

Лабораторная работа №10
Линейные стационарные системы

Кобыжев Александр

12 апреля 2021 г.

Оглавление

1	Упражнение 10.1	4
2	Упражнение 10.2	11
3	Выводы	17

Список иллюстраций

1.1	Визуализация сигнала	5
1.2	Спектр сигнала	5
1.3	Визуализация сигнала	6
1.4	Визуализация сигнала	7
1.5	Визуализация сигнала	8
1.6	Визуализация сигнала	8
1.7	Визуализация сигнала	9
1.8	Визуализация итогового сигнала	10
2.1	Визуализация звука	12
2.2	Спектр звука	12
2.3	Визуализация звука в логарифмическом масштабе	13
2.4	Визуализация звука	14
2.5	Визуализация оригинального звука	15
2.6	Визуализация преобразённого звука	16

Листинги

1.1	Усечение сигнала	4
1.2	Спектр сигнала	5
1.3	Усечение сигнала	6
1.4	Спектр сигнала	6
1.5	Совмещение сигналов	6
1.6	Визуализация сигнала	7
1.7	Прослушивание сигнала	7
1.8	Избавление от нулевого отступа	7
1.9	Избавление от нулевого отступа	8
1.10	Объединение сигналов	8
1.11	Прослушивание итогового сигнала	9
1.12	Сравнение длин сигналов	9
1.13	Изменение сигнала при помощи <code>scipy.signal.fftconvolve</code>	9
1.14	Визуализация итогового сигнала	9
1.15	Прослушивание сигнала	10
1.16	Сравнение результатов	10
2.1	Загрузка звука	11
2.2	Спектр звука	12
2.3	Визуализация звука в логарифмическом масштабе	13
2.4	Загрузка звука	13
2.5	Частота дискретизации первого звука	14
2.6	Частота дискретизации второго звука	14
2.7	Спектр звука	14
2.8	Длина записей	14
2.9	Длина первой записи	15
2.10	Длина второй записи	15
2.11	Перемножение	15
2.12	Визуализация оригинального звука	15
2.13	Визуализация преобразённого звука	15
2.14	Моделирование при помощи <code>convolve</code>	16

Глава 1

Упражнение 10.1

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

Вот импульсный отклик:

```
1 response = thinkdsp.read_wave('180960__kleeb__gunshot.wav')
2
3 start = 0.12
4 response = response.segment(start=start)
5 response.shift(-start)
6
7 response.truncate(2**16)
8 response.zero_pad(2**17)
9
10 response.normalize()
11 response.plot()
12 thinkplot.config(xlabel='Time (s)', ylim=[-1.05, 1.05])
```

Листинг 1.1: Усечение сигнала

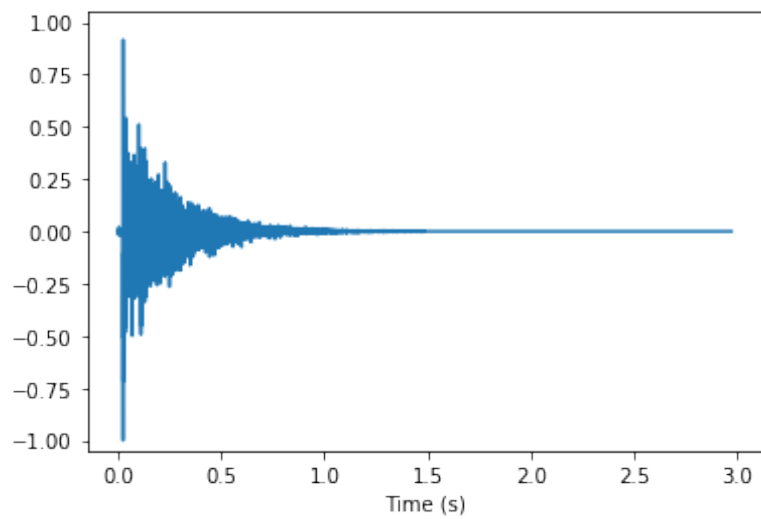


Рис. 1.1: Визуализация сигнала

А теперь составим спектр:

```

1 transfer = response.make_spectrum()
2 transfer.plot()
3 thinkplot.config(xlabel='Frequency (Hz)', ylabel='Amplitude')

