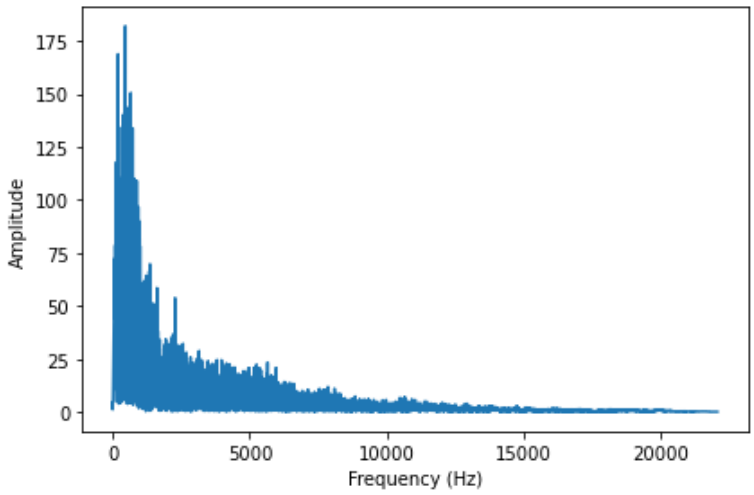

Рис. 2: Спектр импульсного отклика

Теперь возьмем другой сигнал - звук скрипки

```
violin =
    read_wave('res/92002__jcveliz__violin-origional.wav')

start = 0.11
violin = violin.segment(start=start)
violin.shift(-start)

violin.truncate(2**16)
violin.zero_pad(2**17)

violin.normalize()
violin.plot()
decorate(xlabel='Time (s)')
```

Листинг 3: Звук скрипки

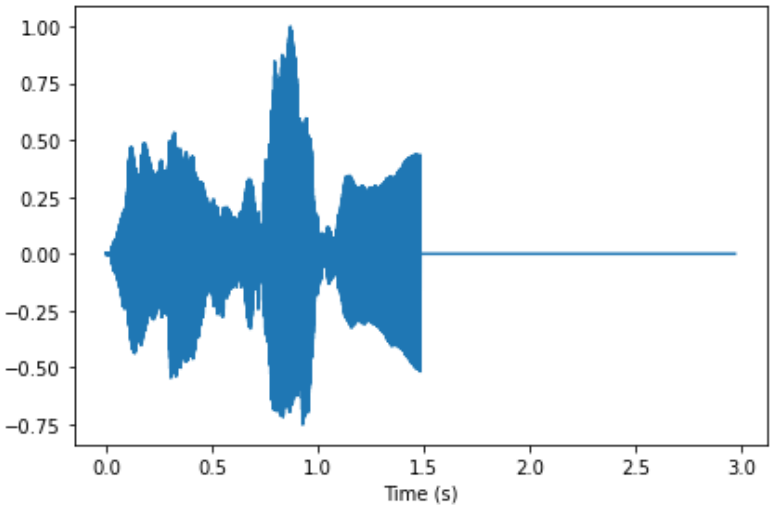

Рис. 3: Спектр импульсного отклика

Построим его спектр, умножим ДПФ сигнала на передаточную функцию и преобразуем обратно в волну

```
spectrum = violin.make_spectrum()

output = (spectrum * transfer).make_wave()
output.normalize()

output.plot()
```

Листинг 4: Визуализация сигнала

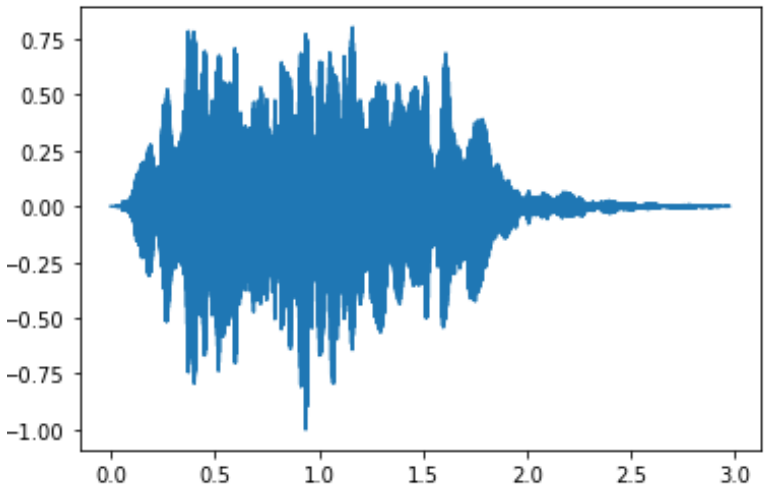

Рис. 4: Визуализация сигнала

Прослушаем

```
output.make_audio()
```

Листинг 5: Прослушивание результата

Теперь лишнюю ноту в начале не слышно

Попробуем добиться таких же результатов с использованием `np.convolve` и `scipy.signal.fftconvolve`

