Лабораторная работа № 11. Модуляция и выборка (квантование).

3530901/80201, Шелаев Н. Р.

31 мая 2021 г.

Оглавление

Список иллюстраций

Листинги

Глава 1

Свертка с импульсами

Свертка с пачкой импульсов - это сложение смещенных и масштабированных копий сигнала.

```
1       from thinkdsp import read_wave
2
3       wave = read_wave('253887__themusicalnomad__positive-
    beeps.wav')
4       wave.normalize()
5       wave.plot()
6
```

Листинг 1.1: Получение сигнала

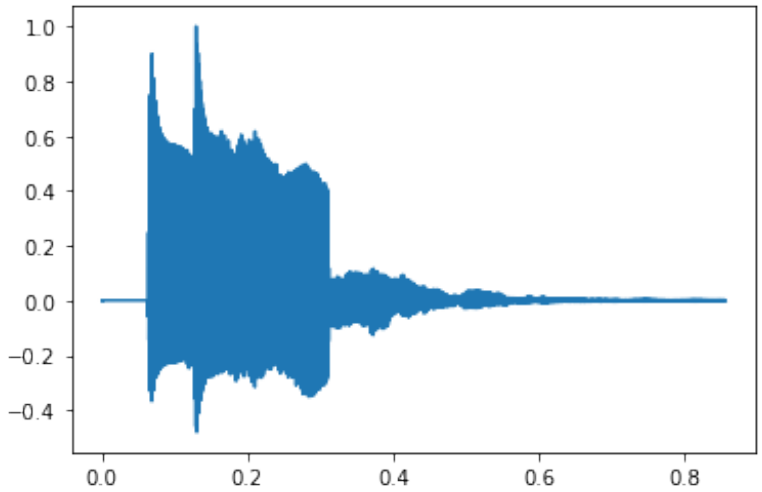

Рис. 1.1: Полученный сигнал

```
1       from thinkdsp import Impulses
```

```
2
3       imp_sig = Impulses([0.005, 0.3, 0.6, 0.9], amps = [1,
    0.5, 0.25, 0.1])
4       impulses = imp_sig.make_wave(start = 0, duration = 1.0,
     framerate = wave.framerate)
5       impulses.plot()
6
```

Листинг 1.2: Последовательность из 4-х импульсов

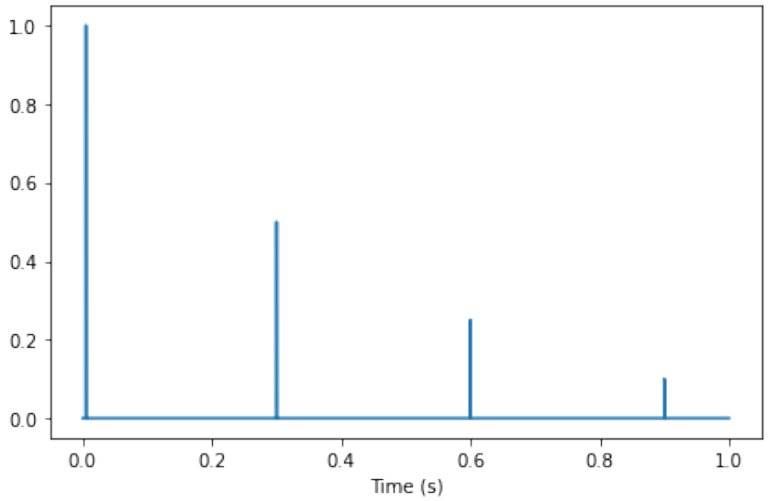

Рис. 1.2: 4 импульса

```
1       convolved = wave.convolve(impulses)
2       convolved.plot()
3
```

Листинг 1.3: Применим свертку сигнала с импульсами

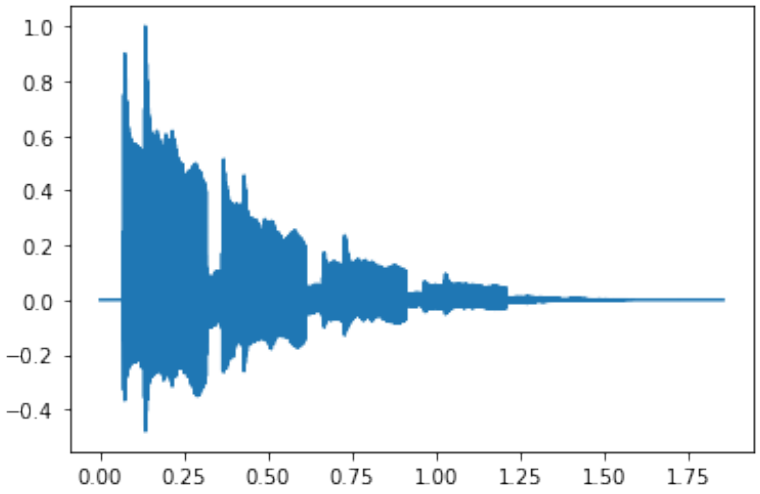

Рис. 1.3: Результат свертки

Свертка сигнала с импульсами дает сдвинутые, масштабированные копии сигнала.

Глава 2

Амплитудная модуляция (AM)

AM - это умножение «несущего» сигнала с полезным сигналом.

```
1       wave=read_wave('105977__wcfl10__favorite-station.wav')
2       wave.unbias()
3       wave.normalize()
4       spectrum = wave.make_spectrum()
5       spectrum.plot()
6       spectrum = wave.make_spectrum(full = True)
7       spectrum.plot()
8
```

Листинг 2.1: Полезный сигнал

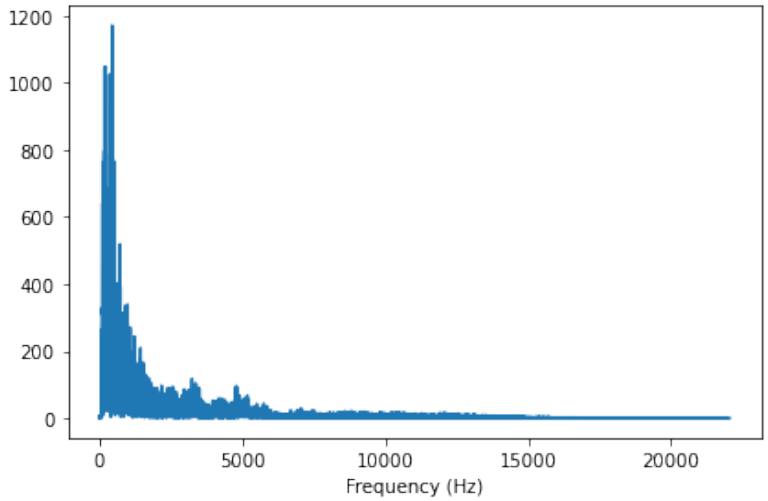

Рис. 2.1: Спектр полезного сигнала

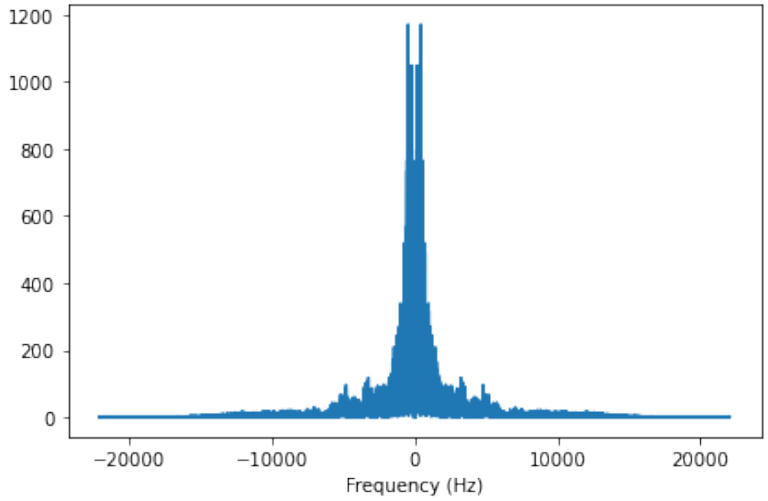

Рис. 2.2: Полный спектр с отрицательными частотами

```
    from thinkdsp import CosSignal

    carrier_sig = CosSignal(freq = 10000)
    carrier_wave = carrier_sig.make_wave(duration=wave.
duration, framerate=wave.framerate)
    modulated = wave * carrier_wave
    modulated.make_spectrum(full = True).plot()

