Санкт-Петербургский Государственный Политехнический Университет
Институт Компьютерных Наук и Технологий

Высшая школа интеллектуальных систем и суперкомпьютерных технологий

Отчёт по лабораторной работе №5
на тему
Автокорреляция

Работу выполнил
Студент группы 3530901/80203
Курняков П.М.
Преподаватель
Богач Н.В.

Санкт-Петербург, 2021 год

1 Настройка проекта

Перед тем как выполнять задания необходимо настроить проект и сделать все необходимые импорты:

```
from __future__ import print_function, division
%matplotlib inline
import thinkdsp
import thinkplot
import thinkstats2
import numpy as np
import pandas as pd
import warnings
warnings.filterwarnings('ignore')
from ipywidgets import interact, interactive, fixed
import ipywidgets as widgets
PI2 = np.pi * 2
```

Рис. 1: 2

2 Упражнение номер №1

Необходимо оценить высоты тона вокального чирпа для нескольких времен начала сегмента.

Возьмем методы из chap05.ipynb

```
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
```

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

Рис. 2: 2

Возьмем звук из репозитория и выделим из него короткий сегмент. Применим для этого сегмента функцию автокорреляции, чтобы оценить высоту тона:

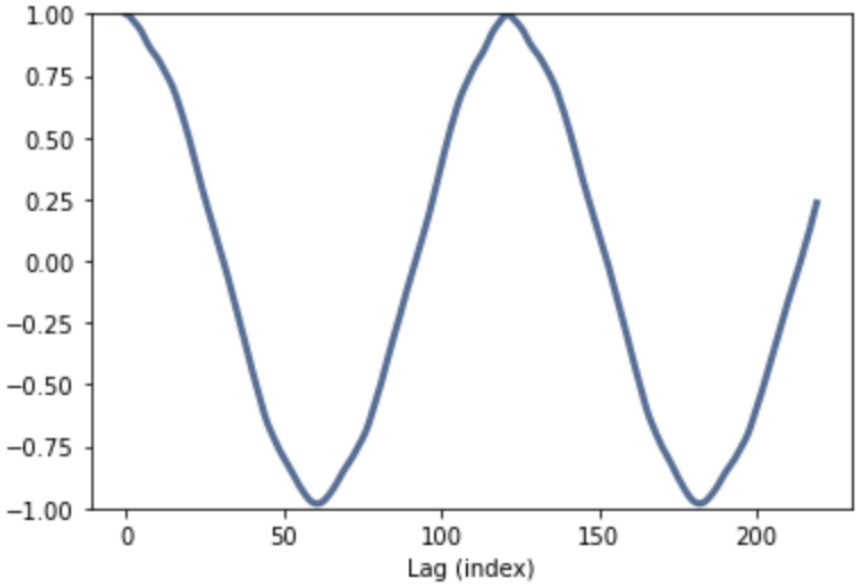


Рис. 3: 2

Пик ∈ [100;150]. Используем argmax, чтобы уточнить значение для этого пика

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

```
121
```

Рис. 4: 2

Вычислим частоту для этого значения:

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

```
364.4628099173554
```

Рис. 5: 2

Возьмем другой сегмент и проделаем те же манипуляции:

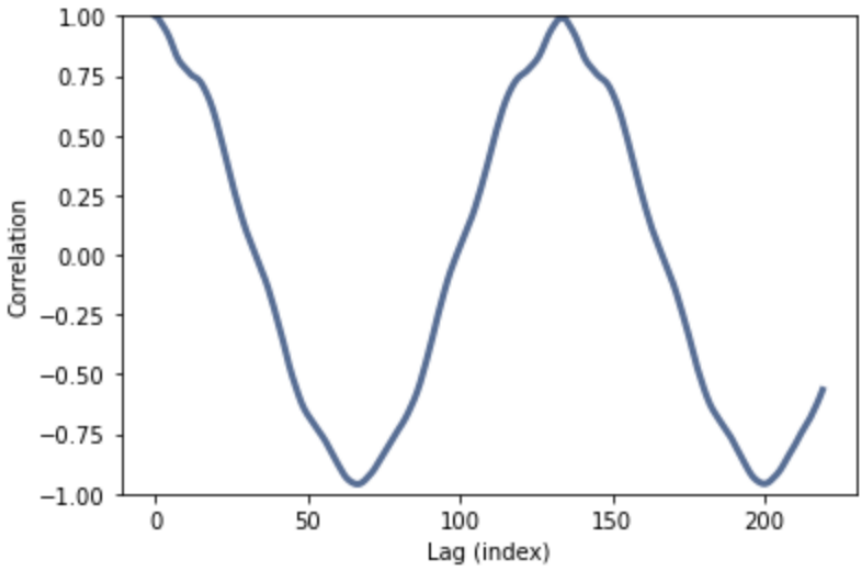


Рис. 6: 2

Пик ∈ [100;150]. Используем argmax, чтобы уточнить значение для этого пика

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

```
134
```

Рис. 7: 2

Вычислим частоту для этого значения:

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

```
329.1044776119403
```

Рис. 8: 2

Делаем вывод, что частота обратно пропорциональна времени начала сегмента

3 Упражнение номер №2

Инкапуслировать код в функцию из преложенного кода, назвав ее estimate_fundamental. Использовать жту функцию для отслеживания высоты тона записанного звука. Проверить насколько она хорошо работает, накладывая оценки высоты тона на спектрограмму записи.

Возьмем тот же звук что и в упражении 1. Распечатаем спектрограмму:

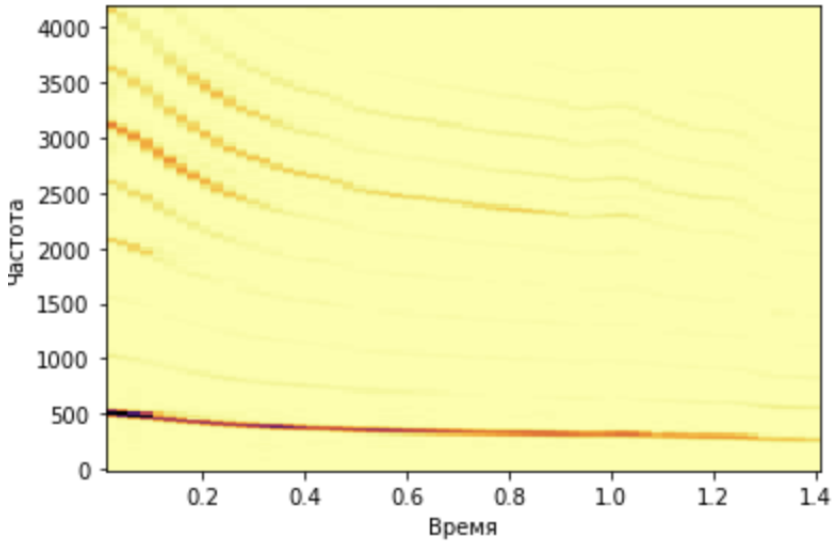


Рис. 9: 2

Упростим задачу, указав диапазон Lag для поиска, так как найти самый первый высокий пик достаточно сложно.

Реализуем функцию:

```
:
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
```

Рис. 10: 2

Рассмотрим пример использования данной функции:

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

```
436.63366336633663
```

Рис. 11: 2

Используем функцию для отслеживания высоты звука по сэмплу. Ts - это средние точки каждого сегмента.

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

ts = []
freqs = []