Для начала, избавимся от нулевого отступа

```
response.truncate(2**16)
response.plot()
```

Листинг 6: Избавляемся от нулевого отступа

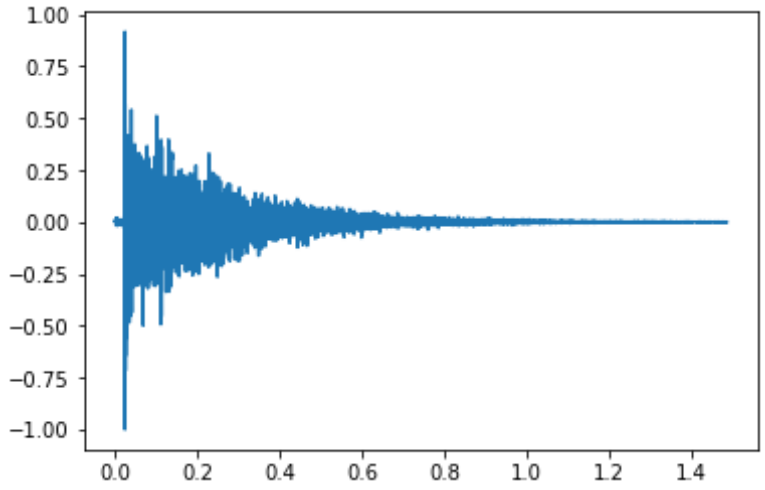

Рис. 5: Избавились от нулевого отступа

```
violin.truncate(2**16)
violin.plot()
```

Листинг 7: Избавляемся от нулевого отступа

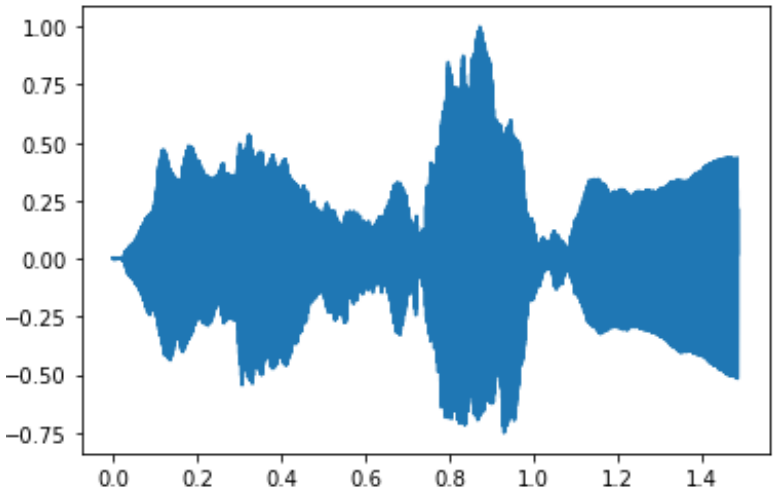

Рис. 6: Избавились от нулевого отступа

Теперь сравним с `np.convolve`

```
output2 = violin.convolve(response)
```

```
output2.plot()
len(output), len(output2)

Output
(131072, 131071)
```

Листинг 8: Сравниваем с `np.convolve`

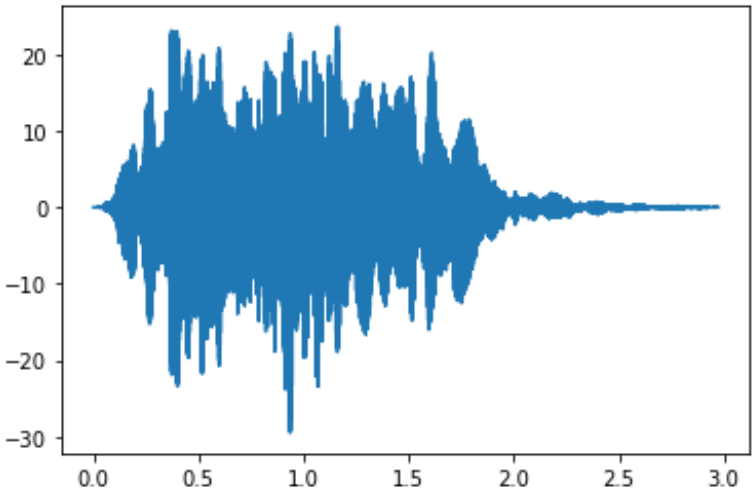

Рис. 7: Сравниваем с `np.convolve`

Видим, что сигналы похожи, хотя длина одного совсем чуть-чуть отличается от другого.