```

Листинг 1.2: Спектр сигнала

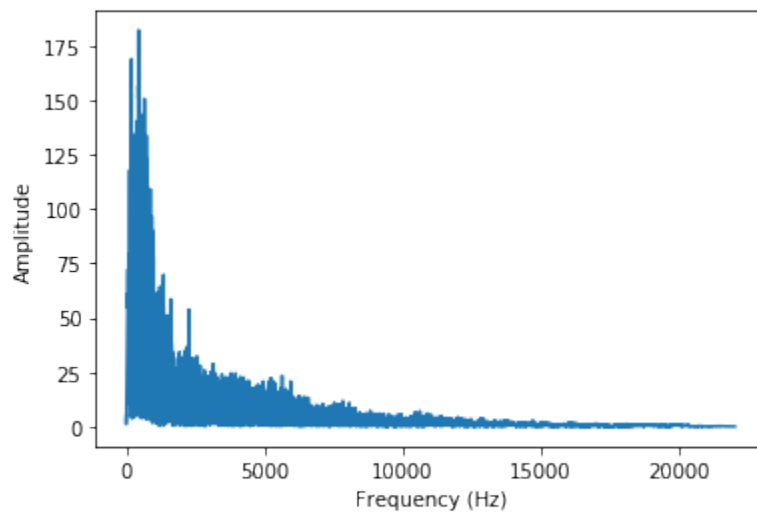


Рис. 1.2: Спектр сигнала

Рассмотрим другой сигнал.

```

1 violin = thinkdsp.read_wave('92002__jcveliz__violin-original.wav')
2
3 start = 0.11
4 violin = violin.segment(start=start)
5 violin.shift(-start)
6
7 violin.truncate(2**16)
8 violin.zero_pad(2**17)
9
10 violin.normalize()
11 violin.plot()
12 thinkplot.config(xlabel='Time (s)', ylim=[-1.05, 1.05])

```

Листинг 1.3: Усечение сигнала

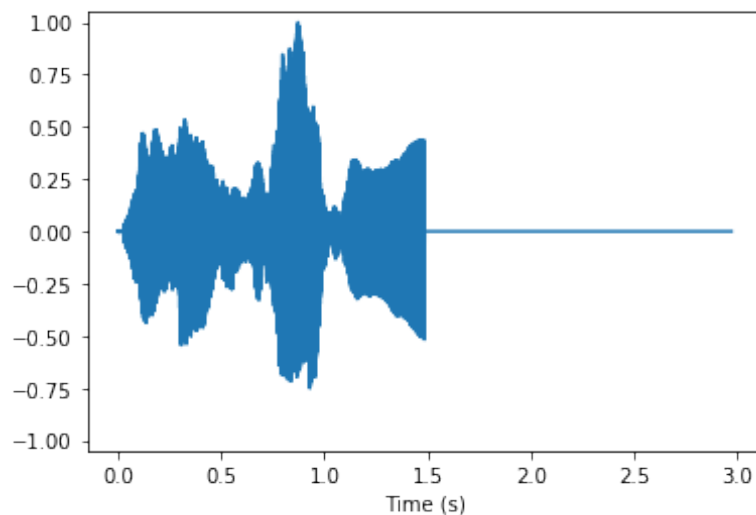


Рис. 1.3: Визуализация сигнала

Составим спектр:

```

1 spectrum = violin.make_spectrum()

```

Листинг 1.4: Спектр сигнала

Теперь умножим ДПФ сигнала на передаточную функцию и преобразуем обратно в волну:

```

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