```

Листинг 2.2: Модуляция сигнала косинусоидой

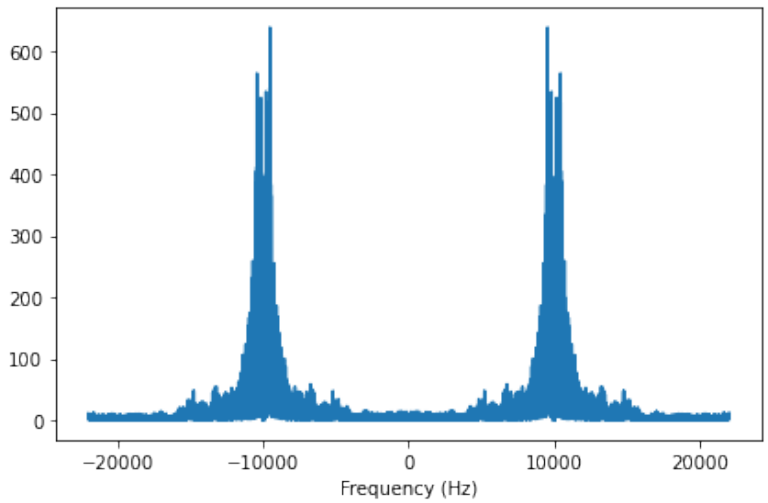

Рис. 2.3: Спектр модулированного сигнала

На слух полученный сигнал не очень приятный, потому что умножение во временной области соответствует свертке в частотной области. AM имеет эффект разделения спектра на 2 половины и сдвига частот на 10 кГц.

```
      demodulated = modulated * carrier_wave
      demodulated_spectrum = demodulated.make_spectrum(full =
    True)
      demodulated_spectrum.plot()
      wave.plot()
      demodulated.plot()

```

Листинг 2.3: Демодуляция сигнала

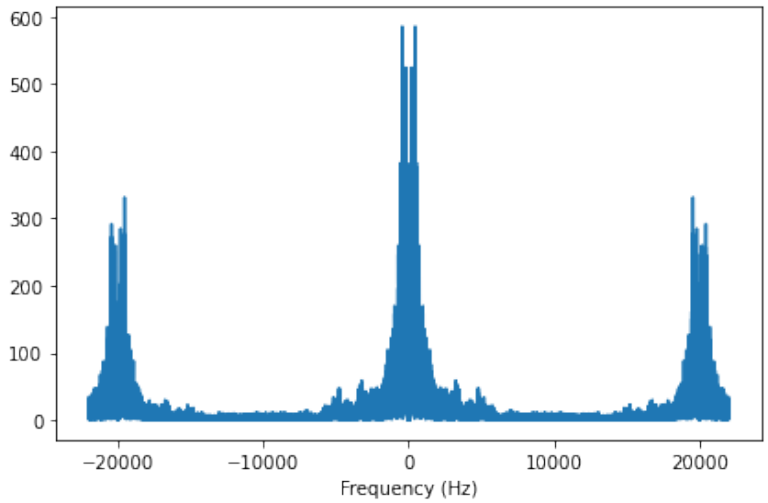

Рис. 2.4: Результат демодуляции сигнала

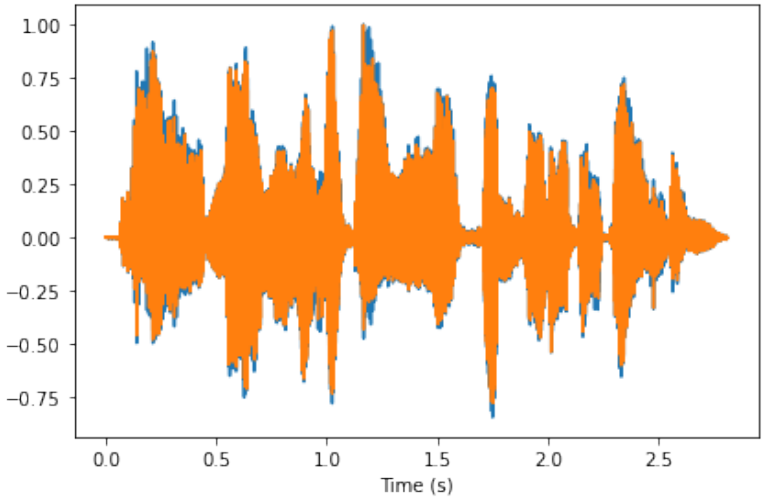

Рис. 2.5: Сравнение исходного и демодулированного сигнала

На слух демодулированный сигнал не отличается от исходного, но на графике видны небольшие различия.

```
demodulated_spectrum.low_pass(10000)
demodulated_spectrum.plot()
filtered = demodulated_spectrum.make_wave()

```

Листинг 2.4: Применим ФНЧ

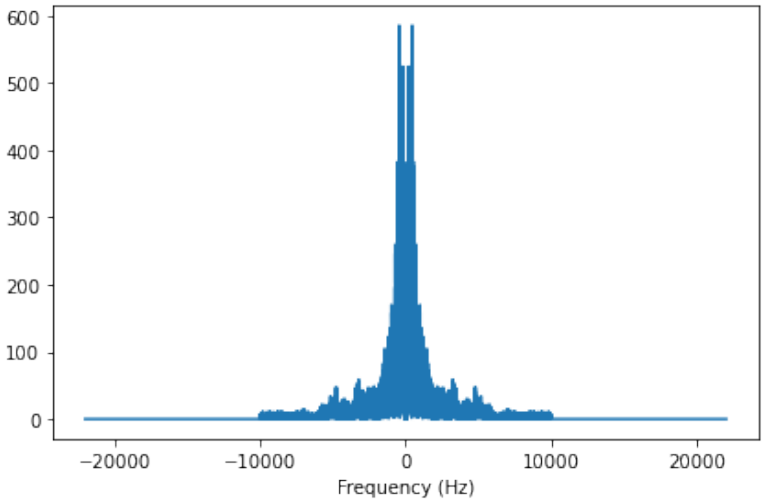

Рис. 2.6: Спектр демодулированного сигнала после применения ФНЧ

```
    carrier_spectrum = carrier_wave.make_spectrum(full =
True)
    carrier_spectrum.plot()

```

Листинг 2.5: Спектр косинусоиды

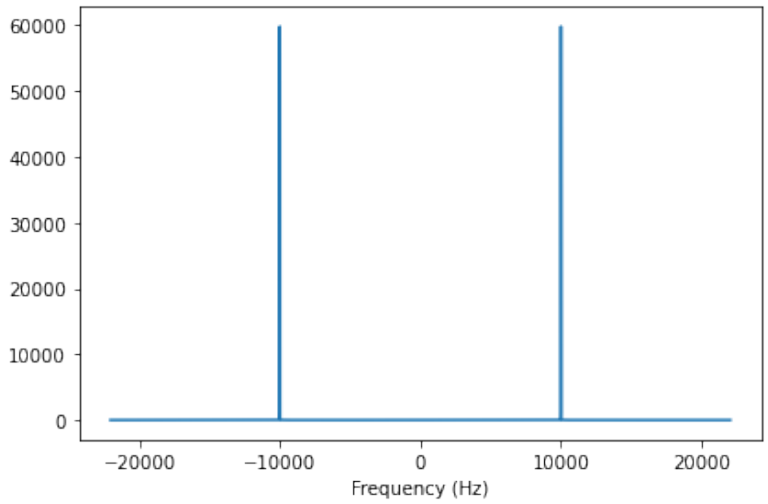

Рис. 2.7: Спектр косинусоиды - два импульса

```
    convolved = spectrum.convolve(carrier_spectrum)
    convolved.plot()
    reconvolved = convolved.convolve(carrier_spectrum)
    reconvolved.plot()