for start in starts:
    ts.append(start + step/2)
    segment = wave.segment(start=start, duration=duration)
    freq = estimate_fundamental(segment)
    freqs.append(freq)
```

Рис. 12: 2

Вот кривая для отслеживания высоты тона, наложенная на спектрограмму:

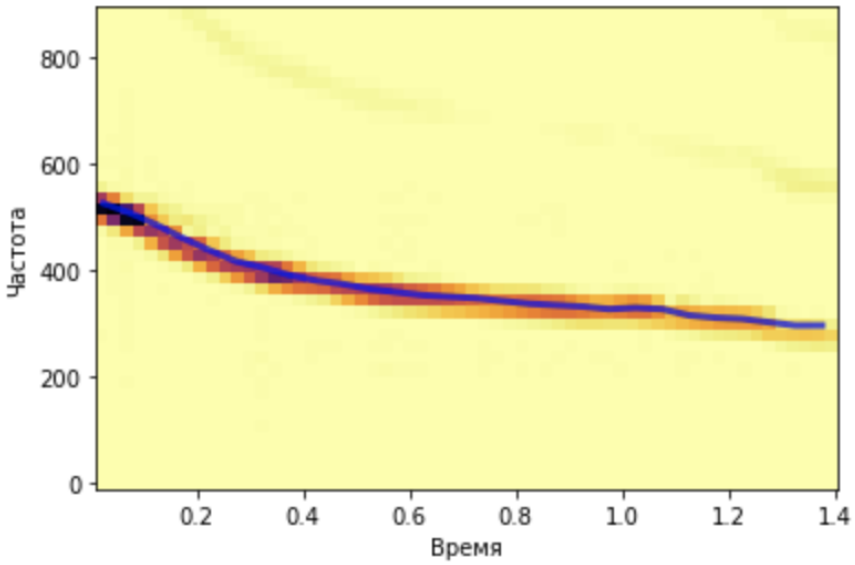


Рис. 13: 2

4 Упражнение номер №3

Вычилсить автокорреляцию цен в платежной системе биткоина.

Возьмем те же данные, что и в прошлой главе:

|:

	Currency	Date	Closing Price (USD)	24h Open (USD)	24h High (USD)	24h Low (USD)
0	BTC	2020-01-02	7174.744012	7179.957689	7237.014866	7152.992402
1	BTC	2020-01-03	6955.487580	7174.712357	7190.188749	6914.857474
2	BTC	2020-01-04	7291.219505	6955.487580	7390.041835	6852.093401
3	BTC	2020-01-05	7337.636670	7291.217504	7390.762935	7263.178696