```

Листинг 1.5: Совмещение сигналов

Визуализируем полученный сигнал.

```
1 output.plot()
```

Листинг 1.6: Визуализация сигнала

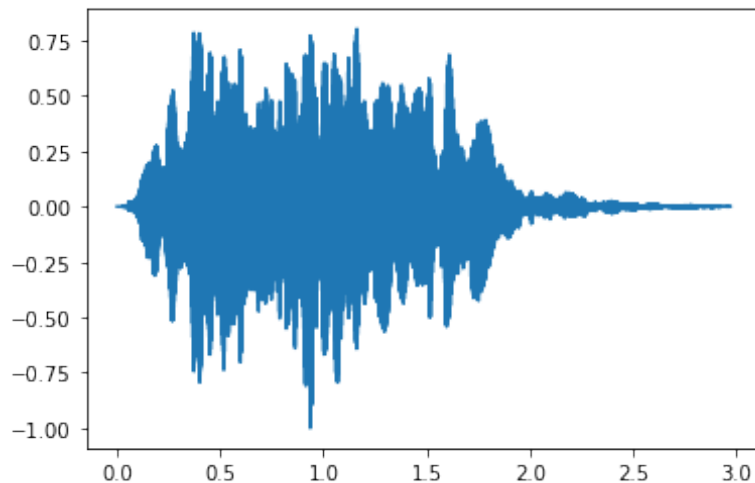


Рис. 1.4: Визуализация сигнала

Результат не выглядит так, как будто он оборачивается вокруг:

```
1 output.make_audio()
```

Листинг 1.7: Прослушивание сигнала

Лишнюю ноту в начале не слышно. Мы должны получить такие же результаты от `np.convolve` и `scipy.signal.fftconvolve`.

Сначала избавимся от нулевого отступа:

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

Листинг 1.8: Избавление от нулевого отступа

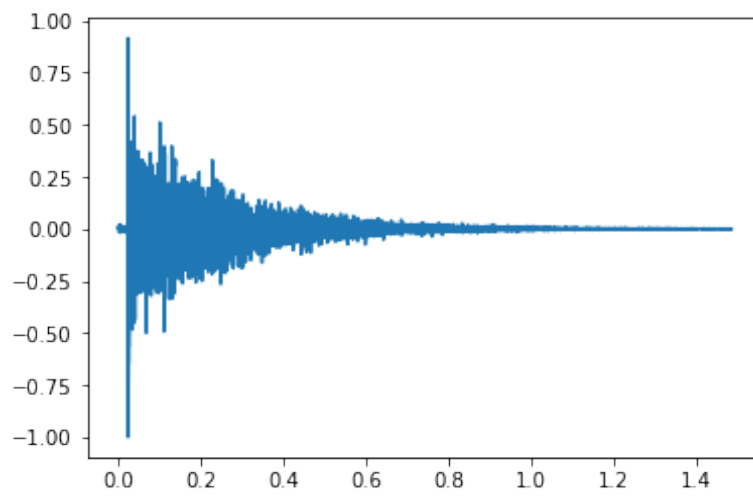


Рис. 1.5: Визуализация сигнала

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

Листинг 1.9: Избавление от нулевого отступа

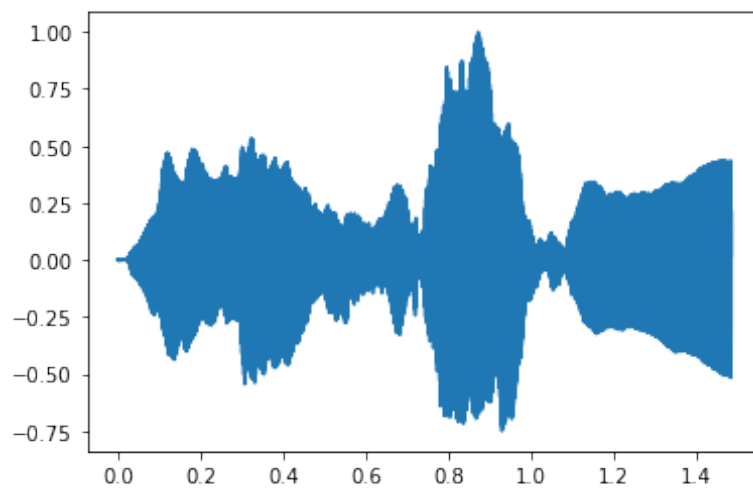


Рис. 1.6: Визуализация сигнала

Теперь мы можем сравнить с `np.convolve`:

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

```
2 output2.plot()
```

Листинг 1.10: Объединение сигналов

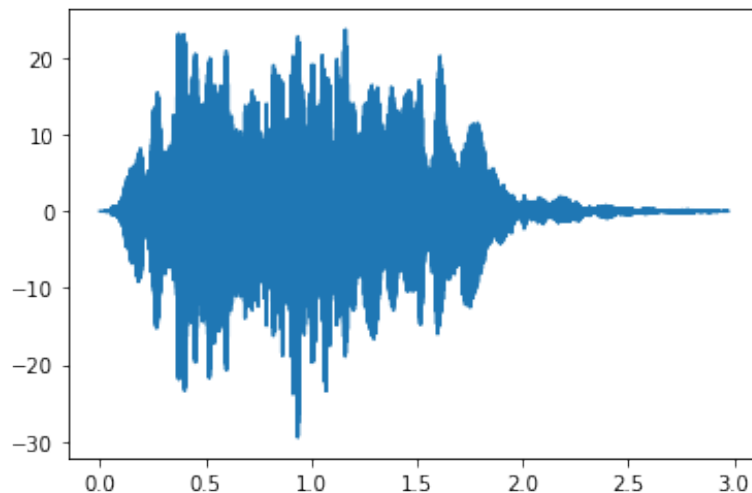


Рис. 1.7: Визуализация сигнала

Результаты максимально похожи.

```
1 output2.make_audio()
```

Листинг 1.11: Прослушивание итогового сигнала

Звук такой же, но длина не та.

```
1 len(output), len(output2)
```

Листинг 1.12: Сравнение длин сигналов

На выходе получились значения (131072, 131071).

`scipy.signal.fftconvolve` делает то же самое, но, как следует из названия, он использует ДПФ, поэтому он значительно быстрее:

```
1 import scipy.signal
2 ys = scipy.signal.fftconvolve(violin.ys, response.ys)
3 output3 = thinkdsp.Wave(ys, framerate=violin.framerate)
```

Листинг 1.13: Изменение сигнала при помощи `scipy.signal.fftconvolve`

```
1 output3.plot()
```

Листинг 1.14: Визуализация итогового сигнала

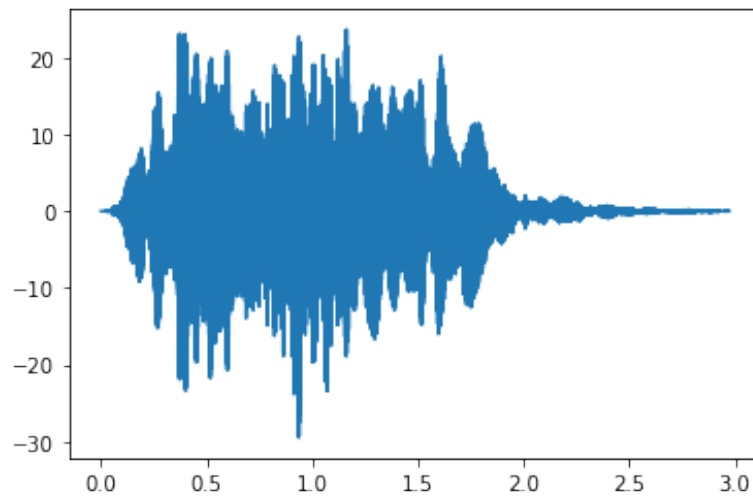


Рис. 1.8: Визуализация итогового сигнала

```
1 output3.make_audio()
```

Листинг 1.15: Прослушивание сигнала

Результат тот же. В пределах ошибки с плавающей запятой результаты такие же:

```
1 output2.max_diff(output3)
```

Листинг 1.16: Сравнение результатов

Ошибка равна $2.1316282072803006 \times 10^{-14}$.