Теперь поэкспериментируем с `scipy.signal.fftconvolve`. Этот метод использует БПФ, поэтому он быстрее

```
from thinkdsp import Wave

import scipy.signal
ys = scipy.signal.fftconvolve(violin.ys, response.ys)
output3 = Wave(ys, framerate=violin.framerate)
output3.plot()
```

Листинг 9: Используем `scipy.signal.fftconvolve`

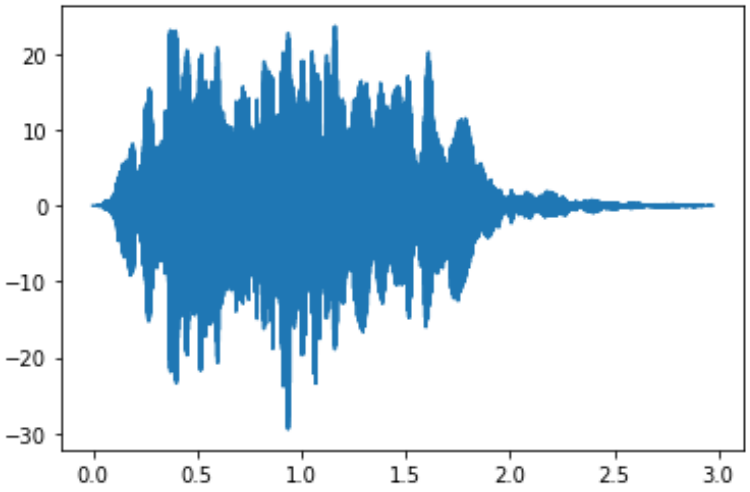

Рис. 8: Используем `scipy.signal.fftconvolve`

Результат получился такой же, как и в прошлом случае, и звучит также.

```
output2.max_diff(output3)

Output
1.4210854715202004e-14
```

Листинг 10: Используем `scipy.signal.fftconvolve`

Разница между `np.convolve` и `scipy.signal.fftconvolve` = 1.4210854715202004e-14

2 Упражнение 10.2

1. Задание

   Смоделируйте двумя способами звучание записи в том пространстве, где была измерена импульсная характеристика, как сверткой самой записи с импульсной характеристикой, так и умножением ДПФ записи на вычисленный фильтр, соответсвующий импульсной характеристике.

2. Ход работы

   В качестве примера я взял звук волныки. Визуализируем его.

```
response = read_wave('res/bagpipe_music.wav')

start = 0
duration = 5
response = response.segment(duration=duration)
response.shift(-start)

response.normalize()
response.plot()
decorate(xlabel='Time (s)')
```

Листинг 11: Звук волынки

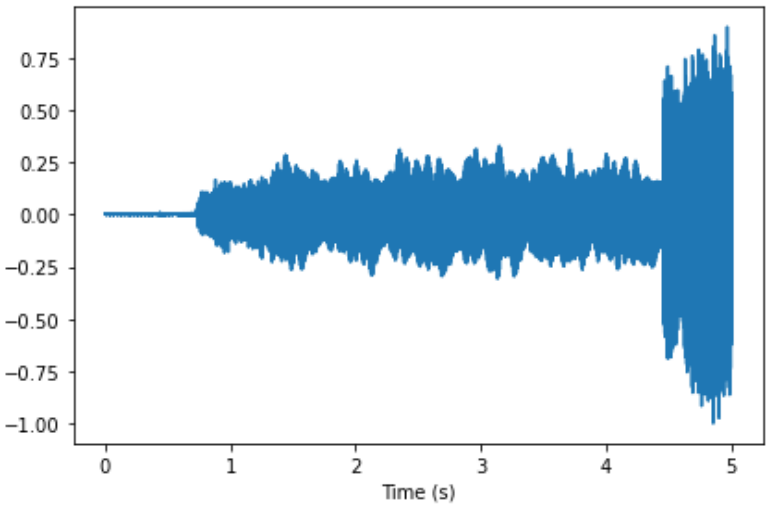

Рис. 9: Звук волынки

Вычислим спектр

```
transfer = response.make_spectrum()
transfer.plot()
decorate(xlabel='Frequency (Hz)', ylabel='Amplitude')
```