```

Листинг 2.6: Два раза применим свертку сигнала с импульсами

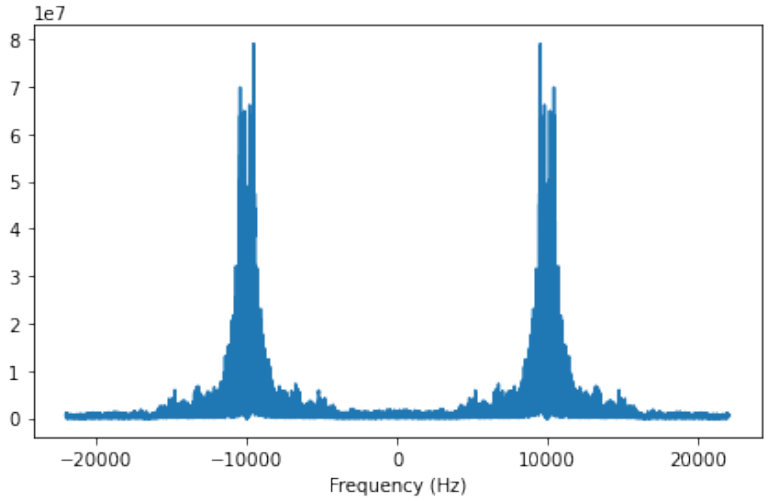

Рис. 2.8: Первая свертка - 2 копии сигнала

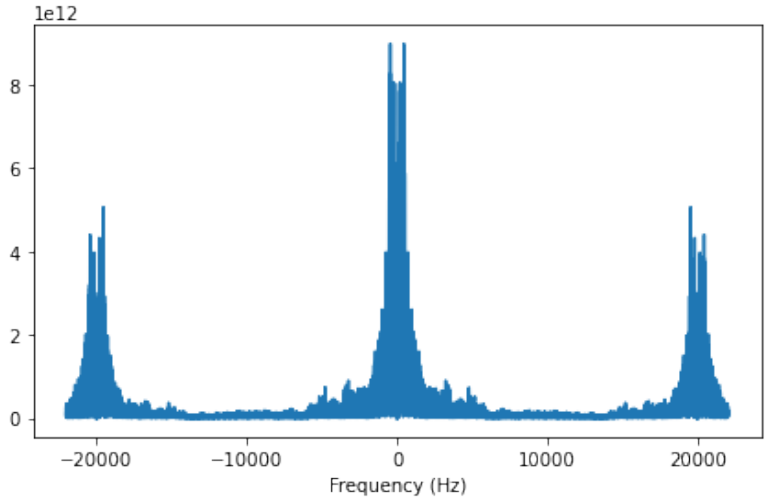

Рис. 2.9: Вторая свертка - 4 копии сигнала

АМ присходит с помощью свертки полезного сигнала с пачкой импульсов «несущего» сигнала.

Глава 3

Выборка

Выборка - процесс измерения аналогового сигнала в серии моментов времени (обычно через равные промежутки).

```
1       wave = read_wave('263868__kevcio__amen-break-a-160-bpm.
    wav')
2       wave.normalize()
3       wave.plot()
4       wave.make_spectrum(full = True).plot()
5
```

Листинг 3.1: Возьмем интересный сигнал

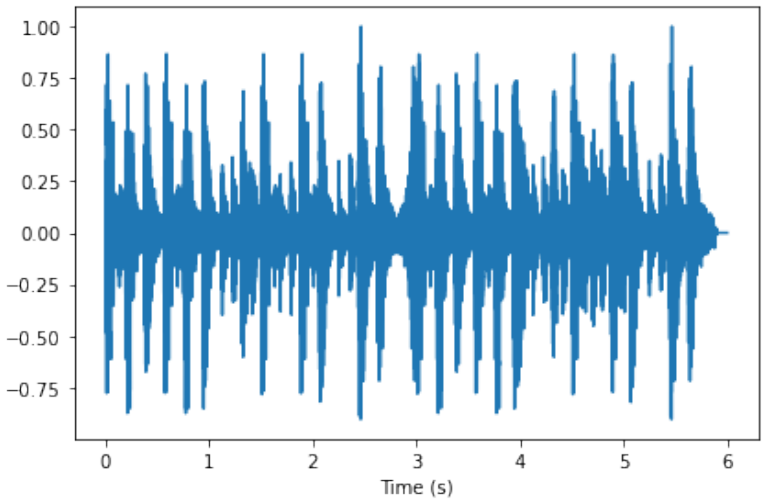

Рис. 3.1: Достаточно интересный сигнал

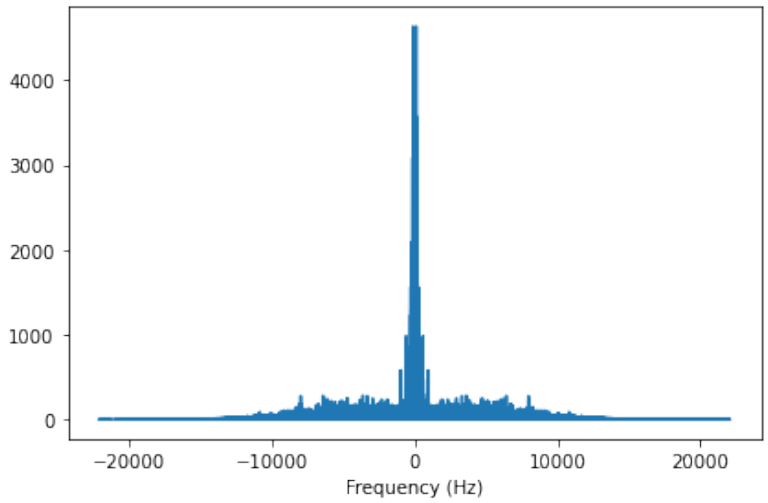

Рис. 3.2: Спектр этого сигнала

```
from thinkdsp import Wave

def sample(wave, factor):
    ys = np.zeros(len(wave))
    ys[::factor] = wave.ys[::factor]
    return Wave(ys, framerate = wave.framerate)

sampled = sample(wave, 4)
sampled.make_spectrum(full = True).plot()

```

Листинг 3.2: Функция для выборки

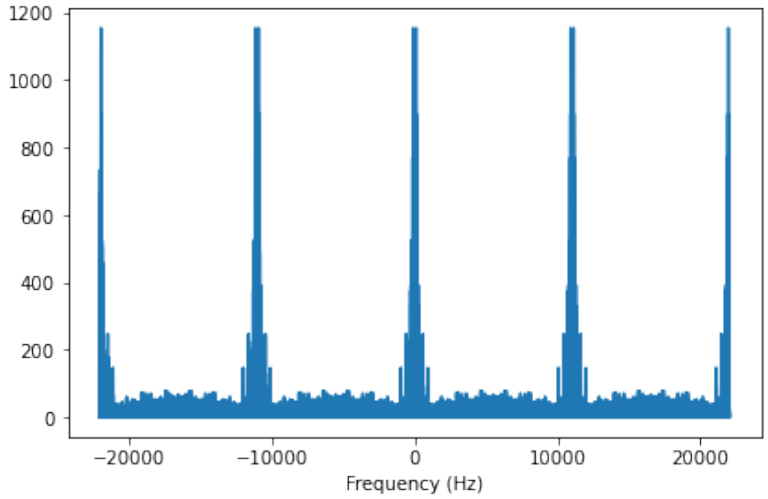

Рис. 3.3: Спектр сигнала после выборки

Результат звучит не очень хорошо, из-за дополнительных компонентов. Попробуем выяснить, откуда они появляются.

```
    def make_impulses(wave, factor):
        ys = np.zeros(len(wave))
        ys[::factor] = 1
        ts = np.arange(len(wave)) / wave.framerate
        return Wave(ys, ts, wave.framerate)

    impulses = make_impulses(wave, 4)
    sampled = wave * impulses
    impulses.make_spectrum(full = True).plot()