Глава 2

Упражнение 10.2

В качестве импульсной характеристики взят за основу звук какой-то церкви из Великобритании. Звук можно скачать по [ссылке](#).

```
1 response = thinkdsp.read_wave('mono_sir1.wav')
2
3 start = 0
4 duration = 5
5 response = response.segment(duration=duration)
6 response.shift(-start)
7
8 response.normalize()
9 response.plot()
10 thinkplot.config(xlabel='Time (s)', ylim=[-1.05, 1.05])
```

Листинг 2.1: Загрузка звука

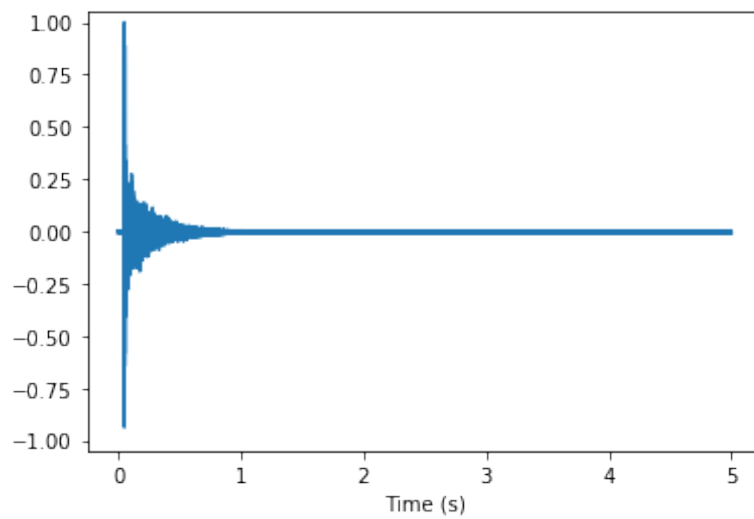


Рис. 2.1: Визуализация звука

ДПФ импульсной характеристики - это передаточная функция:

```

1 transfer = response.make_spectrum()
2 transfer.plot()
3 thinkplot.config(xlabel='Frequency (Hz)', ylabel='Amplitude')

```

Листинг 2.2: Спектр звука

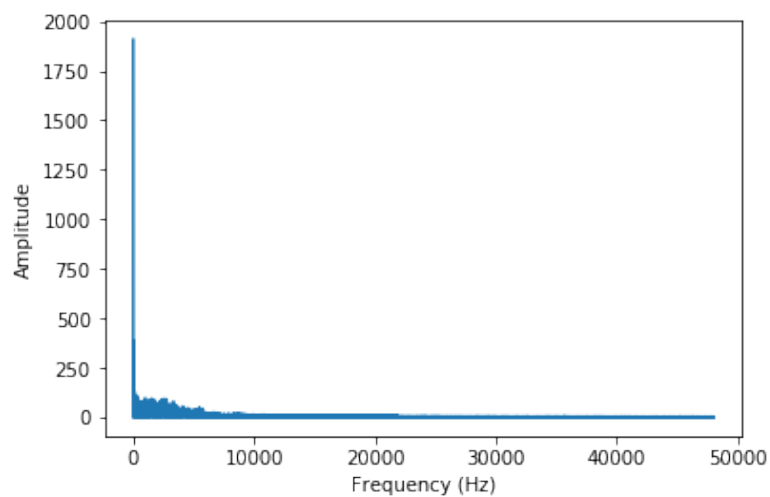


Рис. 2.2: Спектр звука

Рассмотрим передаточную функцию в логарифмическом масштабе:

```

1 transfer.plot()
2 thinkplot.config(xlabel='Frequency (Hz)', ylabel='Amplitude',
3                  xscale='log', yscale='log')

