Лабораторная работа №5
Автокорреляция

Кобыжев Александр

8 апреля 2021 г.

Оглавление

Список иллюстраций

Листинги

Глава 1

Упражнение 5.1

Позаимствуем функции из блокнота `chap05.ipynb` для выполнения данного пункта лабораторной работы.

```
def serial_corr(wave, lag=1):
    N = len(wave)
    y1 = wave.ys[lag:]
    y2 = wave.ys[:N-lag]
    corr = np.corrcoef(y1, y2, ddof=0)[0, 1]
    return corr
```

Листинг 1.1: Функция serial_corr

```
def autocorr(wave):
    lags = range(len(wave.ys)//2)
    corrs = [serial_corr(wave, lag) for lag in lags]
    return lags, corrs
```

Листинг 1.2: Функция `autocorr`

Загрузим вокальный чирп.

```
wave = thinkdsp.read_wave('28042__bcjordan__voicedownbew.wav')
wave.normalize()
wave.make_audio()
```

Листинг 1.3: Вокальный чирп

Возьмём короткий сегмент сигнала через 0.5 секунды после начала и длительностью 0.01 секунды:

```
segment = wave.segment(start=0.5, duration=0.01)
```

Листинг 1.4: Первый сегмент

Применим автокорреляционную функцию, чтобы оценить высоту тона:

```
lags, corrs = autocorr(segment)
thinkplot.plot(lags, corrs)
thinkplot.config(xlabel='Lag (index)', ylabel='Correlation',
    ylim=[-1, 1])
```

Листинг 1.5: Оценка высоты тона

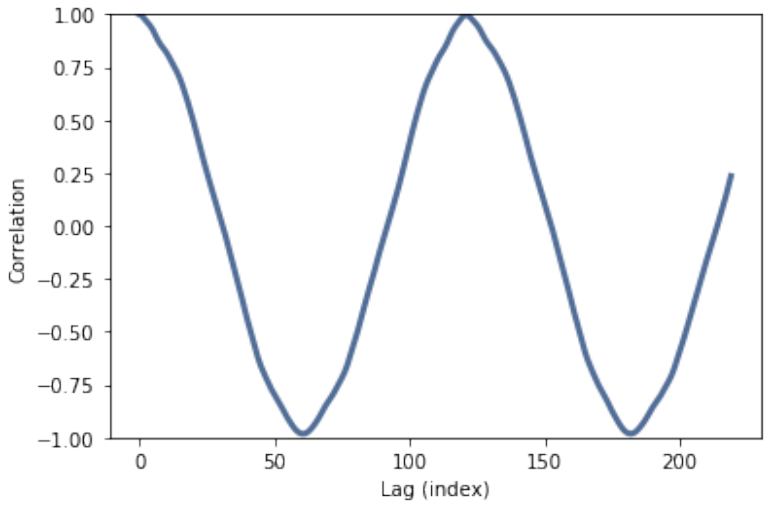

Рис. 1.1: Высота тона

Пик находится между 100 и 150. Используем `argmax`, чтобы уточнить значение `lag` для этого пика:

```
low, high = 100, 150
lag = np.array(corrs[low:high]).argmax() + low
lag
```

Листинг 1.6: Нахождение `lag`

Находим соответствующую частоту для `lag` = 121:

```
period = lag / segment.framerate
frequency = 1 / period
frequency
```

Листинг 1.7: Нахождение частоты

Частота равняется `364.4628099173554`.

Теперь рассмотрим сегмент сигнала через 1 секунду и проделаем аналогичные действия.

```
segment = wave.segment(start=1.0, duration=0.01)
```

Листинг 1.8: Второй сегмент

Применим автокорреляционную функцию, чтобы оценить высоту тона:

```
lags, corrs = autocorr(segment)
thinkplot.plot(lags, corrs)
thinkplot.config(xlabel='Lag (index)', ylabel='Correlation',
    ylim=[-1, 1])
```

Листинг 1.9: Оценка высоты тона

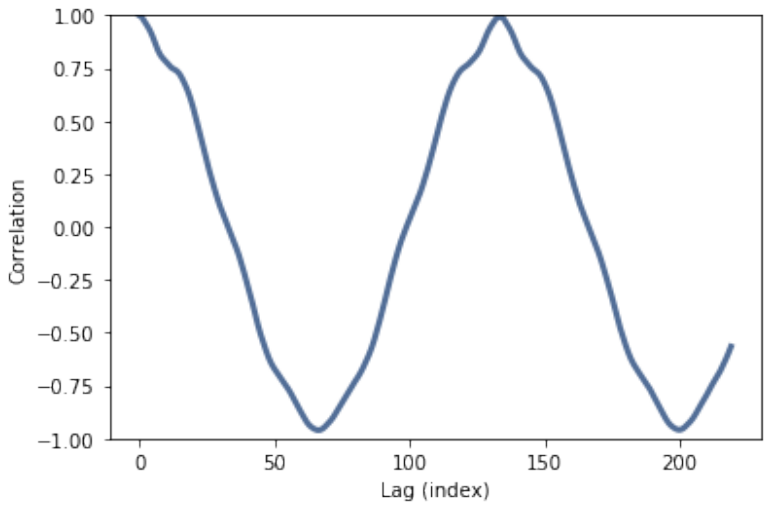

Рис. 1.2: Высота тона

Пик снова находится между 100 и 150. Используем `argmax`, чтобы уточнить значение `lag` для этого пика:

```
low, high = 100, 150
lag = np.array(corrs[low:high]).argmax() + low
lag
```

