Ivanov_lab5

March 12, 2020

Глава 5 (Иванов Илья, группа 3530901/70203)

1 Ряд Фурье, ППФ, ОПФ, свойства преобразования Фурье

1.1 Ряд Фурье

Всякая периодическая функция $\varphi_p(t)$, удовлетворяющая условиям Дирихле, может быть представлена в виде ряда Фурье

$$\varphi_p(t) = \sum_{k=-\infty}^{\infty} C_k e^{i2\pi k f_1 t},$$

где $f_1 = 1/T_1$; T_1 - период функции $\varphi_p(t)$; C_k - постоянные коэффициенты.

Условия Дирихле означают, что функция должна быть ограниченной, кусочно-непрерывной и иметь на протяжении периода конечное число экстремумов.

Ряд Фурье можно представить следующим образом:

$$\varphi_p(t) = \sum_{k=-\infty}^{\infty} \left(\frac{1}{T_1} \int_{-T_1/2}^{T_1/2} \varphi_p(t) e^{-i2\pi k f_1 t}\, dt\right) e^{i2\pi k f_1 t}$$

Ряд Фурье в таком виде содержит бесконечное число членов. Именно в этом случае можно поставить знак равенства между левой и правой частью. На практике, естественно, мы вынуждены ограничиваться конечным числом членов, вследствие чего указанное равенство соблюдается только приближённо.

1.2 Интеграл Фурье

Ряд Фурье справедлив для периодических сигналов. Однако на его основе можно вывести соотношение для преобразования непериодических сигналов. Действительно, непериодический сигнал можно представить как частный случай периодического, но имеющего период, стремящийся к бесконечности.

В результате для непериодической функции получим:

$$\varphi(t) = \int_{-\infty}^{\infty} \left(\int_{-\infty}^{\infty} \varphi(t) e^{-i2\pi f t}\, dt\right) e^{i2\pi f t}\, df.$$

Это соотношение носит название интеграла или преобразования Фурье. Оно объединяет прямое преобразование Фурье (ППФ):

$$\Phi(f) = \int_{-\infty}^{\infty} \varphi(t) e^{-i2\pi f t}\, dt$$

и обратное преобразование Фурье (ОПФ):

$$\varphi(t) = \int_{-\infty}^{\infty} \Phi(f) e^{i2\pi f t}\, df.$$

Прямое и обратное преобразование Фурье существуют для функций с ограниченной энергией, т.е. таких функций, для которых

$$\int_{-\infty}^{\infty} |\varphi(t)|^2 \, dt \neq \infty.$$

1.3 Свойства преобразования Фурье

1.3.1 Суммирование функций

Преобразование Фурье - линейное преобразование. Отсюда следует, что ПФ линейной комбинации некоторых функций равно аналогичной линейной комбинации ПФ этих функций

$$\sum_{i=1}^{n} \alpha_i \varphi_i(t) \leftrightarrow \sum_{i=1}^{n} \alpha_i \Phi_i(f),$$

где α_i - постоянный коэффициент, а стрелки означают переход к преобразованию Фурье и обратно.

1.3.2 Смещение функций

При смещении функции по аргументу на t_0 её ПФ умножается на $e^{i2\pi f t_0}$. Действительно, проводя замену переменной $t' = t + t_0$, получим

$$\varphi(t + t_0) \leftrightarrow \int_{-\infty}^{\infty} \varphi(t + t_0) e^{-i2\pi f t} \, dt = \int_{-\infty}^{\infty} \varphi(t') e^{-i2\pi f (t' - t_0)} \, dt' = e^{i2\pi f t_0} \Phi(f).$$

1.3.3 Перемножение функций

ПФ произведения двух функций $\varphi_1(t)\varphi_2(t)$ равно свёртке их ПФ $\int_{-\infty}^{\infty} \Phi_1(f') \times \Phi_2(f - f') \, df'$. Это свойство доказывается путём использования ОПФ и изменения порядка интегрирования.

Как мы видим, одна из двух функций берётся в том виде, как она исходно задана - $\Phi_1(f')$, а для другой, во-первых, изменяется направление оси абсцисс и, во-вторых, производится сдвиг функции по этой оси на некоторое значение аргумента f:

$$\Phi_2(-f' + f)$$

Затем эти две функции перемножаются и произведение интегрируется, т.е. находится площадь под кривой, соответствующей произведению $\Phi_1(f')\Phi_2(f - f')$. Полученный интеграл и является значением свёртки для заданного значения аргумента f.

1.3.4 Свёртывание функций

ПФ свёртки двух функций $\int_{-\infty}^{\infty} \varphi_1(t')\varphi_2(t - t') \, dt'$ равно произведению ПФ свёртываемых функций $\Phi_1(f)\Phi_2(f)$. Это свойство может быть кратко записано в виде

$$\varphi_1(t) * \varphi_2(t) \leftrightarrow \Phi_1(f)\Phi_2(f).$$

Доказывается оно путём изменения порядка интегрирования.

2 Теория

В данной главе будут точнее определены термины “коррелированный” и “некоррелированный”, а также рассмотрена автокорреляционная функция - полезнейший инструмент для анализа сигналов.

2.1 Корреляция

Обычно корреляция между переменными означает, что в каждой известной величине есть некоторая информация о другой. Из нескольких способов оценки корреляции наиболее известен коэффициент корреляции Пирсона - произведение моментов, обычно обозначаемое ρ. Для двух переменных `x` и `y`, содержащих каждая `N` значений:

$$\rho = \sum_i (x_i - \mu_x)(y_i - \mu_y) / N\sigma_x\sigma_y,$$

где μ_x и μ_y - средние от x и y, а σ_x и σ_y - их стандартные отклонения.