```

Листинг 2.3: Визуализация звука в логарифмическом масштабе

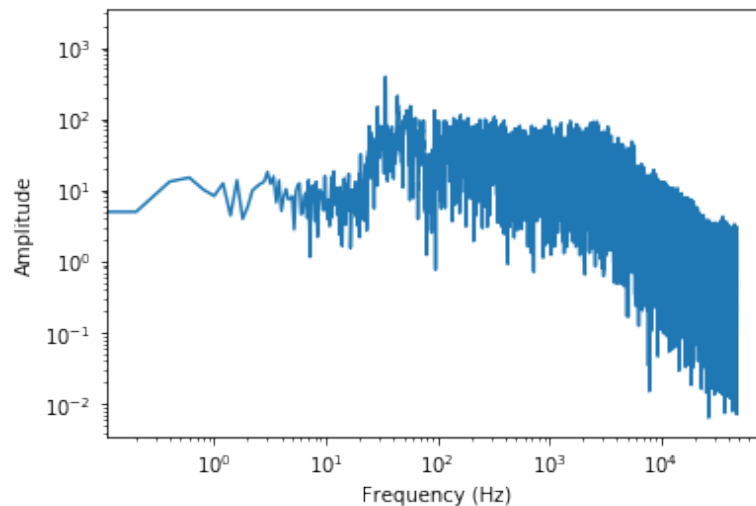


Рис. 2.3: Визуализация звука в логарифмическом масштабе

Теперь мы можем смоделировать звучание записи, если бы её воспроизвели в одной комнате и записали бы таким же образом.

Запись нам нужна той же частоты дискретизации, что и предыдущая, поэтому пришлось искать в интернете запись с 96 kHz. Запись можно скачать по ссылке.

```

1 wave =
    thinkdsp.read_wave('88498__shakaharu__ball-basketball-drop.wav')
2
3 start = 0.0
4 wave = wave.segment(start=start)
5 wave.shift(-start)
6
7 wave.truncate(len(response))
8 wave.normalize()
9 wave.plot()
10 thinkplot.config(xlabel='Time (s)', ylim=[-1.05, 1.05])

```

Листинг 2.4: Загрузка звука

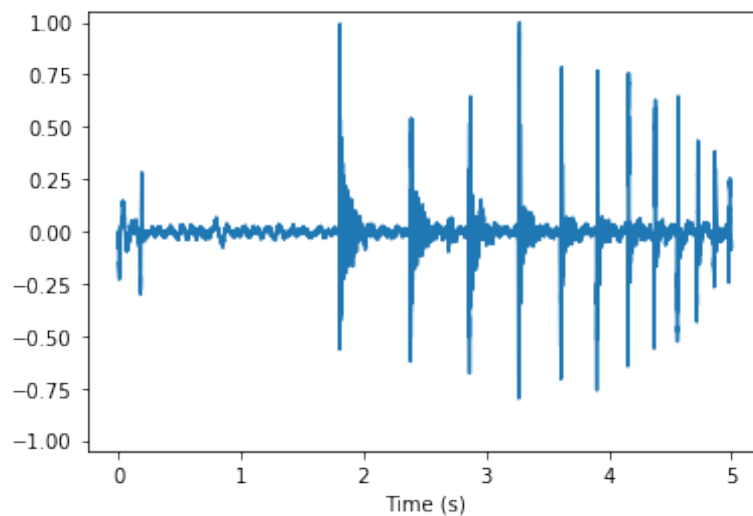


Рис. 2.4: Визуализация звука

Проверим частоту дискретизации.

```
1 #!pip install pydub
2 from pydub import AudioSegment
3 song = AudioSegment.from_mp3("mono_s1r1.wav")
4 song.frame_rate
```

Листинг 2.5: Частота дискретизации первого звука

```
1 song2 =
    AudioSegment.from_mp3("88498__shakaharu__ball-basketball-drop.wav")
2 song2.frame_rate
```

Листинг 2.6: Частота дискретизации второго звука

Оба звука имеют частоту дискретизации 96000, что ожидаемо.