Листинг 1.10: Нахождение `lag`

Находим соответствующую частоту для `lag` = 134:

```
period = lag / segment.framerate
```

```
2 frequency = 1 / period
3 frequency
```

Листинг 1.11: Нахождение частоты

Частота равняется `329.1044776119403`. Отсюда можно сделать вывод, что основная частота ожидаемо уменьшается при увеличении времени начала сегмента.

Глава 2

Упражнение 5.2

Загрузим тот же вокальный чирп.

```
wave = thinkdsp.read_wave('28042__bcjordan__voicedownbew.wav')
wave.normalize()
wave.make_audio()
```

Листинг 2.1: Загрузка звука

Воспользуемся примером кода из `chap05.ipynb`. Вот его спектрограмма:

```
wave.make_spectrogram(2048).plot(high=4200)
thinkplot.config(xlabel='Time (s)',
                 ylabel='Frequency (Hz)',
                 xlim=[0, 1.4],
                 ylim=[0, 4200])
```

Листинг 2.2: Спектрограмма звука

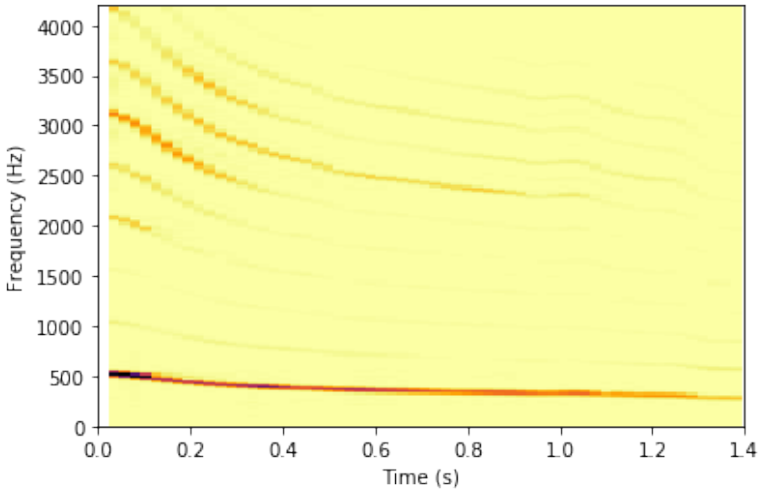

Рис. 2.1: Сегменты звуков

Инкапсулируем предлагаемый код в функцию. Найти первый самый высокий пик в автокорреляционной функции сложно. Поэтому мы просто укажем диапазон `lag` для поиска.

```
from autocorr import autocorr
def estimate_fundamental(segment, low=70, high=150):
    lags, corrs = autocorr(segment)
    lag = np.array(corrs[low:high]).argmax() + low
    period = lag / segment.framerate
    frequency = 1 / period
    return frequency
```

Листинг 2.3: Инкапсуляция функции

Рассмотрим пример работы функции:

```
duration = 0.01
segment = wave.segment(start=0.2, duration=duration)
freq = estimate_fundamental(segment)
freq
```

Листинг 2.4: Пример работы функции

Результатом работы получилась частота `436.63366336633663`.

Используем написанную функцию для отслеживания высоты тона записанного звука. `ts` - средние точки каждого сегмента.

```
step = 0.05
starts = np.arange(0.0, 1.4, step)
```

```
3
4 ts = []
5 freqs = []
6
7 for start in starts:
8     ts.append(start + step/2)
9     segment = wave.segment(start=start, duration=duration)
10    freq = estimate_fundamental(segment)
11    freqs.append(freq)
```

Листинг 2.5: Отслеживание высоты тона

Рассмотрим кривую отслеживания высоты тона, наложенную на спектрограмму:

```
1 wave.make_spectrogram(2048).plot(high=900)
2 thinkplot.plot(ts, freqs, color='green')
3 thinkplot.config(xlabel='Time (s)',
4                   ylabel='Frequency (Hz)',
5                   xlim=[0, 1.4],
6                   ylim=[0, 900])
```