Корреляция Пирсона определена в интервале от -1 до +1 (включительно). Если ρ положительно, то корреляция положительная, то есть если одна переменная велика, то и другая тоже велика. Если ρ отрицательное, то корреляция отрицательная, - если одна переменная велика, то другая мала.

Значение ρ показывает силу корреляции. Если ρ равно +1 или -1, переменные идеально коррелируют, то есть если известна одна, то можно точно предсказать другую. Если ρ близко к нулю, корреляция, вероятно, слабая, так что при одном известном мало что можно сказать о другом.

2.2 Последовательная корреляция

Сигналы - это чаще всего измерения изменяющихся во времени величин. Такие измерения почти всегда имеют последовательную корреляцию, или корреляцию между любым элементом и следующим за ним (или предыдущим). Для расчёта последовательной корреляции сигнал можно сдвинуть и вычислить корреляцию сдвинутой версии с оригиналом:

[1]:
```
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
```

2.3 Автокорреляция

В предыдущем разделе корреляция вычислялась между каждым значением и следующим за ним, поэтому элементы массива сдвигали на 1. Но последовательную корреляцию легко вычислить и для иных интервалов.

`serial_corr` можно рассматривать как функцию, сопоставляющую значение `lag` соответствующей корреляции, и её можно оценить, перебирая значения `lag` в цикле:

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

`autocorr` берёт объект `wave` и возвращает автокорреляционную функцию как пару последовательностей: `lags` - это последовательность целых чисел от 0 до половины длины

сигнала; `corrs` - это значения последовательной корреляции для всех `lag`.

Если значение сигнала зависит от многих предыдущих значений, то говорят, что существует `зависимость дальнего действия`.

2.4 Корреляция как скалярное произведение

В обработке сигналов обычно обрабатываются сигналы без смещения (постоянной составляющей), их среднее равно 0, и нормализованные, со стандартным отклонением, равным 1. В этом случае определение ρ упрщается:

$$\rho = 1/N \sum_i x_i y_i$$

Обычно упрощают ещё больше:

$$\rho = \sum_i x_i y_i$$

Это определение корреляции не "стандартизированно", поэтому его интервал не всегда от -1 до +1. Но у него другие полезные свойства.

Если рассматривать `x` и `y` как вектора, то это формула скалярного произведения.

Скалярное произведение указывает на степень похожести сигналов. Если они нормализованы, то их стандартные отклонения равны 1:

$$x \cdot y = \cos\theta,$$

где θ - угол между векторами.

3 Упражнения

[3]:
```
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
```

3.1 Упражнение 5.1.

The Jupyter notebook for this chapter, chap05.ipynb, includes an interaction that lets you compute autocorrelations for different lags. Use this interaction to estimate the pitch of the vocal chirp for a few different start times.

Запись вокального исполнения чирпа:

[4]:
```
wave = thinkdsp.read_wave('28042__bcjordan__voicedownbew.wav')
wave.normalize()
wave.make_audio()
```

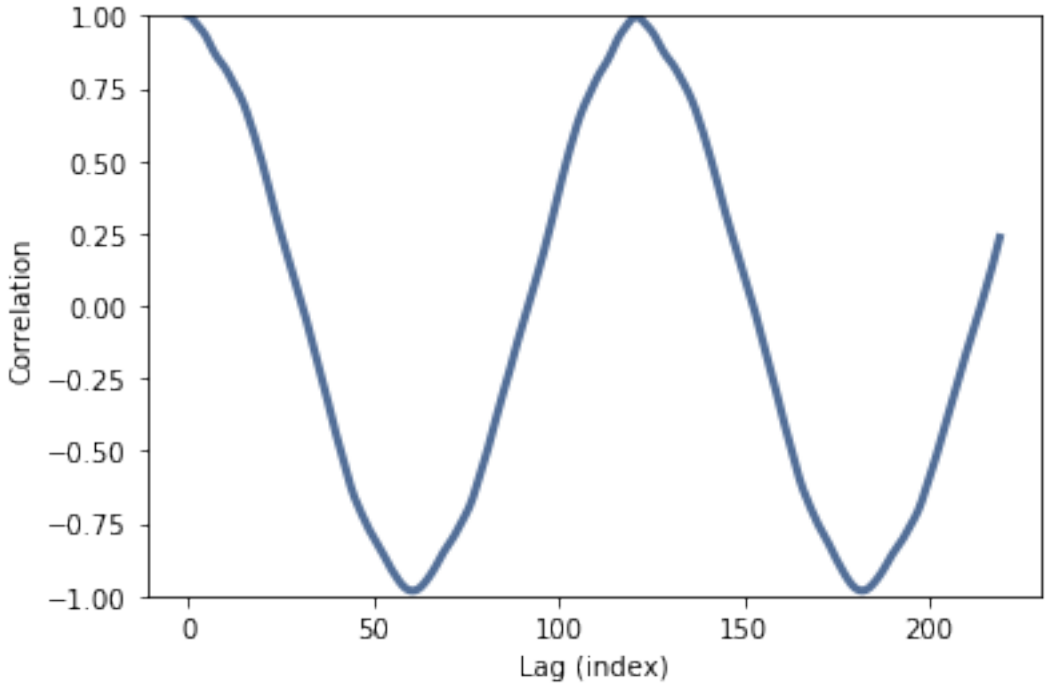
[4]:
```
<IPython.lib.display.Audio object>
```