Теперь мы вычисляем ДПФ записи скачущего баскетбольного мяча.

```
1 spectrum = wave.make_spectrum()
```

Листинг 2.7: Спектр звука

Обрежем запись скрипки до той же длины, что и импульсная характеристика:

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

Листинг 2.8: Длина записей

Длины совпадают и равны (240001, 240001).

```
1 spectrum.fs
```

Листинг 2.9: Длина первой записи

```
1 transfer.fs
```

Листинг 2.10: Длина второй записи

Массивы значений совпадают и равны `array([0. , 0.2, 0.4, ..., 47999.6, 47999.8, 48000.])`.

Мы можем умножить в частотной области и преобразовать обратно во временную область.

```
1 output = (spectrum * transfer).make_wave()
2 output.normalize()
```

Листинг 2.11: Перемножение

Рассмотрим сравнение оригинальной и преобразованной записи:

```
1 wave.plot()
```

Листинг 2.12: Визуализация оригинального звука

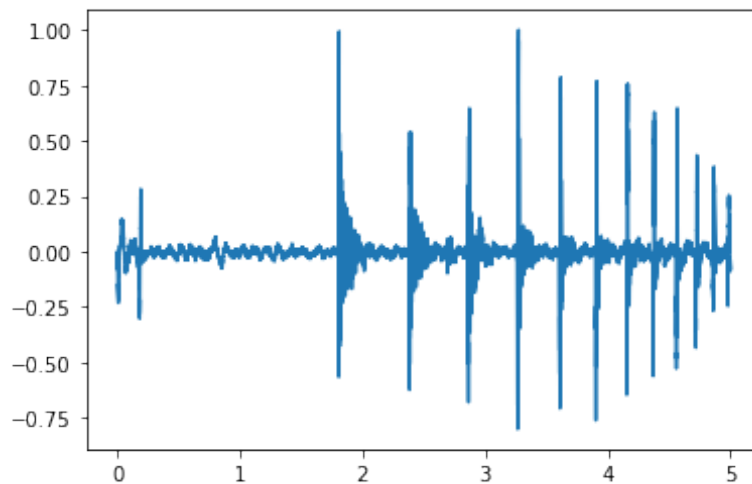


Рис. 2.5: Визуализация оригинального звука

```
1 output.plot()
```

Листинг 2.13: Визуализация преобразённого звука

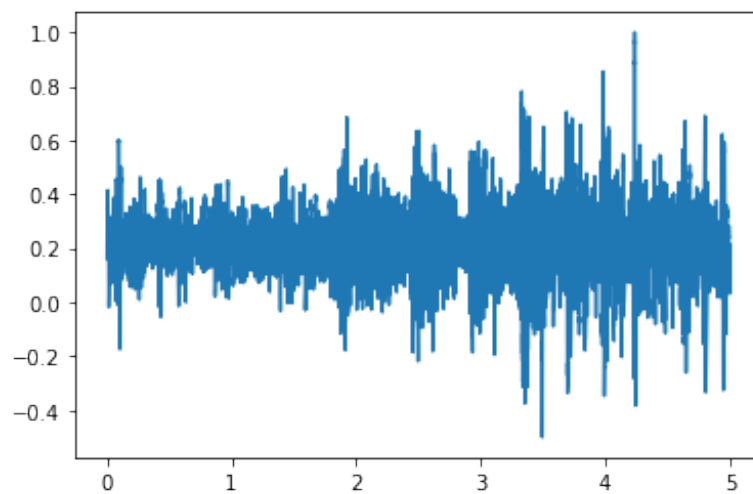


Рис. 2.6: Визуализация преобразённого звука

Теперь, когда мы распознаем эту операцию как свёртку, мы можем вычислить ее с помощью метода `convolve`:

```
1 convolved2 = wave.convolve(response)
2 convolved2.normalize()
3 convolved2.make_audio()
```

Листинг 2.14: Моделирование при помощи `convolve`

Глава 3

Выводы

Во время выполнения лабораторной работы получены навыки работы с теорией сигналов и систем, а также применения теоремы свёртки, характеризующая линейные, инвариантные во времени системы.