Листинг 2.6: Кривая отслеживания высоты тона на спектрограмме

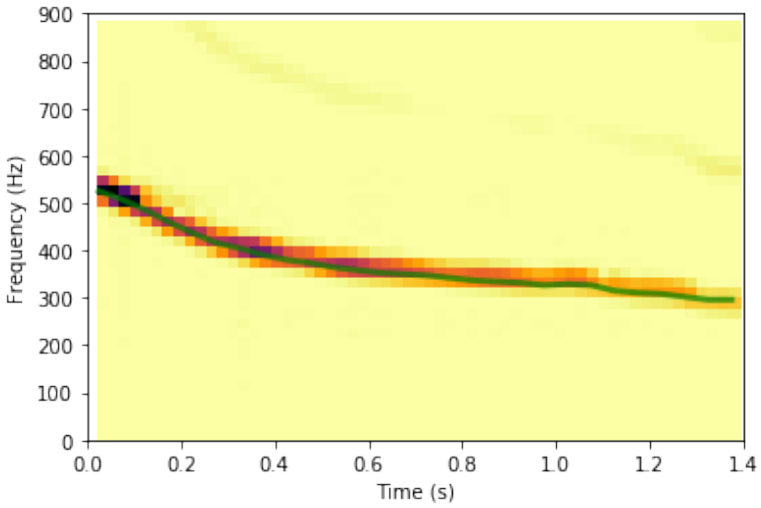

Рис. 2.2: Кривая отслеживания высоты тона на спектрограмме

Наложив оценки высоты тона на спектрограмму записи, можно видеть, что функция полностью справляется со своей задачей.

Глава 3

Упражнение 5.3

Воспользуемся данными о ежедневной цене `BitCoin` в течение года из прошлой лабораторной работы.

```
data = pd.read_csv('BTC_USD_2020-04-08_2021-04-07-CoinDesk.csv')
data
```

Листинг 3.1: Таблица данных

	Currency	Date	Closing Price (USD)	24h Open (USD)	24h High (USD)	24h Low (USD)
0	BTC	2020-04-08	7175.667477	7277.704282	7464.732245	7081.639209
1	BTC	2020-04-09	7367.293398	7175.669418	7424.743721	7155.211053
2	BTC	2020-04-10	7321.816614	7366.900961	7399.469133	7125.775519
3	BTC	2020-04-11	6866.398189	7321.815746	7325.324778	6752.593664
4	BTC	2020-04-12	6873.848495	6872.137266	6949.788875	6777.889694
...	...	...	...	...	...	...
360	BTC	2021-04-03	58821.626994	58726.084566	60101.752326	58478.598349
361	BTC	2021-04-04	57517.798773	58958.428985	59713.210136	57185.768006
362	BTC	2021-04-05	58177.402764	57134.860051	58540.984706	56552.222275
363	BTC	2021-04-06	58843.559540	58230.675538	59243.036175	56846.969047
364	BTC	2021-04-07	58040.187602	59133.655740	59484.199475	57421.853085

Рис. 3.1: Таблица данных

Визуализируем скачанные данные.

```
wave = thinkdsp.Wave(data['Closing Price (USD)'], data.index,
    framerate=1)
wave.plot()
thinkplot.config(xlabel='Time (days)',
                 ylabel='Price of BitCoin ($)')
```

Листинг 3.2: Визуализация данных

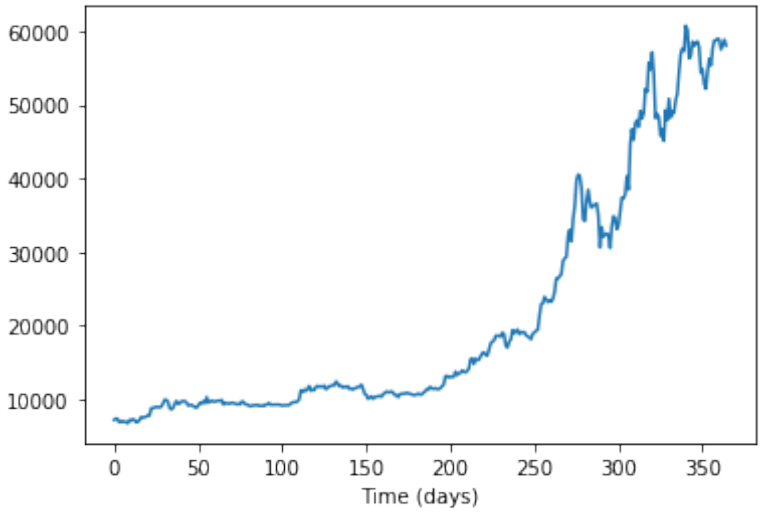

Рис. 3.2: Визуализация данных

Воспользуемся функцией автокорреляции, использующая статистическое определение, то есть она сдвигает среднее значение к нулю, делит на стандартное отклонение и делит сумму на N.

```
from autocorr import autocorr

lags, corrs = autocorr(wave)
thinkplot.plot(lags, corrs)
thinkplot.config(xlabel='Lag',
                 ylabel='Correlation')
```