Возьмём короткий сегмент сигнала с началом через 0.5 секунды и длительностью 0.01 секунды:

[5]:
```
segment = wave.segment(start=0.5, duration=0.01)
```

Оценим высоту тона, применив автокорреляционную функцию:

[6]:
```
lags, corrs = autocorr(segment)
thinkplot.plot(lags, corrs)
thinkplot.config(xlabel='Lag (index)', ylabel='Correlation', ylim=[-1, 1])
```



Пик находится близко к `lag`= 120. Используем argmax, чтобы уточнить значение `lag` для этого пика:

[7]:
```
low, high = 110, 130
lag = np.array(corrs[low:high]).argmax() + low
lag
```

[7]:
```
121
```

Вычислим соответствующую частоту:

[8]:
```
period = lag / segment.framerate
frequency = 1 / period
frequency
```

[8]:
```
364.4628099173554
```

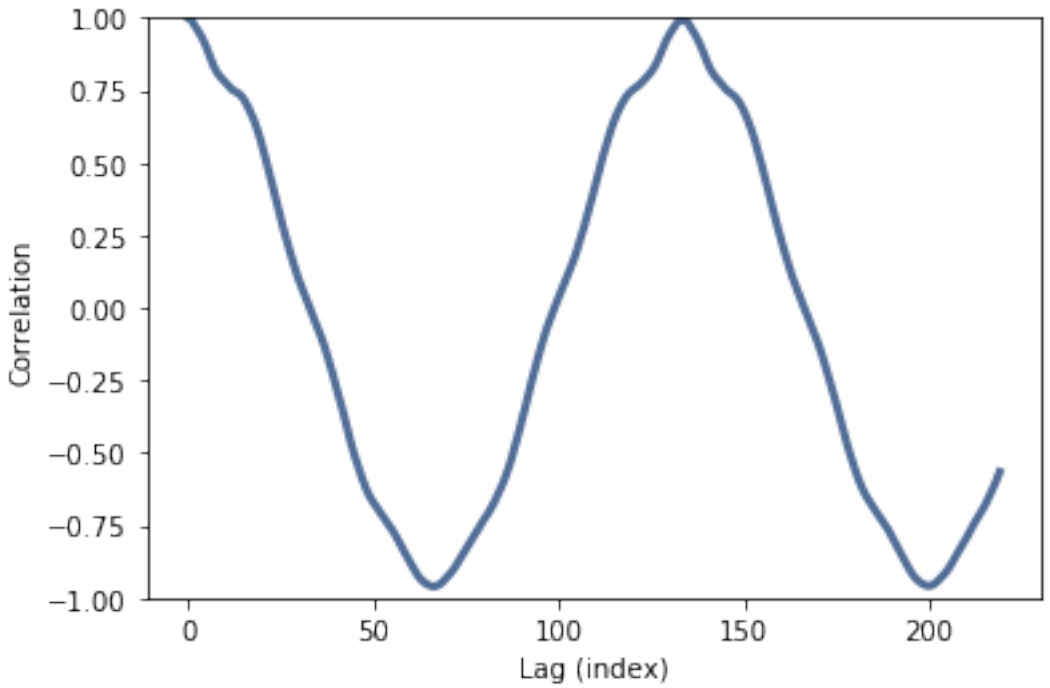
Искомая основная частота - 364 Гц.

Теперь возьмём сегмент сигнала с началом через 1 секунду:

[9]:
```
segment = wave.segment(start=1.0, duration=0.01)
```

Оценим высоту тона, применив автокорреляционную функцию:

[10]:
```
lags, corrs = autocorr(segment)
thinkplot.plot(lags, corrs)
thinkplot.config(xlabel='Lag (index)', ylabel='Correlation', ylim=[-1, 1])
```



Пик находится близко к `lag`= 135. Используем `argmax`, чтобы уточнить значение `lag` для этого пика:

[11]:
```
low, high = 130, 140
lag = np.array(corrs[low:high]).argmax() + low
lag
```

[11]:
```
134
```

Вычислим соответствующую частоту:

[12]:
```
period = lag / segment.framerate
frequency = 1 / period
frequency
```

[12]:
```
329.1044776119403
```

Искомая основная частота - 329 Гц.

Основная частота ожидаемо уменьшается при увеличении времени начала сегмента.

3.2 Упражнение 5.2.

The example code in chap05.ipynb shows how to use autocorrelation to estimate the fundamental frequency of a periodic signal. Encapsulate this code in a function called `estimate_fundamental`, and use it to track the pitch of a recorded sound.

To see how well it works, try superimposing your pitch estimates on a spectrogram of the recording.