```

Листинг 3.3: Функция для создания явных импульсов

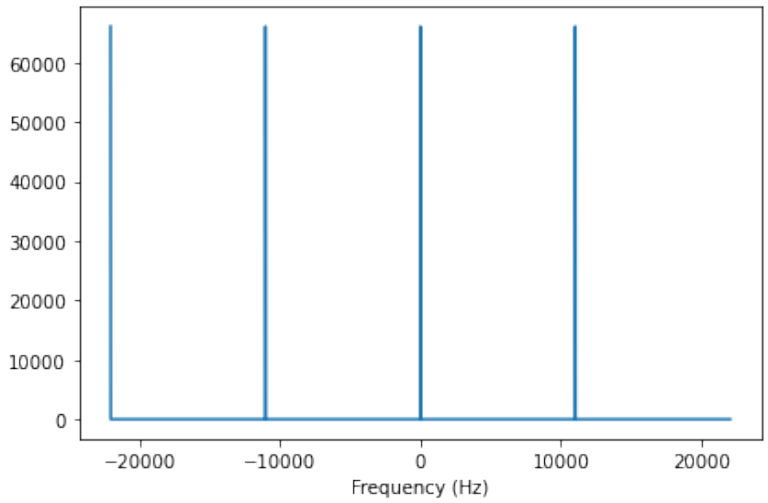

Рис. 3.4: Спектр сигнала после умножения на импульсы

Умножение на импульсы создает 4 сдвинутые копии исходного спектра. При умножении на пачку импульсов происходит свертка с ДПФ пачки импульсов.

```
1       def show_impulses(wave, factor):
2           impulses = make_impulses(wave, factor)
3           plt.subplot(1, 2, 1)
4           impulses.segment(0, 0.001).plot_vlines(linewidth=2)
5           plt.subplot(1, 2, 2)
6           impulses.make_spectrum(full = True).plot()
7
8       slider = widgets.IntSlider(min=2, max=32, value=4)
9       interact(show_impulses, wave = fixed(wave), factor =
    slider);
10
```

Листинг 3.4: Изменение количества импульсов

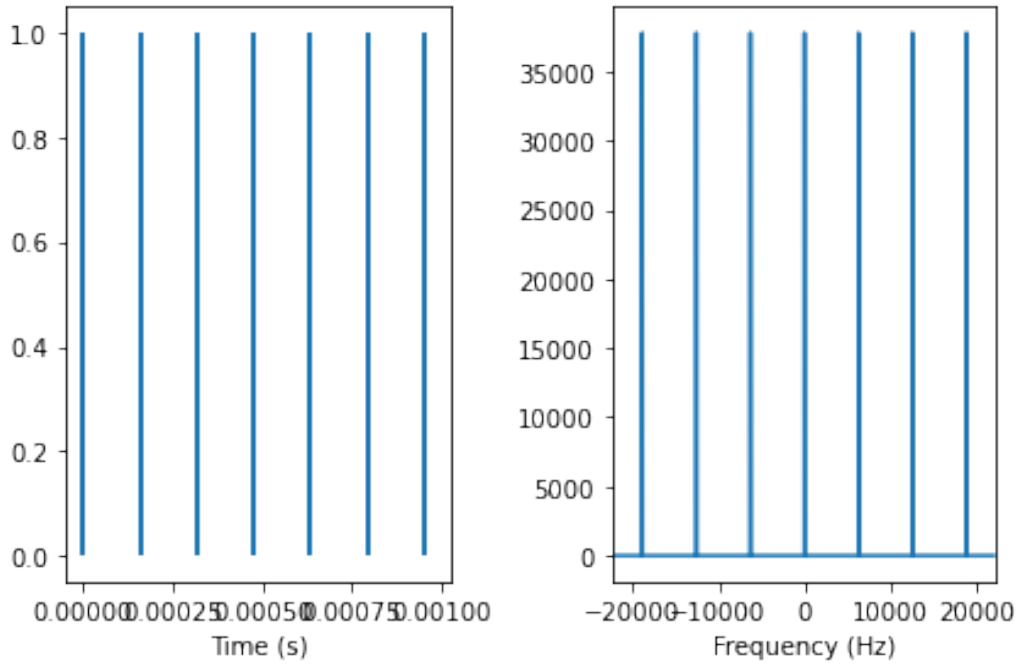

Рис. 3.5: Зависимость формы спектра от количества импульсов

Чем меньше количество импульсов, тем дальше они удаляются друг от друга во временной области и тем ближе копии спектра сближаются в частотной области.

```
spectrum = sampled.make_spectrum(full = True)
spectrum.low_pass(5512.5)
spectrum.plot()
filtered = spectrum.make_wave()

```

Листинг 3.5: Применим ФНЧ

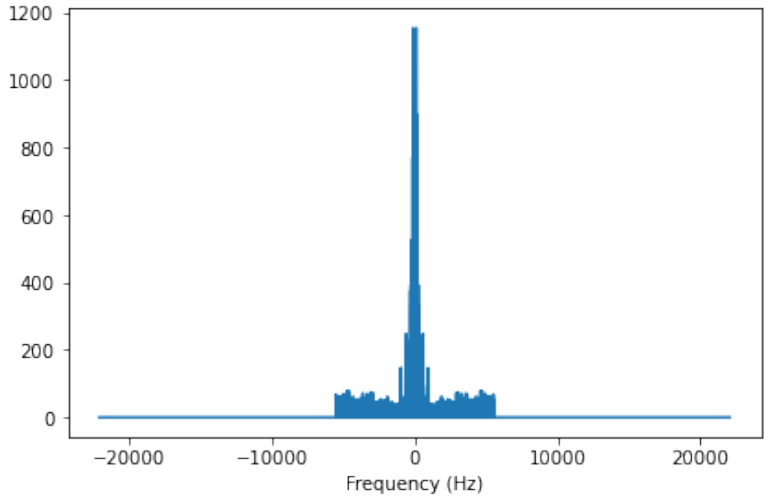

Рис. 3.6: Спектр после применения ФНЧ

Результат звучит совсем не так, как оригинал, из-за дополнительных копий в спектре.

```
1       def plot_segments(original, filtered):
2           start = 1
3           duration = 0.01
4           original.segment(start = start, duration = duration)
    .plot(color = 'gray')
5           filtered.segment(start = start, duration = duration)
    .plot()
6
7       plot_segments(wave, filtered)
8
```

Листинг 3.6: Сравнение исходного и полученного сигнала

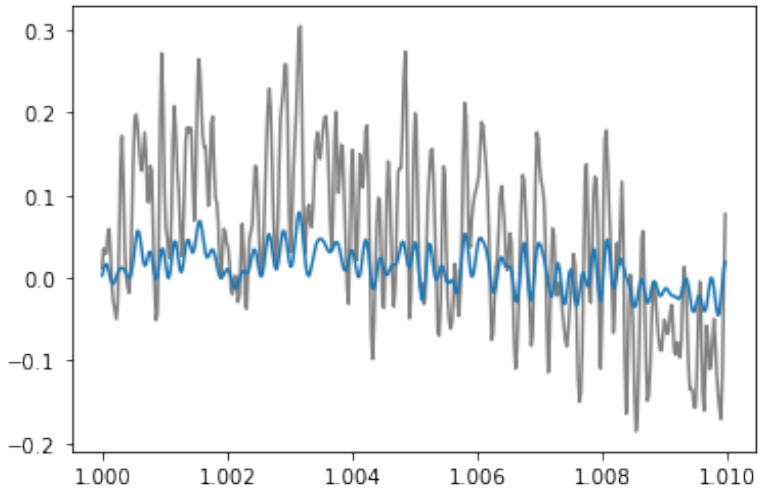

Рис. 3.7: Они явно не совпадают

```
    wave = read_wave('328878__tzurkan__guitar-phrase-tzu.wav')
    wave.normalize()
    wave.plot()
    wave.make_spectrum(full = True).plot()