Листинг 3.3: Автокорреляция при помощи `autocorr`

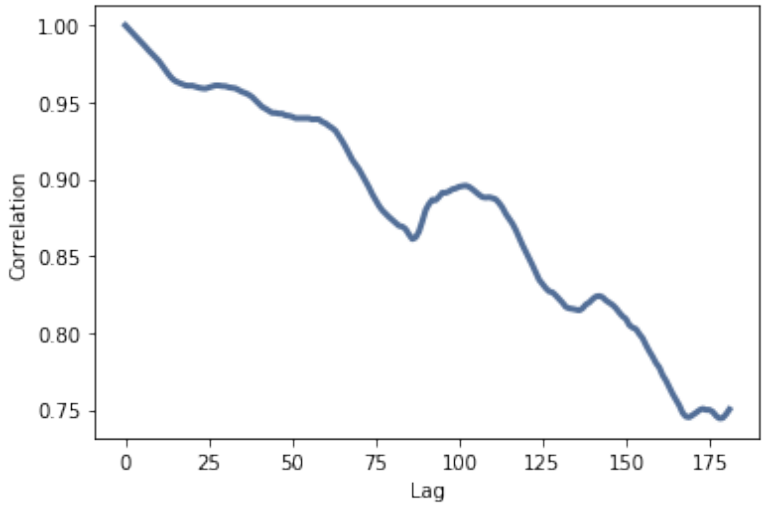

Рис. 3.3: Автокорреляция при помощи `autocorr`

Автокорреляционная функция падает медленно, поскольку задержка увеличивается, предполагая какой-то розовый шум.

Мы можем сравнить реализацию `autocorr` с `np.correlate`, в котором используется определение корреляции, используемое при обработке сигналов.

```
N = len(wave)
corrs2 = np.correlate(wave.ys, wave.ys, mode='same')
lags = np.arange(-N//2, N//2)
thinkplot.plot(lags, corrs2)
thinkplot.config(xlabel='Lag',
                 ylabel='Dot product')
```

Листинг 3.4: Автокорреляция при помощи `np.correlate`

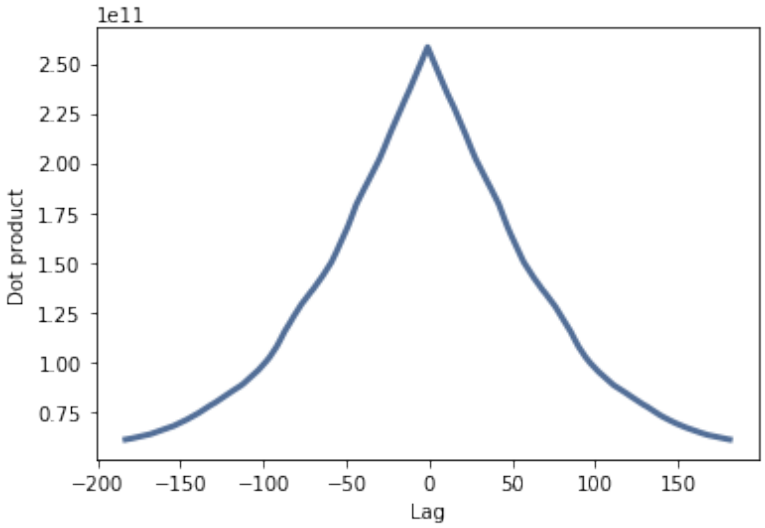

Рис. 3.4: Автокорреляция при помощи `np.correlate`

Результат симметричен, потому что два сигнала идентичны, и отрицательный `lag` у одного даёт такой же эффект, как и положительный `lag` у другого. Вторая половина результата соответствует положительным `lag`:

```
N = len(corrs2)
half = corrs2[N//2:]
thinkplot.plot(half)
thinkplot.config(xlabel='Lag',
                 ylabel='Dot product')
```

Листинг 3.5: Вторая половина результата

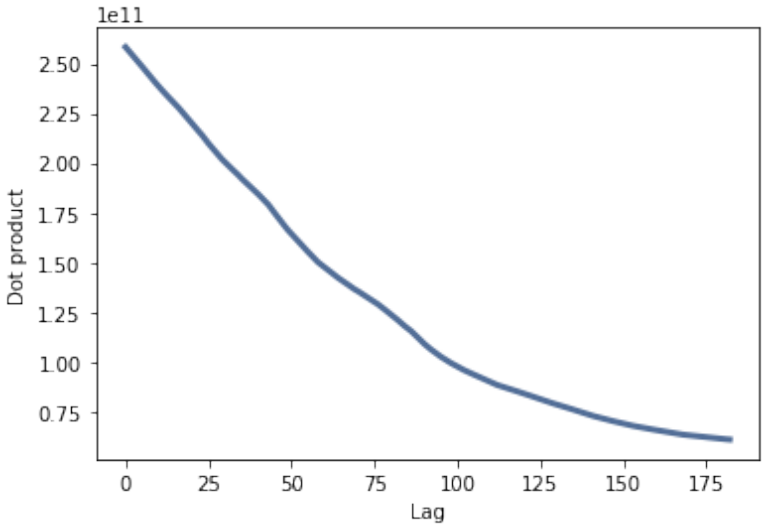

Рис. 3.5: Вторая половина результата

Мы можем нормализовать результаты, разделив их на длины (`lengths`):

```
lengths = range(N, N//2, -1)
half /= lengths
half /= half[0]
thinkplot.plot(half)
thinkplot.config(xlabel='Lag',
                 ylabel='Dot product')
```