4	BTC	2020-01-06	7347.433264	7337.421391	7487.333871	7316.763370
...	...	...	...	...	...	...
362	BTC	2020-12-29	26718.029463	26226.066130	27447.551384	26046.625578
363	BTC	2020-12-30	26975.729565	27038.735676	27169.225558	25875.049786
364	BTC	2020-12-31	28768.836208	27349.327233	28928.214391	27349.283204
365	BTC	2021-01-01	29111.521567	28872.829775	29280.045328	27916.625059
366	BTC	2021-01-02	29333.605121	28935.810981	29601.594898	28753.412314

367 rows × 6 columns

Рис. 14: 2

Построим волну:

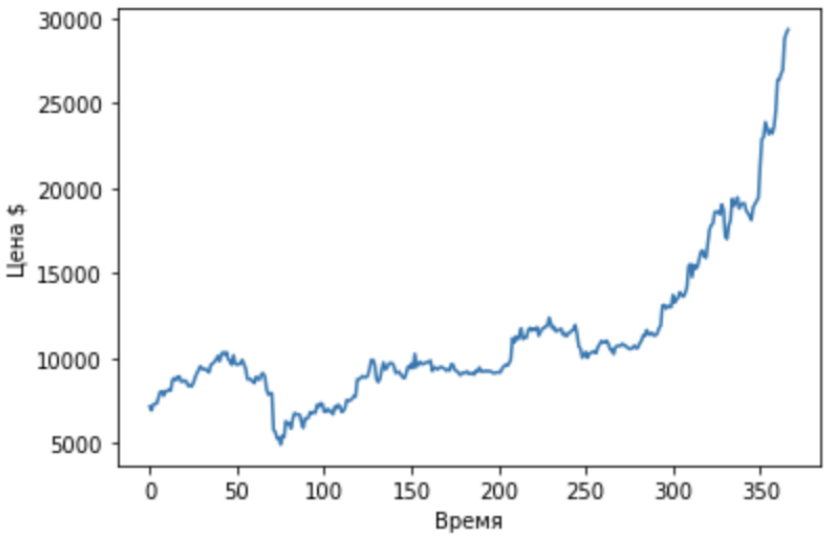


Рис. 15: 2

Применим функцию автокорреляции:

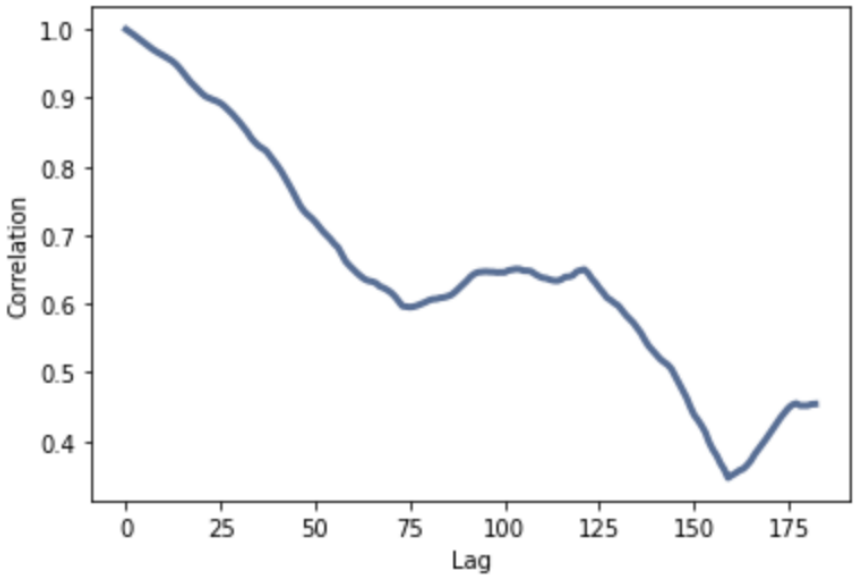


Рис. 16: 2

Функция медленно снижается по мере увеличения задержки, что указывает на какой-то розовый шум.