```

Листинг 3.7: Возьмем другой сигнал

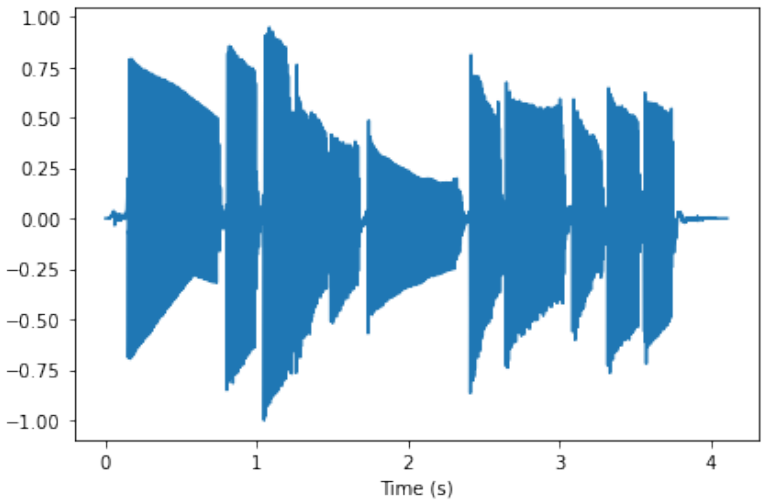

Рис. 3.8: Сигнал игры на бас-гитаре

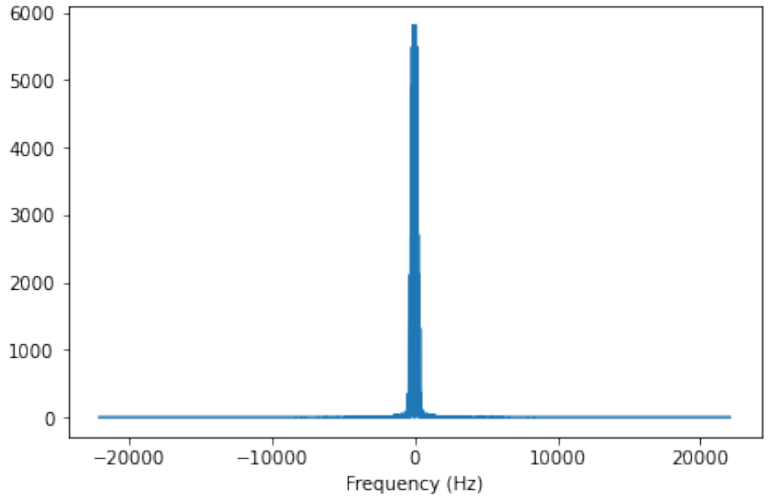

Рис. 3.9: Спектр этого сигнала

```
sampled = sample(wave, 4)
sampled.make_spectrum(full = True).plot()

```

Листинг 3.8: Сделаем выборку сигнала

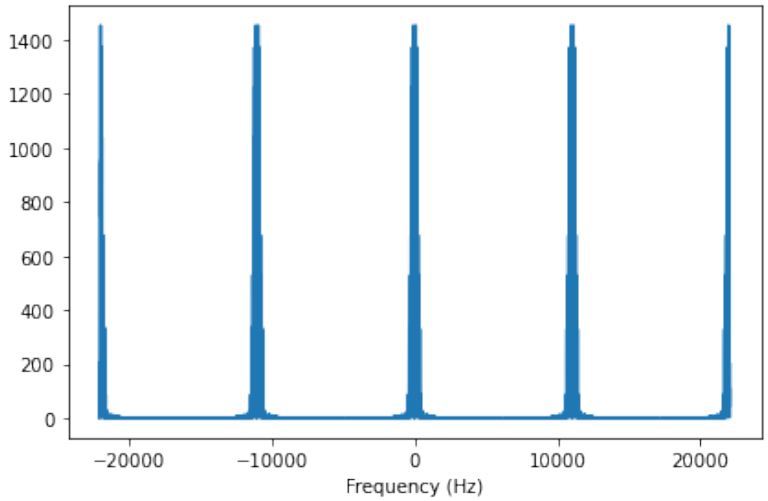

Рис. 3.10: Спектр сигнала после выборки

```
from thinkdsp import Spectrum

def make_boxcar(spectrum, factor):
```

```
        fs = np.copy(spectrum.fs)
        hs = np.zeros_like(spectrum.hs)
        cutoff = (spectrum.framerate / 2) / factor
        for i, f in enumerate(fs):
            if abs(f) <= cutoff: hs[i] = 1
        return Spectrum(hs, fs, spectrum.framerate, full =
spectrum.full)

    spectrum = sampled.make_spectrum(full = True)
    boxcar = make_boxcar(spectrum, 4)
    boxcar.plot()

```

Листинг 3.9: Функция для построения прямоугольного фильтра

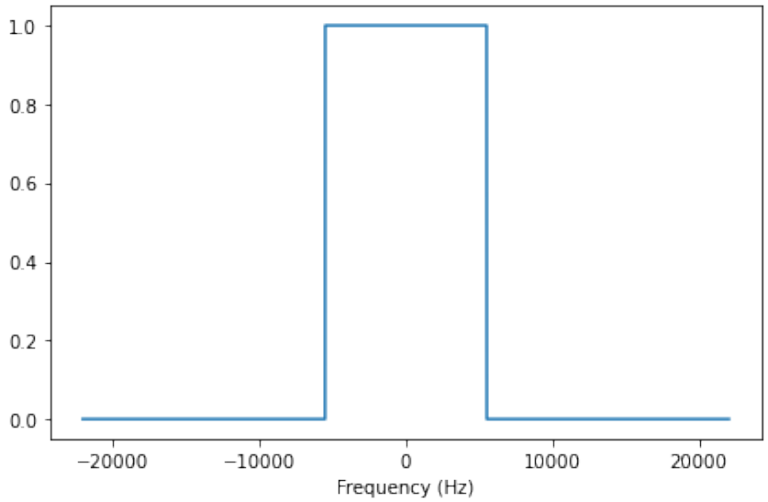

Рис. 3.11: Спектр прямоугольного фильтра

```
    filtered = (spectrum * boxcar).make_wave()
    filtered.scale(4)
    filtered.make_spectrum(full = True).plot()
    plot_segments(wave, filtered)
    diff = wave.ys - filtered.ys
    plt.plot(diff.real)

```

Листинг 3.10: Применим прямоугольный фильтр и результат сравним с исходным сигналом