Листинг 3.6: Нормализованный результат

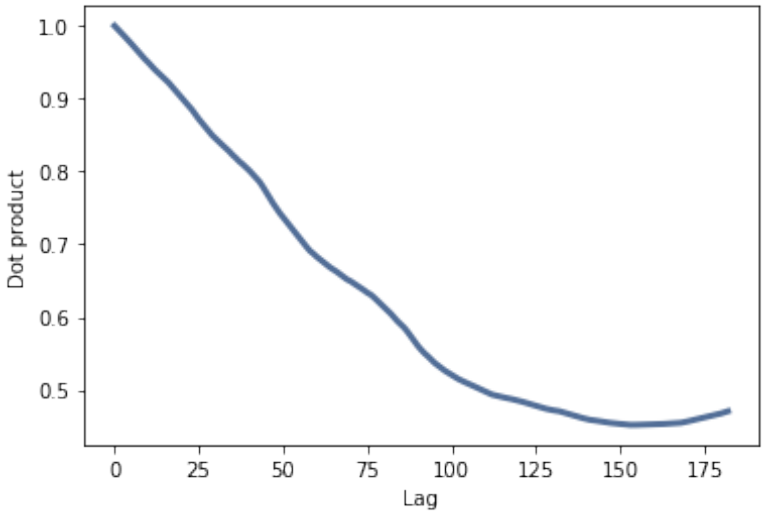

Рис. 3.6: Нормализованный результат

Но даже после нормализации результаты выглядят совершенно по-другому. В результате корреляции пики стираются.

```
thinkplot.preplot(2)
thinkplot.plot(corrs, label='autocorr')
thinkplot.plot(half, label='correlate')
thinkplot.config(xlabel='Lag', ylabel='Correlation')
```

Листинг 3.7: Сравнение результатов

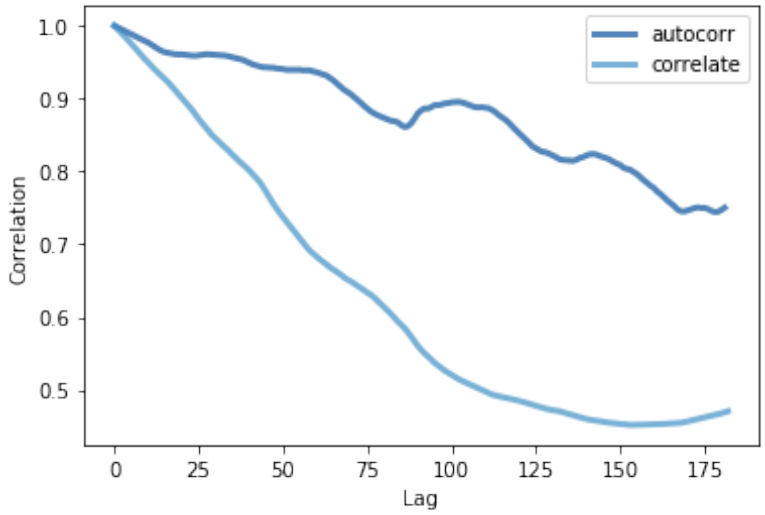

Рис. 3.7: Сравнение результатов

Скорее всего причина различия результатов в том, что данные выглядят очень по-разному в разных частях диапазона, в частности, дисперсия сильно меняется с течением времени.

Для этого набора данных, вероятно, более уместно статистическое определение автокорреляционной функции.

Глава 4

Упражнение 5.4

Для выполнения данного задания я нашёл другой звук саксофона, который можно прослушать по ссылке.

```
wave = thinkdsp.read_wave('77685__juskiddink__anna-britta-sax.wav')
wave.normalize()
wave.make_audio()
```

Листинг 4.1: Загрузка звука

Рассмотрим спектрограмму, которая показывает гармоническую структуру во времени.

```
gram = wave.make_spectrogram(seg_length=1024)
gram.plot(high=3000)
thinkplot.config(xlabel='Time (s)', ylabel='Frequency (Hz)')
```