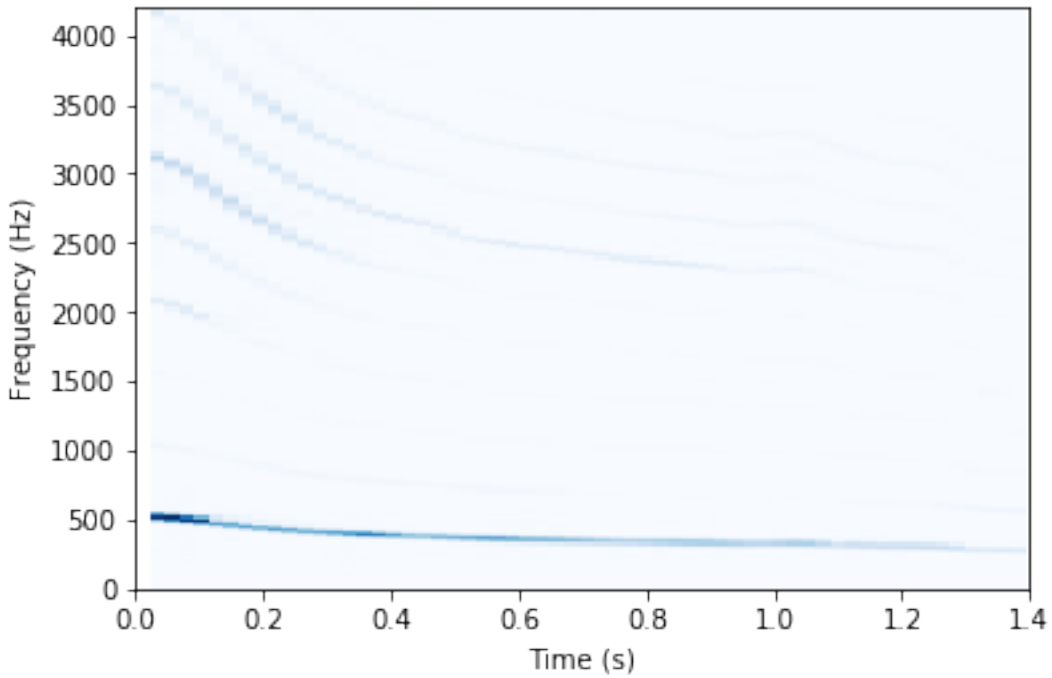
Уже использовавшаяся в предыдущем пункте запись вокального исполнения чирпа:

[13]:
```
wave = thinkdsp.read_wave('28042__bcjordan__voicedownbew.wav')
wave.normalize()
wave.make_audio()
```

[13]:
```
<IPython.lib.display.Audio object>
```

Получим спектрограмму сигнала:

[14]:
```
wave.make_spectrogram(2048).plot(high=4200)
thinkplot.config(xlabel='Time (s)', ylabel='Frequency (Hz)', xlim=[0, 1.4],␣
 ↪ylim=[0, 4200])
```



Функция для оценки основной частоты периодического сигнала, использующая автокорреляцию:

[15]:
```
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
```

Найти самый высокий пик в автокорреляционной функции довольно сложно. В данном случае это реализовано путём указания диапазона `lag`-ов для поиска.

Пример работы функции:

[16]:
```
duration=0.01
segment = wave.segment(start=0.3, duration=duration)
freq = estimate_fundamental(segment)
freq
```

[16]:
```
404.5871559633028
```

Цикл, оценивающий основную частоту сегментов по всей записи:

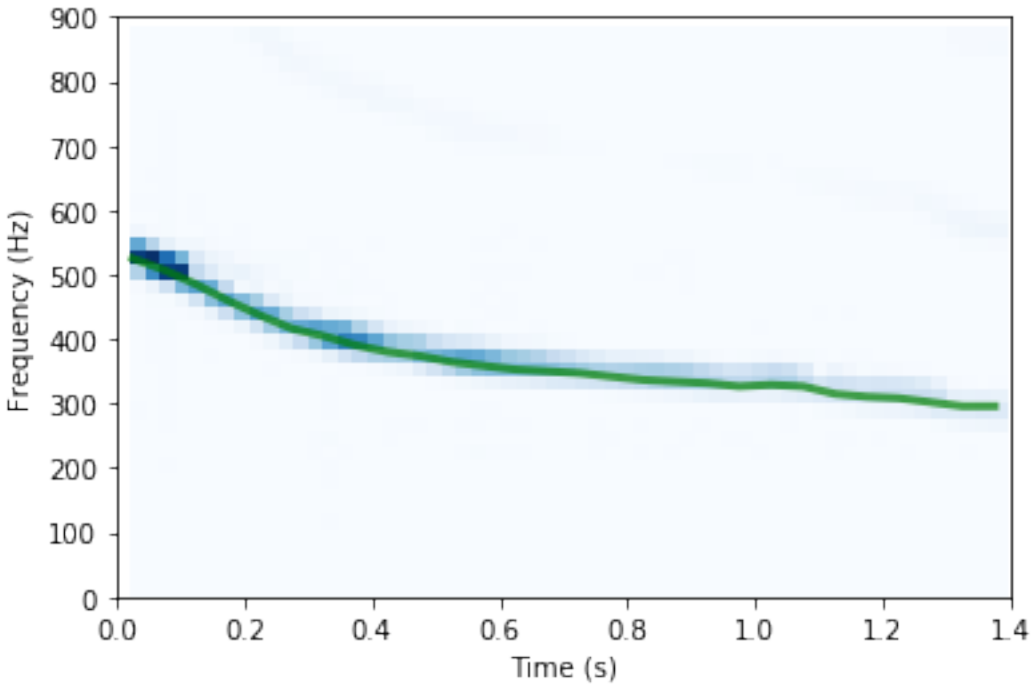
[17]:
```
step = 0.05
starts = np.arange(0.0, 1.4, step)
```

```
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)
```

Наложим полученную оценку высоты тона на спектрограмму записи:

[18]:
```
wave.make_spectrogram(2048).plot(high=900)
thinkplot.plot(ts, freqs, color='green')
thinkplot.config(xlabel='Time (s)', ylabel='Frequency (Hz)', xlim=[0, 1.4],␣
 ↪ylim=[0, 900])
```