Мы можем сравнить autocorr с np.correlate, в котором используется определение корреляции, используемое при обработке сигналов. Он не устраняет предвзятость, нормализует и не стандартизирует волну.

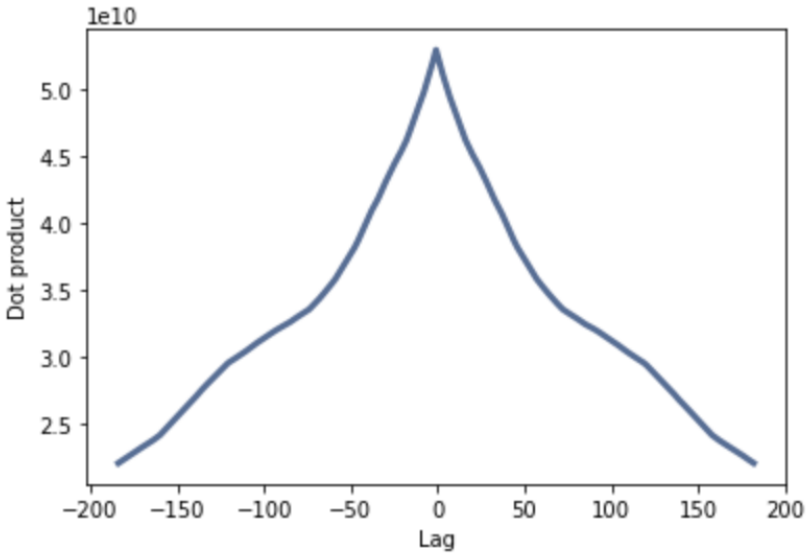


Рис. 17: 2

Вторая половина результата соответствует положительным Lags:

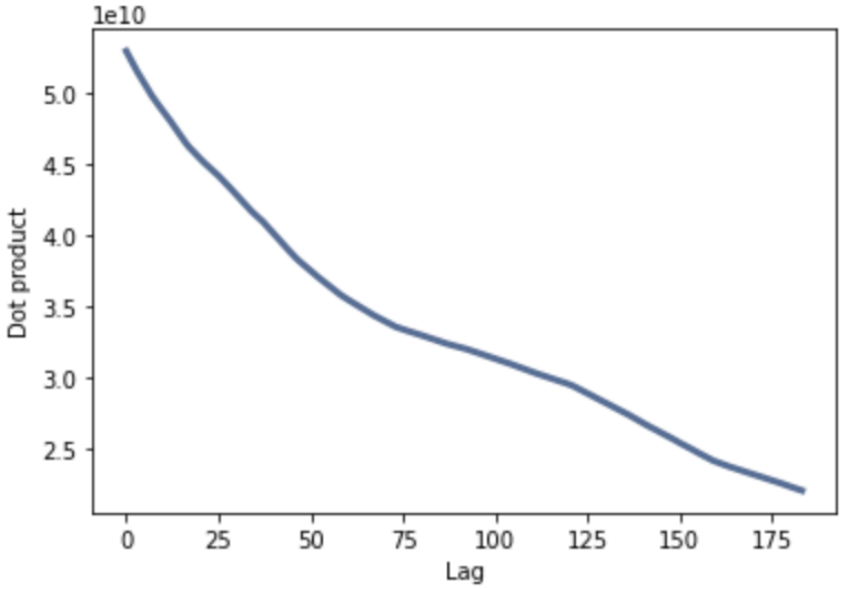


Рис. 18: 2

Нормализуем результаты постфактум, разделив их по длине:

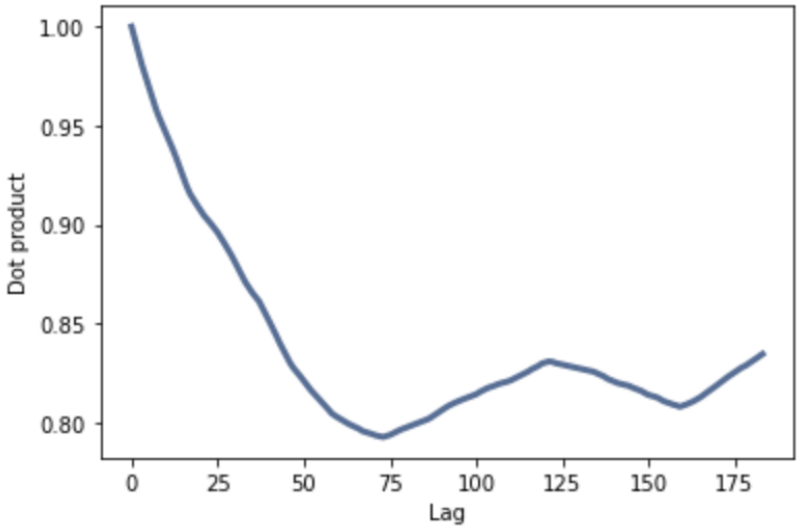


Рис. 19: 2

Наложим и сравним полученные результаты:

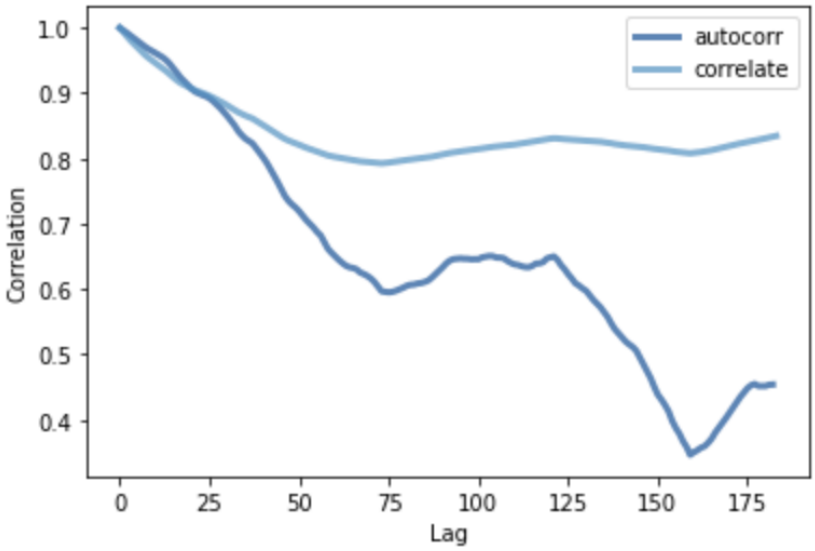


Рис. 20: 2

Даже после стандартизации результаты выглядят существенно иначе.

Для этого набора данных, вероятно, более подходящим является статистическое определение ACF.

5 Упражнение номер №4

Изучить предложенный файл saxophone.ipynb

Сделаем необходимые импорты:

```
:
from __future__ import print_function, division

import thinkdsp
import thinkplot
import thinkstats2

import numpy as np

import warnings
warnings.filterwarnings('ignore')

from IPython.html.widgets import interact, fixed
from IPython.html import widgets

PI2 = np.pi * 2

%matplotlib inline
```

Рис. 21: 2

Возьмем звук, предложенный учебником и выведем спектрограмму:

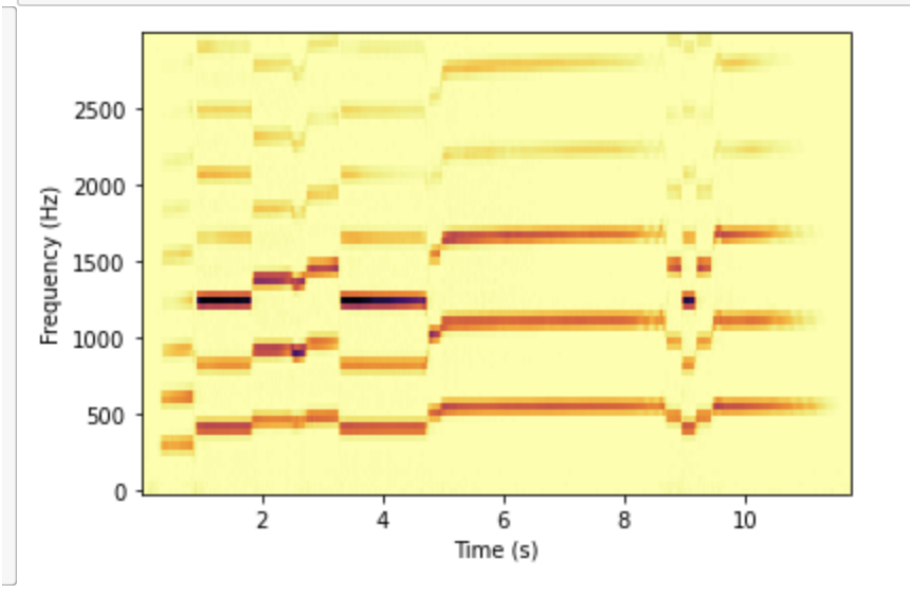


Рис. 22: 2

Спектрограмма показывает гармоническую структуру во времени.
Выберем сегмент, чтобы лучше увидеть гармоники и выведем его спектр:

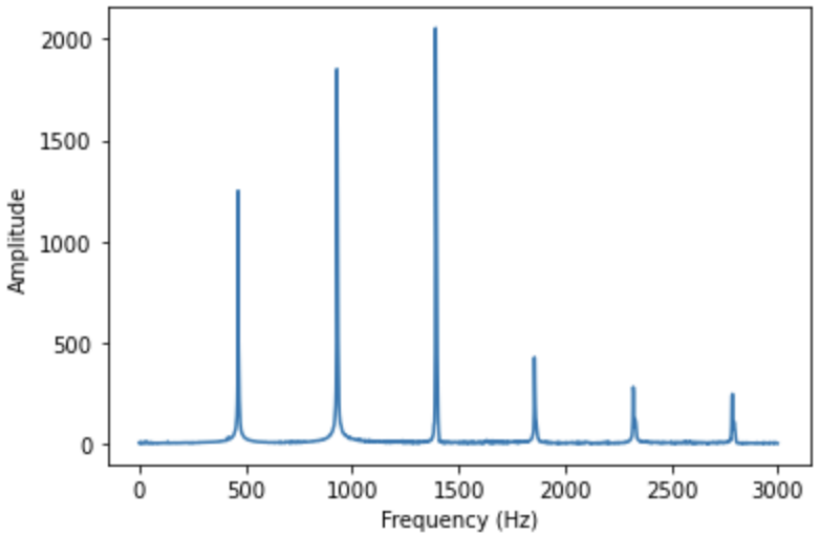


Рис. 23: 2

Пики в спектре находятся на частотах 1392, 928 и 464 Гц.

```
[(2054.0622639591206, 1392.0),
 (1850.8544230639036, 928.0),
 (1684.8468845494765, 1394.0),
 (1332.8150506072802, 930.0),
 (1249.1774991462646, 464.0),
 (1177.6718910227576, 1396.0),
 (857.3729096557305, 932.0),
 (742.841588837269, 1398.0),
 (515.1804113061312, 934.0),
 (513.7226300908811, 466.0)]
```

Рис. 24: 2

Высота, которую мы воспринимаем, является основной, 464 Гц, хотя это не доминирующая частота.

Для сравнения, вот треугольная волна на частоте 464 Гц.

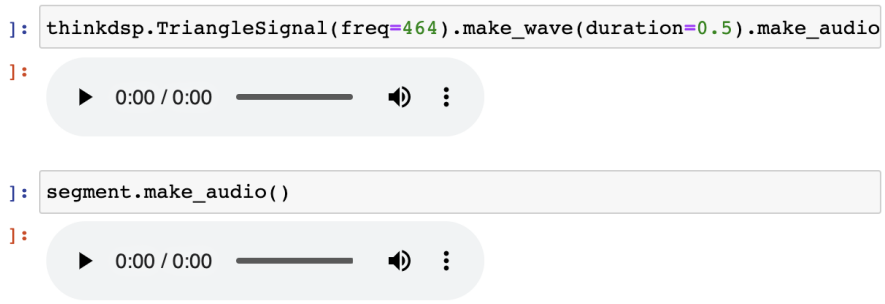


Рис. 25: 2

У них одинаковая воспринимаемая высота звука.

Чтобы понять, почему мы воспринимаем основную частоту, даже если она не является доминирующей, полезно взглянуть на функцию автокорреляции (ACF).

Следующая функция вычисляет ACF, выбирает вторую половину (которая соответствует положительным задержкам) 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
```

```
: corrs = autocorr(segment)
thinkplot.plot(corrs[:200])
thinkplot.config(xlabel='Lag', ylabel='Correlation', ylim=[-1.05, 1.
```