Листинг 12: Вычисляем спектр

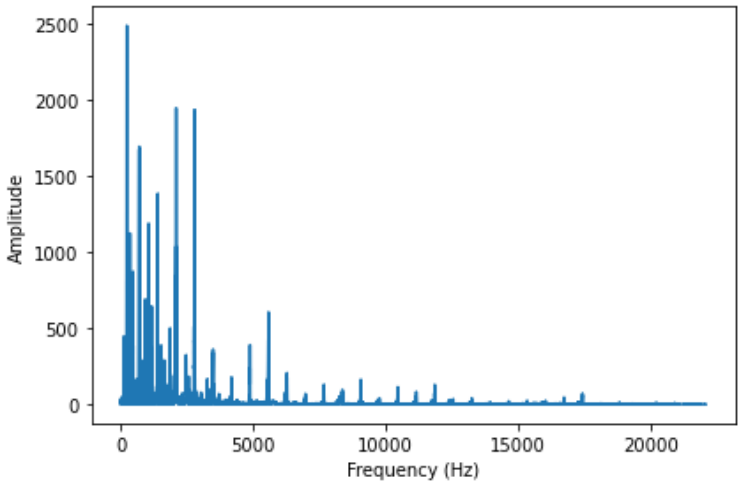

Рис. 10: Спектр звука

Рассмотрим передаточную функцию в логарифмическом масштабе

```
transfer.plot()
decorate(xlabel='Frequency (Hz)', ylabel='Amplitude',
xscale='log', yscale='log')
```

Листинг 13: Передаточная функция в логарифмическом масштабе

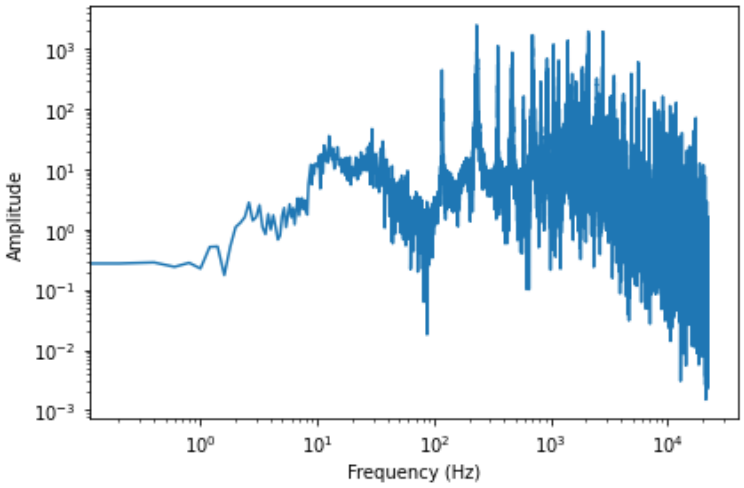

Рис. 11: Передаточная функция в логарифмическом масштабе

Частота звука волныки = 612 Возьмем еще один звук с примерно такой же частотой - звук флейты с частотой = 630

```
wave = read_wave('res/flute_music.wav')

start = 0.0
wave = wave.segment(start=start)
wave.shift(-start)

wave.truncate(len(response))
wave.normalize()
wave.plot()
decorate(xlabel='Time (s)')
```