Листинг 4.2: Спектрограмма звука

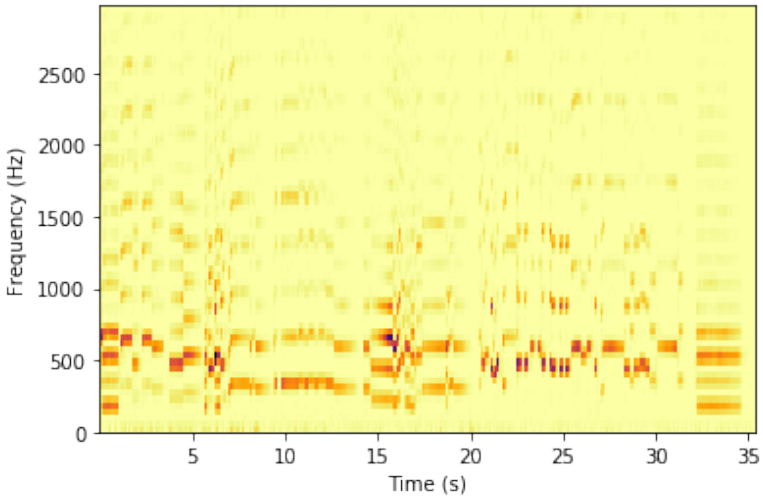

Рис. 4.1: Спектрограмма звука

Чтобы увидеть гармоники более четко, я возьму сегмент от 3 секунд продолжительностью 0.5 секунды и вычислю его спектр.

```
start = 3.0
duration = 0.5
segment = wave.segment(start=start, duration=duration)
segment.make_audio()
```

Листинг 4.3: Создание сегмента звука

```
spectrum = segment.make_spectrum()
spectrum.plot(high=3000)
thinkplot.config(xlabel='Frequency (Hz)', ylabel='Amplitude')
```

Листинг 4.4: Спектр звука

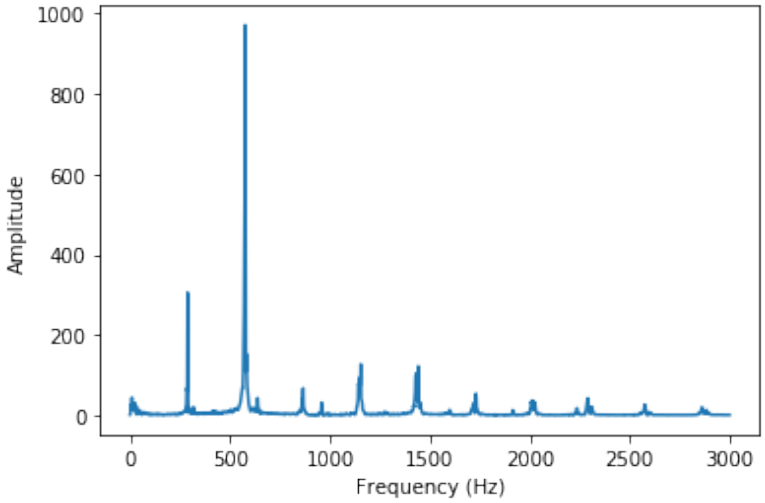

Рис. 4.2: Спектр звука

Пики в спектре находятся на частотах 288, 576 и 1154 Гц.

Высота звука, которую мы воспринимаем, является основной, на частоте 288 Гц, хотя она и не является доминирующей частотой.

Для сравнения, рассмотрим треугольную волну на частоте 288 Гц.

```
thinkdsp.TriangleSignal(freq=288).make_wave(duration=0.5).make_audio()
```

Листинг 4.5: Треугольная волна

```
segment.make_audio()
```

Листинг 4.6: Сегмент звука

У них одинаковая воспринимаемая высота тона. Чтобы понять, почему мы воспринимаем основную частоту, даже если она не является доминирующей, полезно взглянуть на функцию автокорреляции (АКФ). Написанная функция вычисляет АКФ, выбирает вторую половину (которая соответствует положительным `lag`) и нормализует результаты:

```
def autocorr(segment):
    corrs = np.correlate(segment.ys, segment.ys, mode='same')
    N = len(corrs)
    lengths = range(N, N//2, -1)

    half = corrs[N//2:].copy()
    half /= lengths
    half /= half[0]
```

```
    return half
```

Листинг 4.7: Функция `autocorr`

```
corrs = autocorr(segment)
thinkplot.plot(corrs[:200])
thinkplot.config(xlabel='Lag', ylabel='Correlation', ylim=[-1.05,
    1.05])
```

Листинг 4.8: Визуализация АКФ

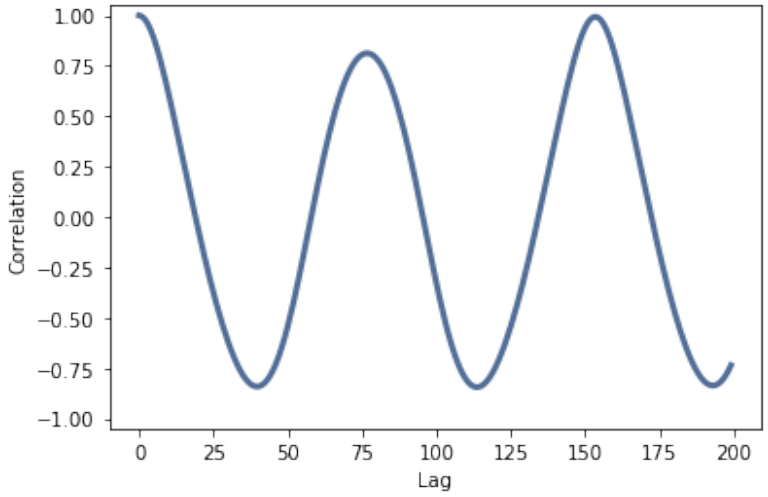

Рис. 4.3: Визуализация АКФ

Первый крупный пик находится вблизи `lag` = 75.