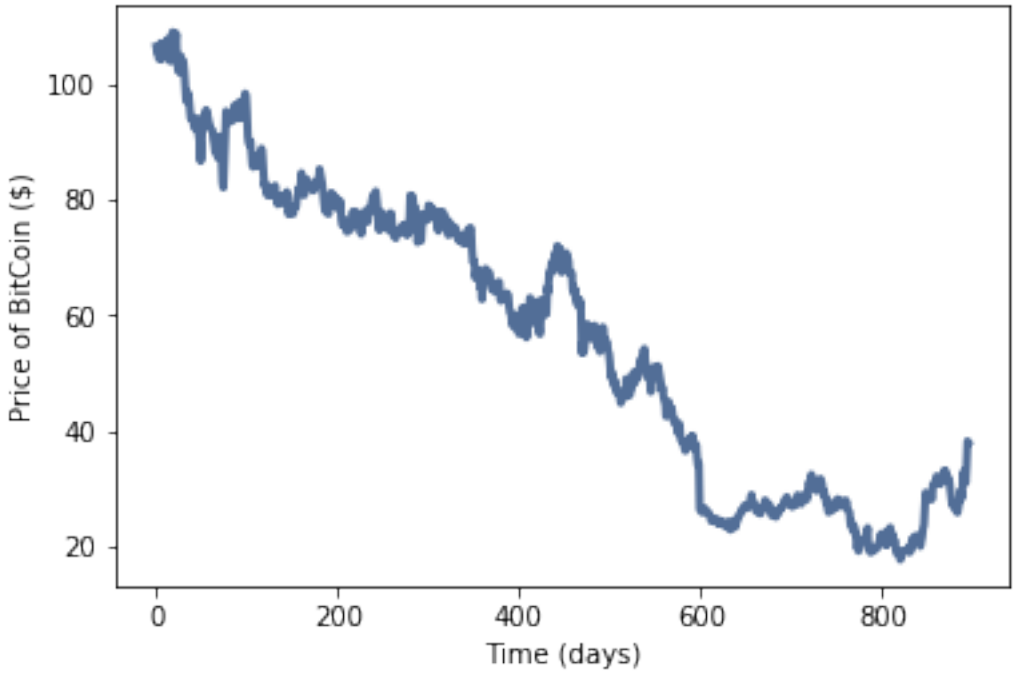
Полученная оценка мало отличается от спектрограммы.

3.3 Упражнение 5.3.

If you did the exercises in the previous chapter, you downloaded the historical price of BitCoins and estimated the power spectrum of the price changes. Using the same data, compute the autocorrelation of BitCoin prices. Does the autocorrelation function drop off quickly? Is there evidence of periodic behavior?

[19]:
```
df = pd.read_csv('BitCoin.csv', nrows=1625, parse_dates=[0])
ys = df.Close.values
```

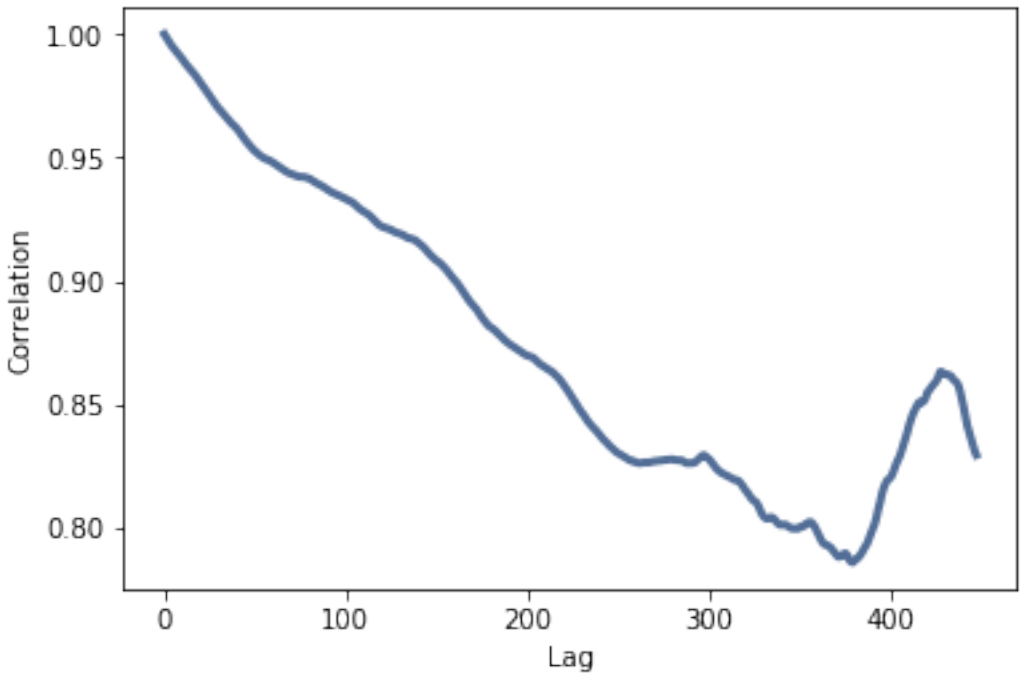
[20]:
```
wave = thinkdsp.Wave(ys, framerate=1)
wave.plot()
thinkplot.config(xlabel='Time (days)', ylabel='Price of BitCoin ($)')
```



Получим автокорреляционную функцию:

[21]:
```
from autocorr import autocorr

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



Корреляция медленно снижается по мере увеличения `lag`, что наводит на мысль о розовом шуме. Кроме того, есть умеренная корреляция с `lag` около 425. Заметных признаков периодичности процесса нет.

3.4 Упражнение 5.4.

In the repository for this book you will find a Jupyter notebook called `saxophone.ipynb` that explores autocorrelation, pitch perception, and a phenomenon called the missing fundamental. Read through this notebook and run the examples. Try selecting a different segment of the recording and running the examples again.

`saxophone.ipynb` был просмотрен и разобран.

Выберем сегмент, отличный от рассматриваемого в `saxophone.ipynb` и повторим исследование.

Запись саксофона:

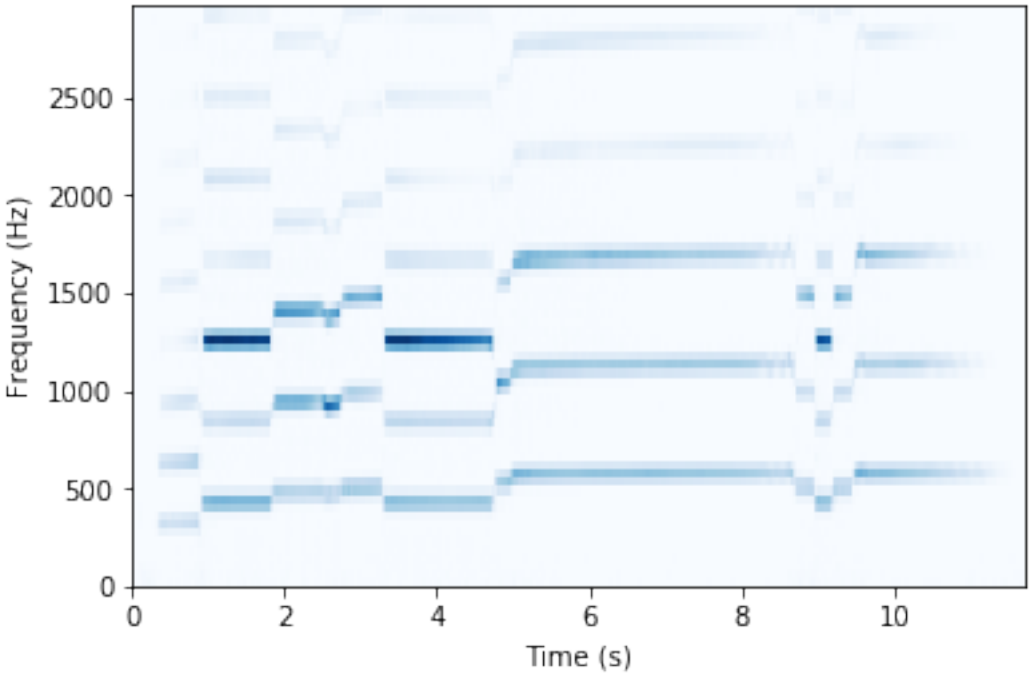
[22]:
```
wave = thinkdsp.read_wave('100475__iluppai__saxophone-weep.wav')
wave.normalize()
wave.make_audio()
```

[22]:
```
<IPython.lib.display.Audio object>
```

Получим спектрограмму:

[23]:
```
gram = wave.make_spectrogram(seg_length=1024)
gram.plot(high=3000)
```

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

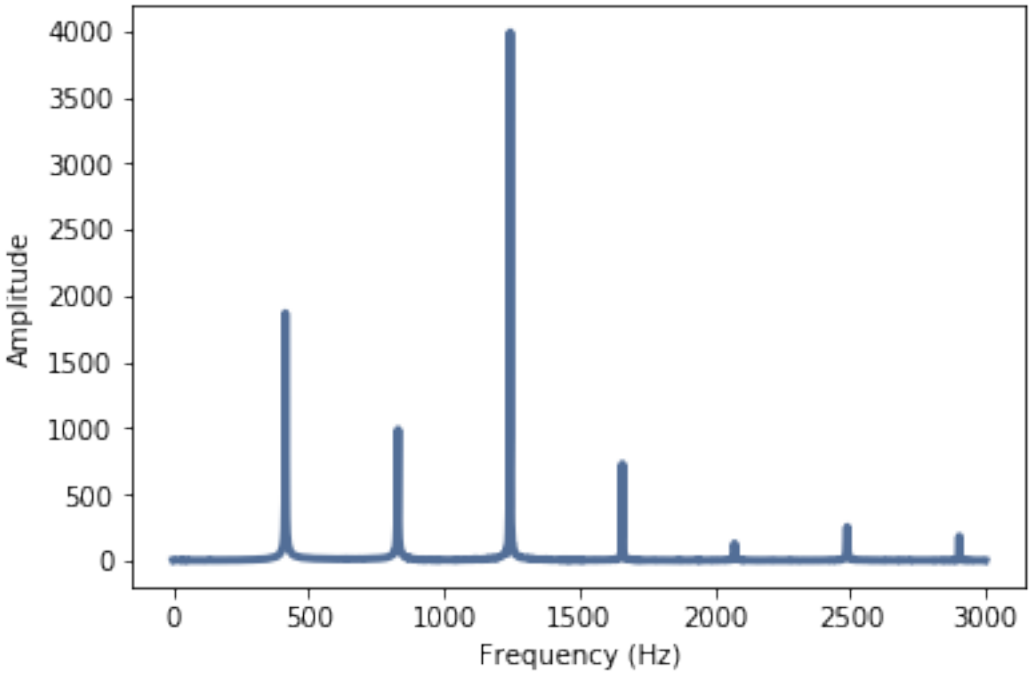


Чтобы увидеть гармоники более четко, выберем сегмент около 4-секундной отметки длительностью 0.5 секунды и получим его спектр:

[24]:
```
start = 4.0
duration = 0.5
segment = wave.segment(start=start, duration=duration)
segment.make_audio()
```

[24]:
```
<IPython.lib.display.Audio object>
```

[25]:
```
spectrum = segment.make_spectrum()
spectrum.plot(high=3000)
thinkplot.config(xlabel='Frequency (Hz)', ylabel='Amplitude')