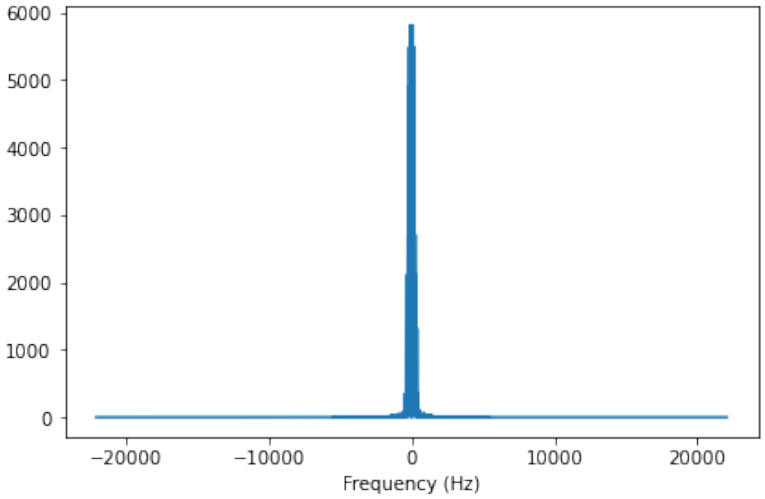

Рис. 3.12: Спектр отфильтрованного сигнала

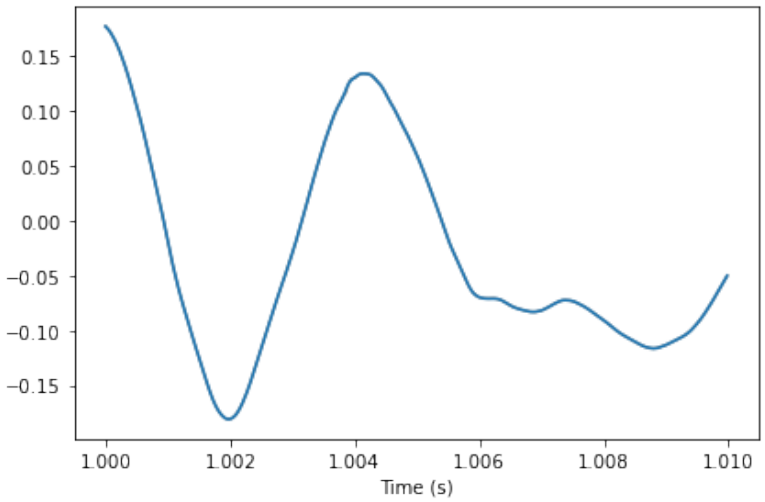

Рис. 3.13: Сегмент отфильтрованного сигнала

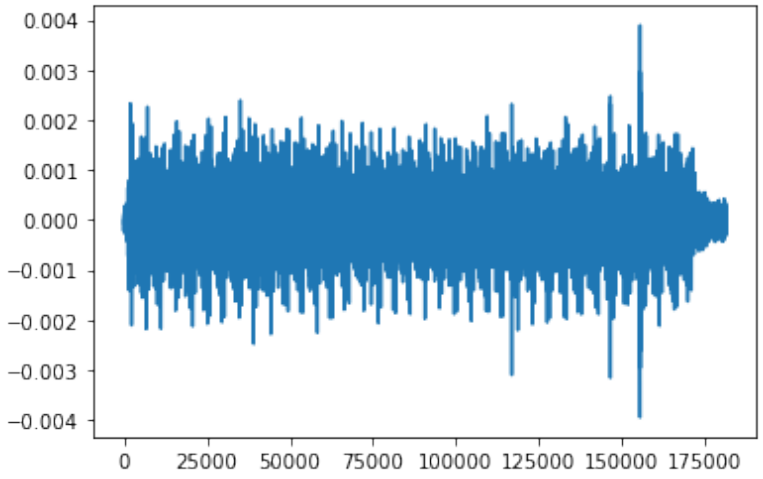

Рис. 3.14: Сравнение с исходным сигналом

Полученный сигнал не полностью идентичен исходному, потому что исходный сигнал имел некоторые компоненты на высоких частотах, но ими можно пренебречь.

```
sinc = boxcar.make_wave()
ys = np.roll(sinc.ys, 50)
plt.plot(ys[:100].real)

```

Листинг 3.11: Окно свертки

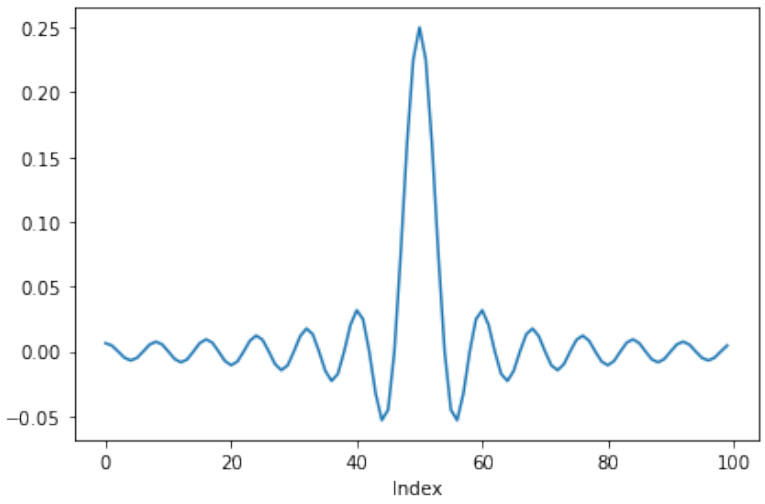

Рис. 3.15: Так выглядит окно свертки

Глава 4

Интерполяция

Здесь мы узучаем `sinc`-функцию.

```
start = 1.0
duration = 0.01
factor = 8
short = wave.segment(start=start, duration=duration)
short.plot()
sampled = sample(short, factor)
sampled.plot_vlines(color = 'gray')

```

Листинг 4.1: Увеличим масштаб

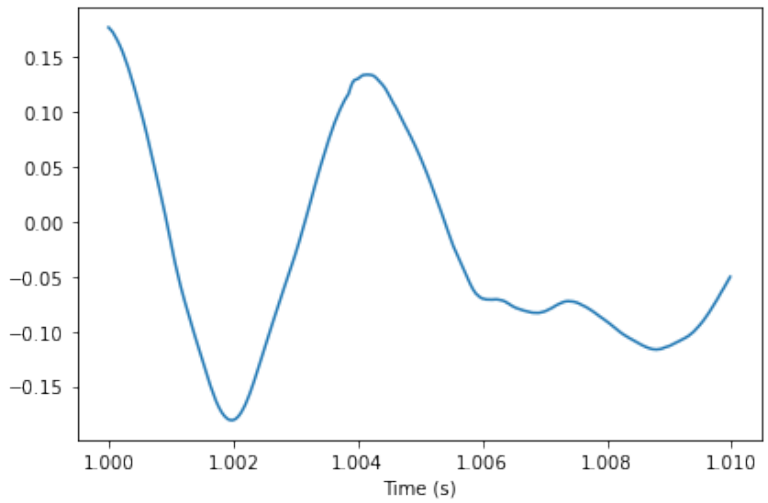

Рис. 4.1: Уменьшенный сегмент сигнала

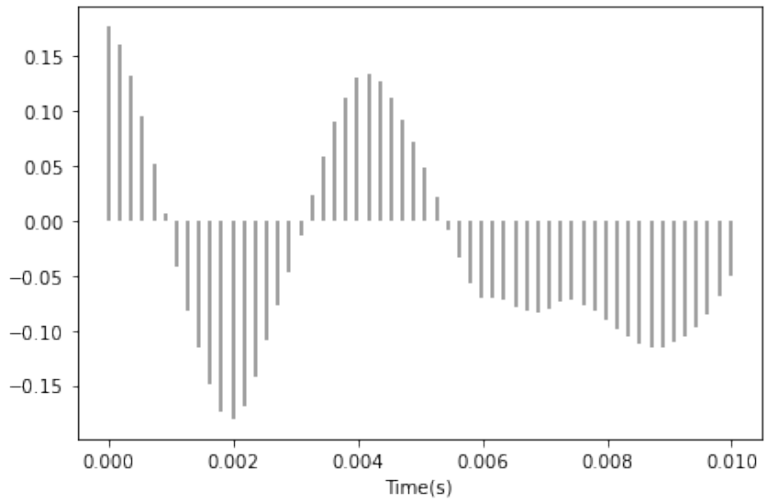

Рис. 4.2: Выборки сигнала

```
spectrum = sampled.make_spectrum()
boxcar = make_boxcar(spectrum, factor)
sinc = boxcar.make_wave()
sinc.shift(sampled.ts[0])
sinc.roll(len(sinc) // 2)
sinc.plot()

```

Листинг 4.2: Изменение положения sinc-функции

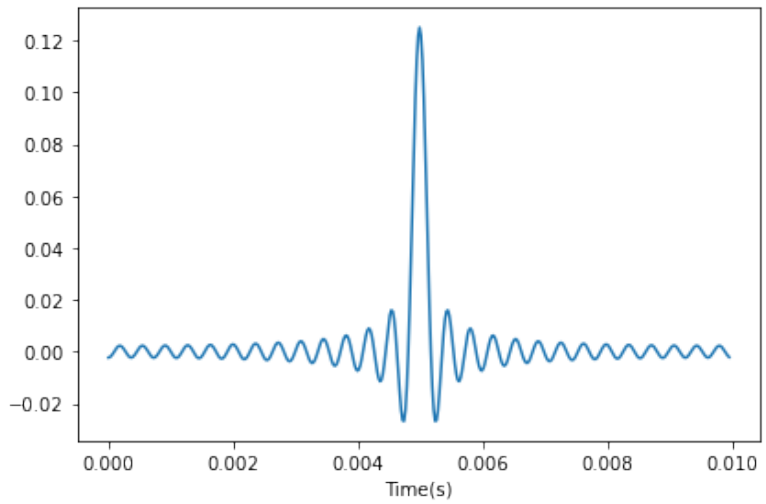

Рис. 4.3: sinc-функция расположена посередине

```
    def plot_sinc_demo(wave, factor, start = None, duration = None):

        def make_sinc(t, i, y):
            sinc = boxcar.make_wave()
            sinc.shift(t)
            sinc.roll(i)
            sinc.scale(y * factor)
            return sinc

        def plot_sincs(wave):
            t0 = wave.ts[0]
            for i in range(0, len(wave), factor):
                sinc = make_sinc(t0, i, wave.ys[i])
                seg = sinc.segment(start, duration)
                seg.plot(color = 'green', linewidth = 0.5, alpha = 0.3)
                if i == 0: total = sinc
                else: total += sinc
            seg = total.segment(start, duration)
            seg.plot(color = 'blue', alpha = 0.5)

        sampled = sample(wave, factor)
        spectrum = sampled.make_spectrum()
        boxcar = make_boxcar(spectrum, factor)
        start = wave.start if start is None else start
        duration = wave.duration if duration is None else duration
        sampled.segment(start, duration).plot_vlines(color = 'gray')
        wave.segment(start, duration).plot(color = 'gray')
        plot_sincs(wave)

    plot_sinc_demo(short, 4)