Следующая написанная функция находит самую высокую корреляцию в заданном диапазоне лагов и возвращает соответствующую частоту.

```
def find_frequency(corrs, low, high):
    lag = np.array(corrs[low:high]).argmax() + low
    print(lag)
    period = lag / segment.framerate
    frequency = 1 / period
    return frequency

find_frequency(corrs, 65, 90)
```

Листинг 4.9: Функция find_frequency

Наибольший пик приходится на `lag` 77, что соответствует частоте 572.(72) Гц.

По крайней мере, в этом примере воспринимаемый нами шаг соответствует самому высокому пику автокорреляционной функции (АКФ), а не самому высокому компоненту спектра.

Удивительно, но воспринимаемый шаг не меняется, если мы полностью удаляем фундаментальный. Вот как выглядит спектр, если мы используем фильтр высоких частот, чтобы убрать фундаментальный.

```
spectrum2 = segment.make_spectrum()
spectrum2.high_pass(600)
spectrum2.plot(high=3000)
thinkplot.config(xlabel='Frequency (Hz)', ylabel='Amplitude')
```

Листинг 4.10: Отфильтрованный спектр

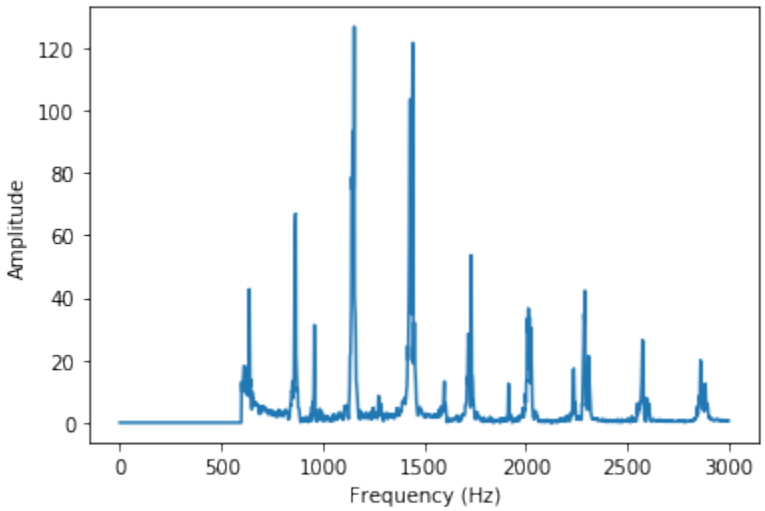

Рис. 4.4: Отфильтрованный спектр

Прослушаем полученный сегмент.

```
segment2 = spectrum2.make_wave()
segment2.make_audio()
```

Листинг 4.11: Прослушивание сегмента

Воспринимаемый шаг всё ещё составляет 288 Гц, хотя на этой частоте нет мощности. Это явление называется "недостающим фундаментальным".

Чтобы понять, почему мы слышим частоту, которой нет в сигнале, полезно взглянуть на функцию автокорреляции.

```
corrs = autocorr(segment2)
thinkplot.plot(corrs[:200])
thinkplot.config(xlabel='Lag', ylabel='Correlation', ylim=[-1.05,
    1.05])
```

Листинг 4.12: Визуализация АКФ

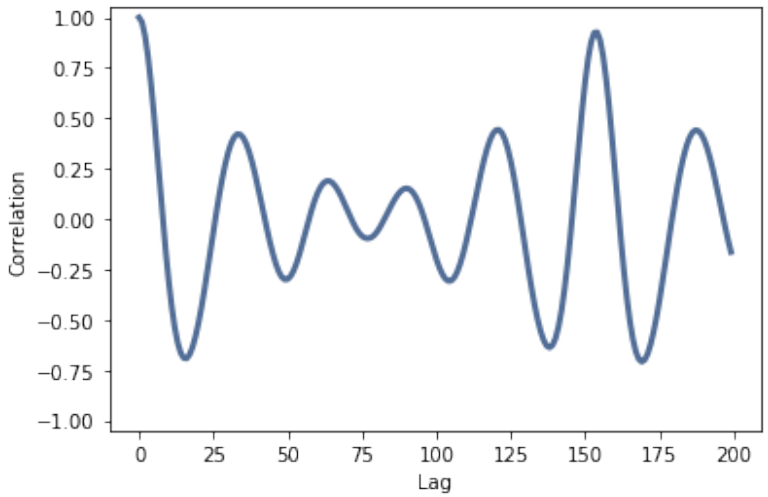

Рис. 4.5: Визуализация АКФ

Пятый пик, который соответствует 288 Гц, по-прежнему самый высокий. Остальные пики не являются гармониками.