```



Выведем пики спектра:

[26]:
```
spectrum.peaks()[:10]
```

[26]:
```
[(3992.340463145424, 1244.0),
 (1870.0503470749502, 414.0),
 (990.3942903418163, 830.0),
 (948.6612140134521, 1242.0),
 (784.1921786995277, 416.0),
 (730.5614131345402, 1658.0),
 (691.7629429880423, 828.0),
 (667.6068421265664, 1246.0),
 (437.4682504156278, 412.0),
 (419.39953596812086, 1240.0)]
```

Получаем пики на 1244, 414 и 830 Гц.

Воспринимаемая нами частота - основная - 414 Гц, хоть она и не является доминирующей.

Для сравнения, создадим треугольную волну с частотой 414 Гц:

[27]:
```
thinkdsp.TriangleSignal(freq=414).make_wave(duration=0.5).make_audio()
```

[27]:
```
<IPython.lib.display.Audio object>
```

Выбранный сегмент:

[28]:
```
segment.make_audio()
```

[28]:
```
<IPython.lib.display.Audio object>
```

Они имеют одинаковую воспринимаемую высоту.

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

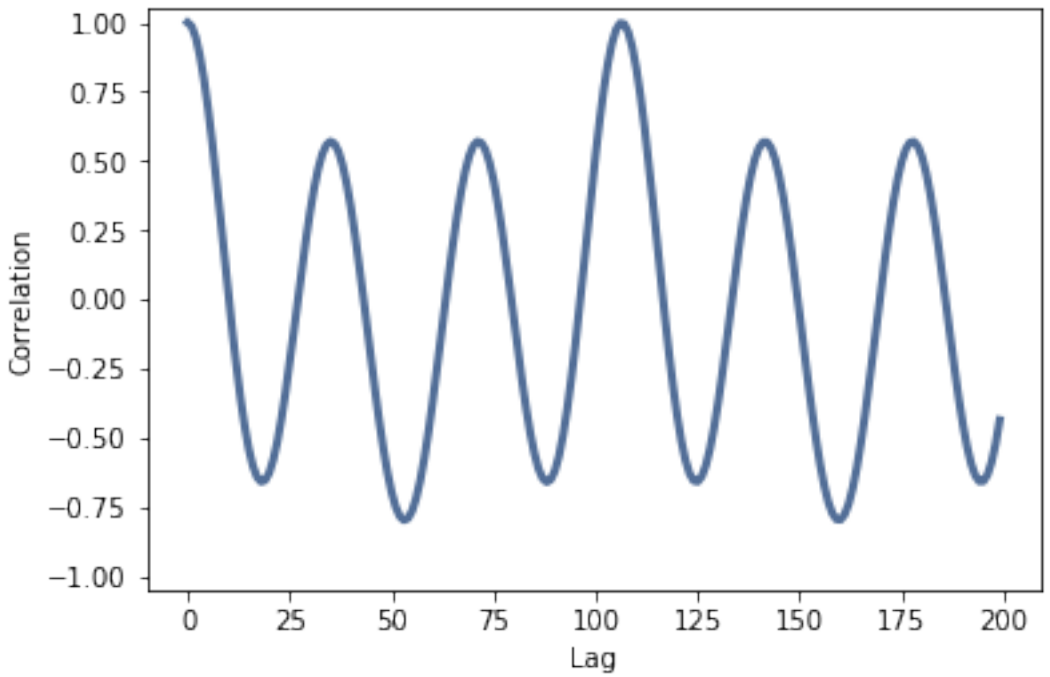
Следующая функция вычисляет ACF, выбирает вторую половину (что соответствует положительным `lag`) и нормализует результаты:

[29]:
```
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
```

Результат применения:

[30]:
```
corrs = autocorr(segment)
thinkplot.plot(corrs[:200])
thinkplot.config(xlabel='Lag', ylabel='Correlation', ylim=[-1.05, 1.05])
```



Основной пик близок к `lag` = 100

Следующая функция находит наибольшую корреляцию в заданном диапазоне `lag`-ов и возвращает соответствующую частоту:

[31]:
```
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
```

Результат применения:

[32]:
```
find_frequency(corrs, 100, 120)
```