```

Листинг 4.3: Исследование sinc-функции

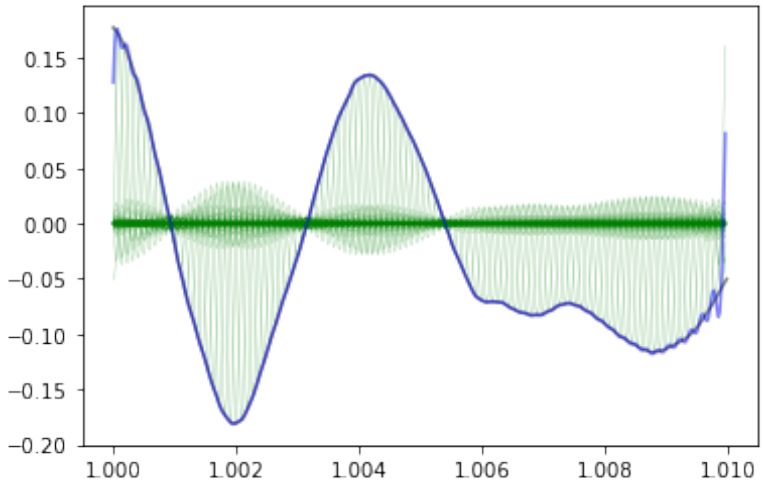

Рис. 4.4: Результат исследования работы sinc-функции

Тонкие зеленые линии - это сдвинутые, масштабированные копии sinc-функции. Синяя линия - это сумма sinc-функций. Серая линия - это исходная функция. Сумма `sinc`-функций интерполируется между образцами и восстанавливает исходную волну.

```
1       start = short.start + 0.004
2       duration = 0.00061
3       plot_sinc_demo(short, 4, start, duration)
4       decorate(xlim = [start, (start + duration)], ylim =
    [-0.05, 0.17])
5
```

Листинг 4.4: Рассмотрим результат внимательнее

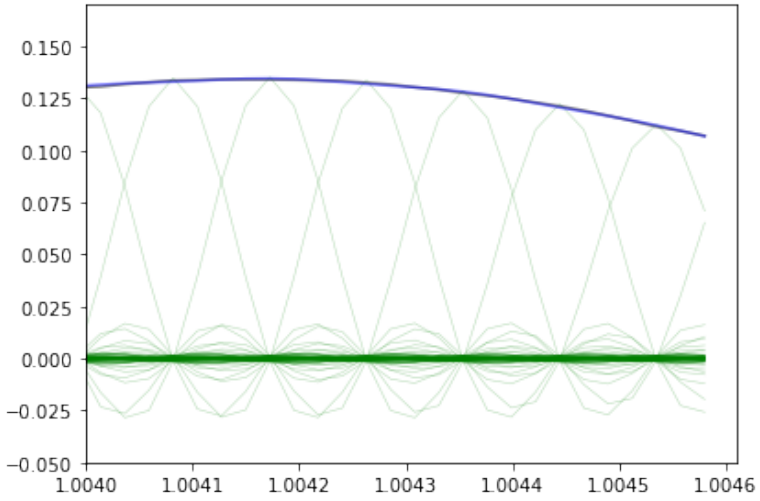

Рис. 4.5: Крупный план

Вертикальные серые линии - это выборки. Тонкие зеленые линии - это сдвинутые, масштабированные копии `sinc`-функции. В этом сегменте интерполяция очень хорошо соответствует исходной волне. Сумма `sinc`-функций идентична оригинальному сигналу.

Глава 5

Упражнения

5.1 Задание 3

Фильтрация частоты до выборки - фильтр сглаживания.

```
1       wave = read_wave('263868__kevcio__amen-break-a-160-bpm.
    wav')
2       wave.normalize()
3       wave.plot()
4       spectrum = wave.make_spectrum(full = True)
5       spectrum.plot()
6
```

Листинг 5.1: Соло на барабане

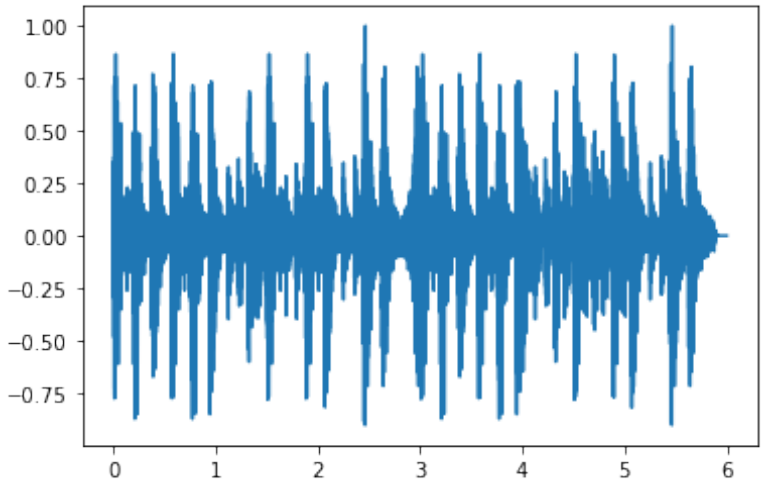

Рис. 5.1: График сигнала

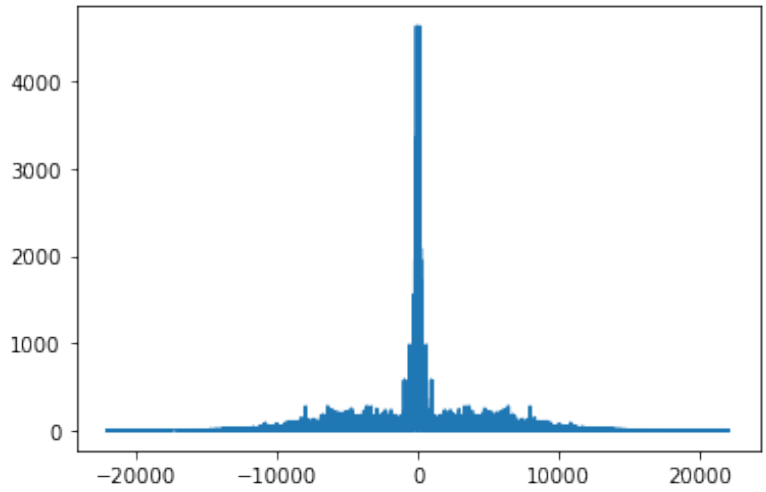

Рис. 5.2: И его спектр

```
framerate = wave.framerate / 5
cutoff = framerate / 2 - 1
spectrum.low_pass(cutoff)
spectrum.plot()