Так почему же мы не воспринимаем ни одну из этих частот вместо 288 Гц? Причина в том, что высшие компоненты, присутствующие в сигнале, являются гармониками 288 Гц, а не гармониками 364,490,689 или 1336 Гц.

Наше ухо интерпретирует высокие гармоники как свидетельство того, что "правильный"фундаментал находится на частоте 288 Гц.

Если мы избавимся от высоких гармоник, эффект исчезнет. Рассмотрим спектр с удалёнными гармониками выше 1200 Гц.

```
spectrum4 = segment.make_spectrum()
spectrum4.high_pass(1000)
spectrum4.low_pass(1200)
spectrum4.plot(high=3000)
```

```
thinkplot.config(xlabel='Frequency (Hz)', ylabel='Amplitude')
```

Листинг 4.13: Спектр с удалёнными гармониками

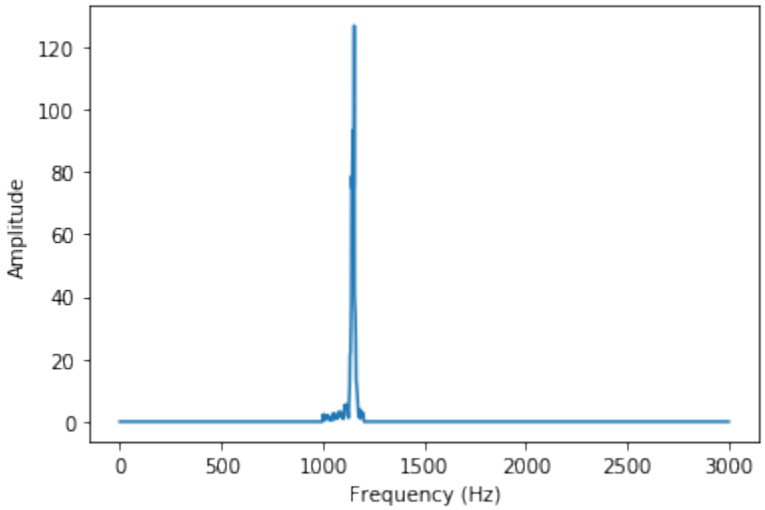

Рис. 4.6: Спектр с удалёнными гармониками

Теперь воспринимаемая частота составляет 1200 Гц.

```
segment4 = spectrum4.make_wave()
segment4.make_audio()
```

Листинг 4.14: Прослушивание нового звука

```
thinkdsp.TriangleSignal(freq=1200).make_wave(duration=0.5).make_audio()
```

Листинг 4.15: Треугольный сигнал на 1200 Гц

```
corrs = autocorr(segment4)
thinkplot.plot(corrs[:200])
thinkplot.config(xlabel='Lag', ylabel='Correlation', ylim=[-1.05,
    1.05])
```

Листинг 4.16: Визуализация АКФ

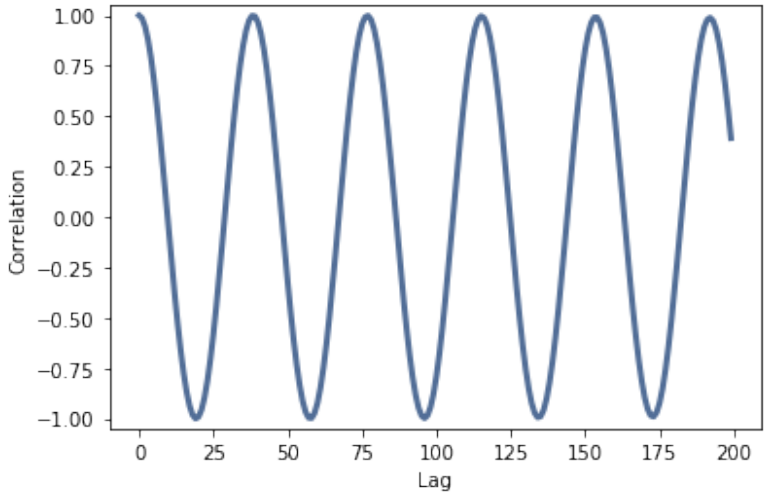

Рис. 4.7: Визуализация АКФ

```
find_frequency(corrs, 25, 50)
```

Листинг 4.17: Поиск пика

А если мы посмотрим на автокорреляционную функцию, то найдем самый высокий пик при `lag` = 38, что соответствует 1160 Гц.

Таким образом, эти эксперименты показывают, что восприятие высоты тона не полностью основано на спектральном анализе, но также информируется чем-то вроде автокорреляции.

Глава 5

Выводы

Во время выполнения лабораторной работы получены навыки работы с корреляцией, последовательной корреляцей, автокорреляцей. Также рассмотрена автокорреляционная функция (АКФ).