Листинг 14: Звук флейты

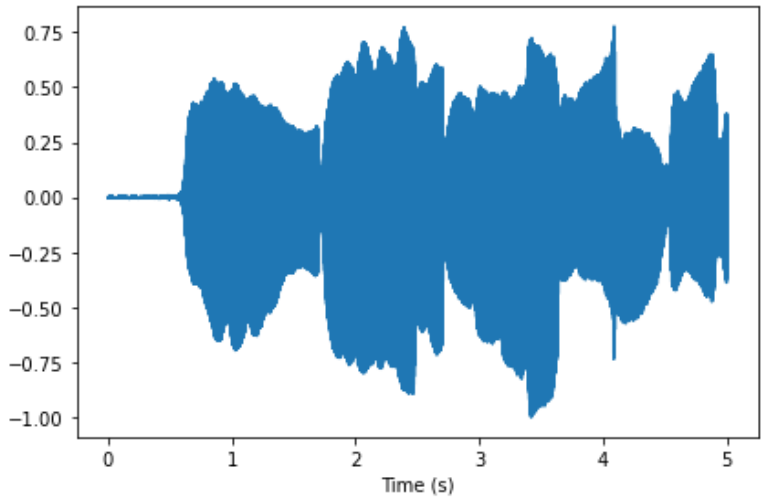

Рис. 12: Звук флейты

Сравним спектры волынки и флейты

```
len(spectrum.hs), len(transfer.hs)

Output
(110251, 110251)

spectrum.fs
Output
array([0.00000e+00, 2.00000e-01, 4.00000e-01, ...,
    2.20496e+04,
2.20498e+04, 2.20500e+04])

transfer.fs
Output
array([0.00000e+00, 2.00000e-01, 4.00000e-01, ...,
    2.20496e+04,
2.20498e+04, 2.20500e+04])
```

Листинг 15: Сравниваем спектры

Спектры совпадают

Теперь трансформируем запись

```
output = (spectrum * transfer).make_wave()
```

```
output.normalize()
```

Листинг 16: Траснформация записи

И сравним оригинал с транформацией

```
wave.plot()
```

Листинг 17: Оригинальный звук

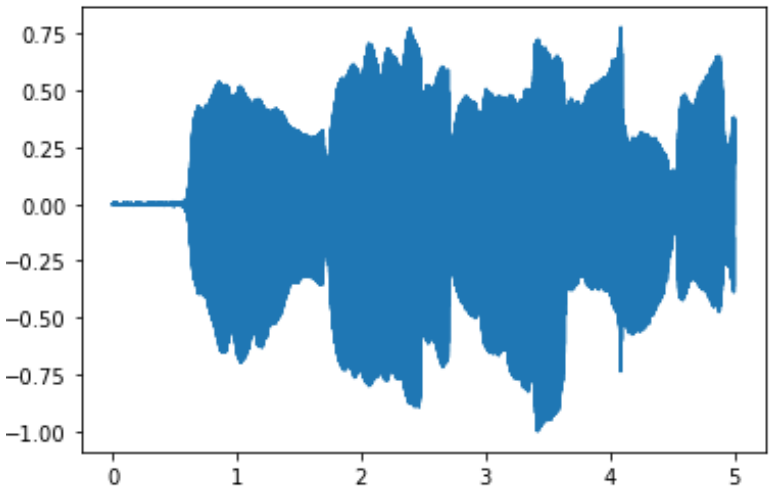

Рис. 13: Оригинальный звук

```
output.plot()
```

Листинг 18: Трансформированный звук

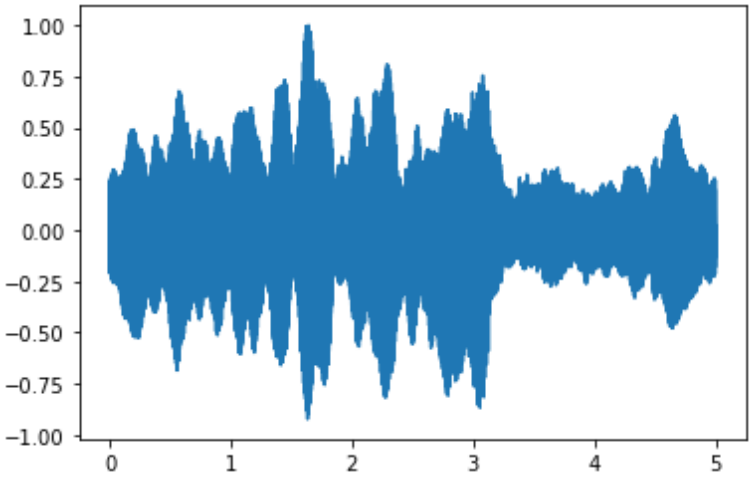

Рис. 14: Трансформированный звук

По звучанию трансформированный звук стал похож на звук из космоса.

Теперь можно сказать, что все эти проделанные операции похожи на свертку. Попробуем применить `convolve`

```
convolved2 = wave.convolve(response)
convolved2.normalize()
convolved2.make_audio()
```

Листинг 19: Оригинальный звук

Результат звучит ожидаемо точно также, как и трансформированная запись.

3 Вывод

В результате выполнения лабораторной работы получены навыки работы с фильтрами и импульсными характеристиками.