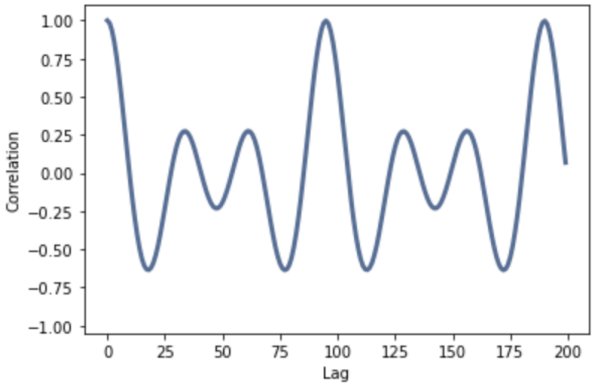


Рис. 26: 2

Первый главный пик находится рядом с лагом 100.

Следующая функция находит самую высокую корреляцию в заданном диапазоне задержек 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, 80, 100)
```

```
95

464.2105263157895
```

Рис. 27: 2

Наибольший пик приходится на lag 95, что соответствует частоте 464.21 Гц. По крайней мере, в этом примере высота звука, которую мы воспринимаем, соответствует наивысшему пику автокорреляционной функции (ACF), 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.

Выведем спектр:

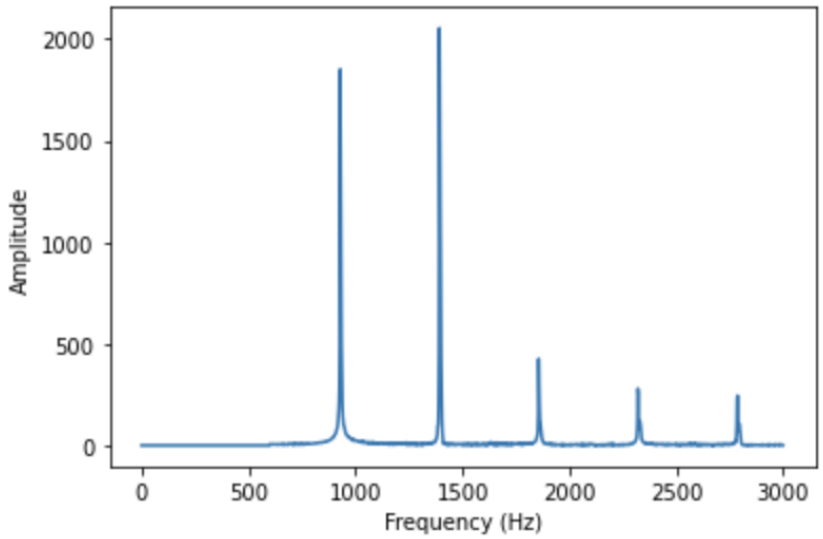


Рис. 28: 2

Создадим звук. Воспринимаемая высота звука по-прежнему составляет 464 Гц, хотя на этой частоте нет мощности.

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

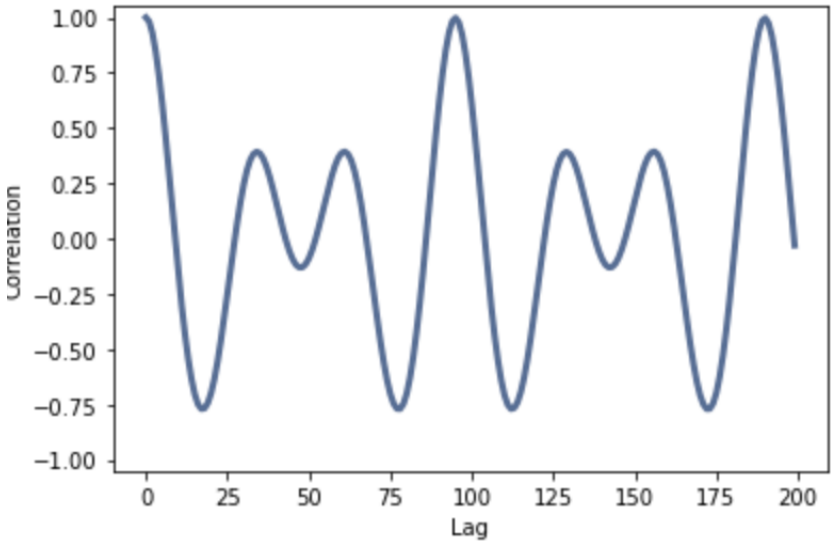


Рис. 29: 2

Третий пик, соответствующий 464 Гц, по-прежнему самый высокий:

```
B [60]: find_frequency(corrs, 80, 100)
        95
Out[60]: 464.2105263157895

B [61]: find_frequency(corrs, 20, 50)
        34
Out[61]: 1297.0588235294117

B [62]: find_frequency(corrs, 50, 80)
        61
Out[62]: 722.9508196721312
```

Рис. 30: 2

Остальные пики не являются гармониками.

Наше ухо интерпретирует высокие гармоники как свидетельство того, что "правильный"фундаментал находится на частоте 464 Гц.

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

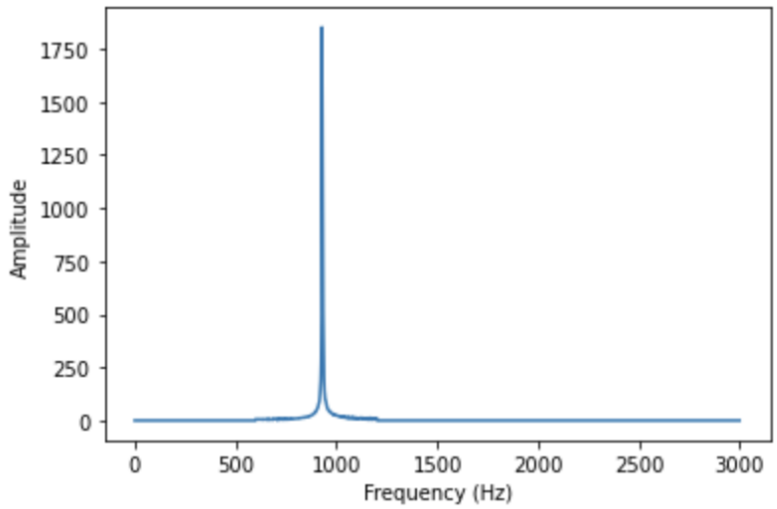
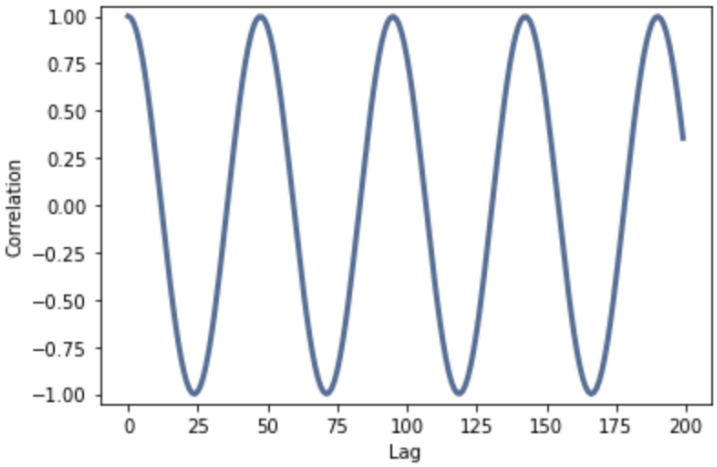


Рис. 31: 2

Сейчас воспринимается частота примерно равная 930Гц. Создадим треугольный сигнал. И если мы посмотрим на функцию автокорреляции, мы найдем самый высокий пик при лаге = 47, что соответствует 938 Гц.




```
]: find_frequency(corrs, 30, 50)
```

```
47
```

```
]: 938.2978723404256
```

Рис. 32: 2