```

Листинг 5.2: Фильтр сглаживания

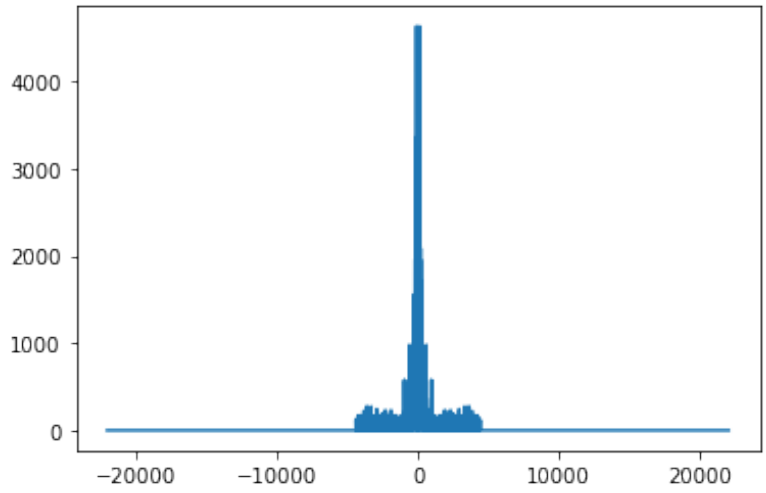

Рис. 5.3: Результат работы фильтра

После фильтрации сигнал все еще довольно хорошо звучит.

```
filtered = spectrum.make_wave()
sampled = sample(filtered, 5)
sampled_spectrum = sampled.make_spectrum(full = True)
sampled_spectrum.plot()

```

Листинг 5.3: Выборка

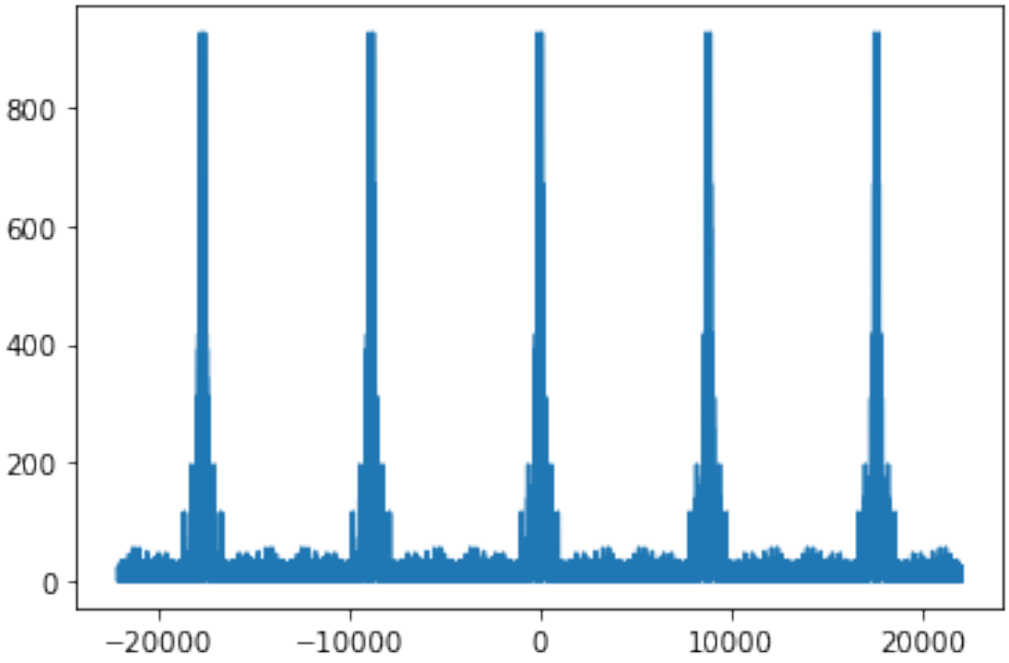

Рис. 5.4: Спектр сигнала после выборки

```
sampled_spectrum.low_pass(cutoff)
sampled_spectrum.plot()
sampled_spectrum.scale(5)
spectrum.plot()
sampled_spectrum.plot()

```

Листинг 5.4: Строим спектр полученного сигнала

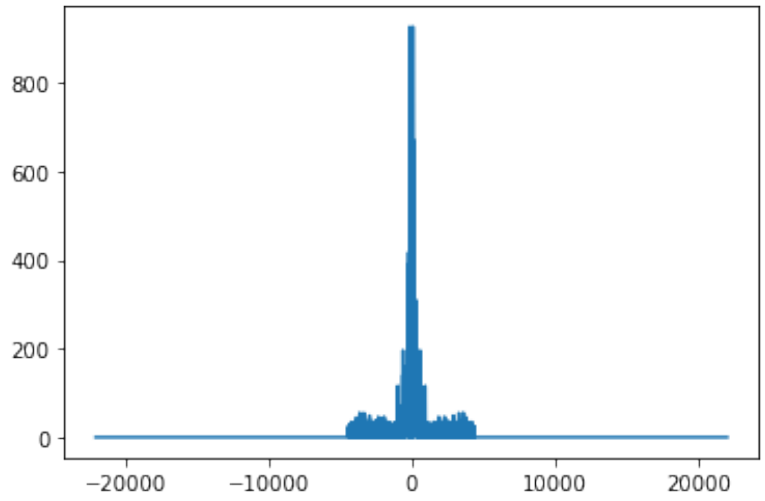

Рис. 5.5: Спектр полученного сигнала

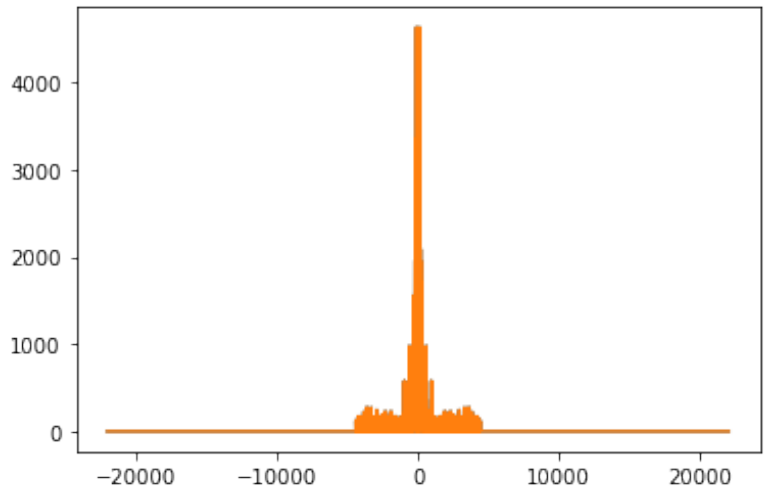

Рис. 5.6: Сравнение полученного спектра с исходным спектром

Разница между ними близка к 0.

```
    interpolated = sampled_spectrum.make_wave()
    filtered.plot()
    interpolated.plot()

```

Листинг 5.5: Разница между интерполяцией и фильтрацией сигнала

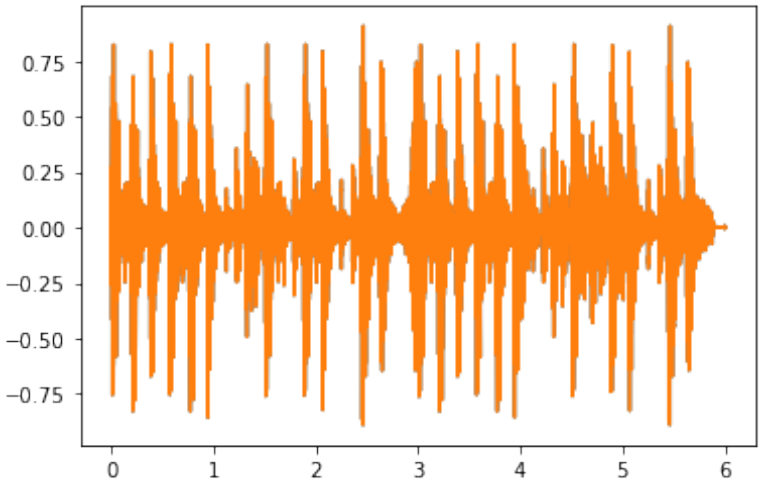

Рис. 5.7: Разница почти равняется нулю

Глава 6

Вывод

В данной работе мы познакомились с работой ампдитудной модуляции, узнали, как влияет количество выборок на качество сигнала, и немного изучили алгоритм `sinc`-функции.