```
106
```

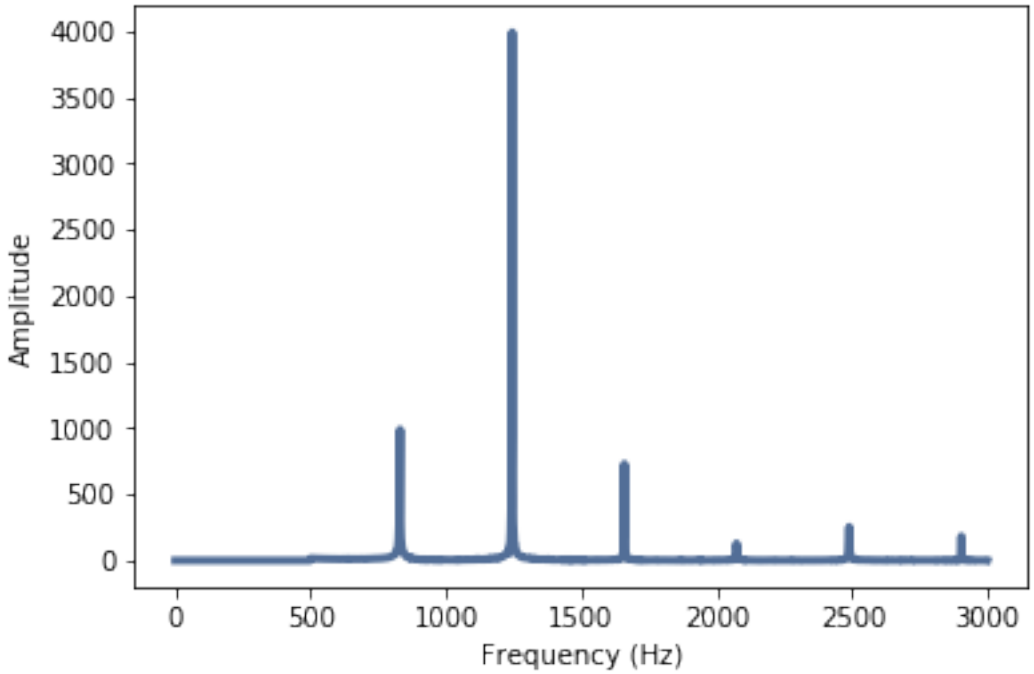
[32]:
```
416.0377358490566
```

Самый высокий пик находится на `lag` = 106, что соответствует частоте 416 Гц (тот же пик на спектре, что и 414 Гц).

По крайней мере в этом примере, высота, которую мы воспринимаем, соответствует самому высокому пику в автокорреляционной функции (ACF), а не самому высокому пику спектра.

Удивительно, но воспринимаемая высота не меняется даже если мы удаляем основную частоту. Используем фильтр верхних частот, после чего получим новый спектр:

[33]:
```
spectrum2 = segment.make_spectrum()
spectrum2.high_pass(500)
spectrum2.plot(high=3000)
thinkplot.config(xlabel='Frequency (Hz)', ylabel='Amplitude')
```



Получившийся сегмент:

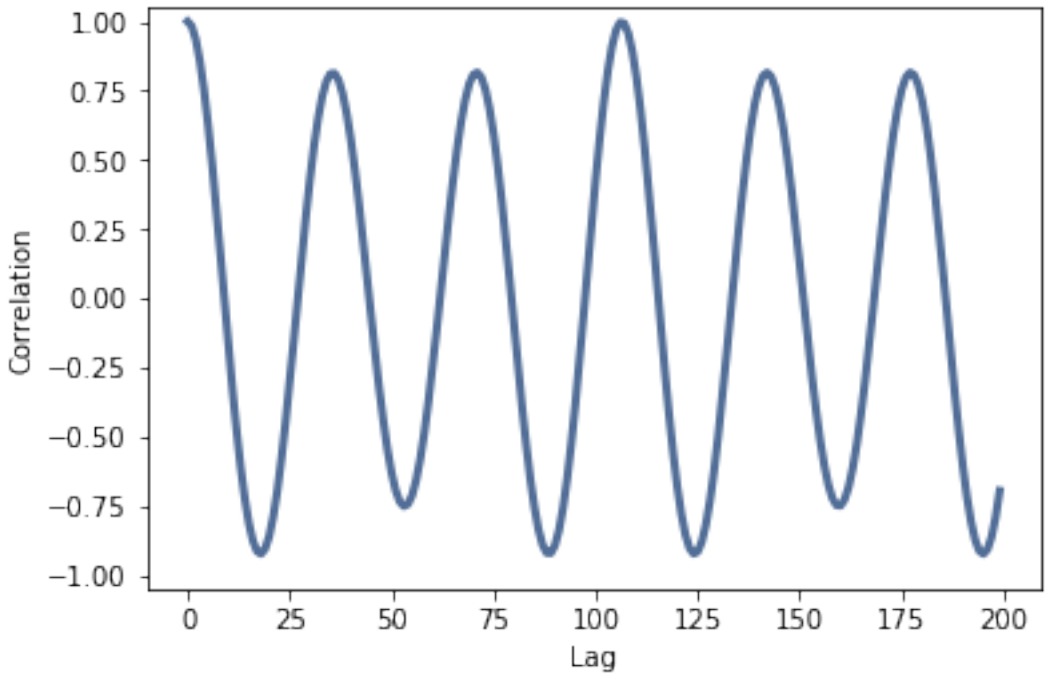
[34]:
```
segment2 = spectrum2.make_wave()
segment2.make_audio()
```

[34]:
```
<IPython.lib.display.Audio object>
```

Воспринимаемая высота звука по-прежнему составляет 414 Гц, хотя на этой частоте нет мощности. Это явление называется «подавленная основная».

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

[35]:
```
corrs = autocorr(segment2)
thinkplot.plot(corrs[:200])
thinkplot.config(xlabel='Lag', ylabel='Correlation', ylim=[-1.05, 1.05])
```



Пик, соответствующий 416 Гц, по-прежнему самый высокий.

[36]:
```
find_frequency(corrs, 100, 120)
```

```
106
```

[36]:
```
416.0377358490566
```

Но есть два других пика, соответствующих 1225 и 621 Гц.

[37]:
```
find_frequency(corrs, 25, 50)
```

```
36
```

[37]:
```
1225.0
```

[38]:
```
find_frequency(corrs, 50, 80)
```

```
71
```

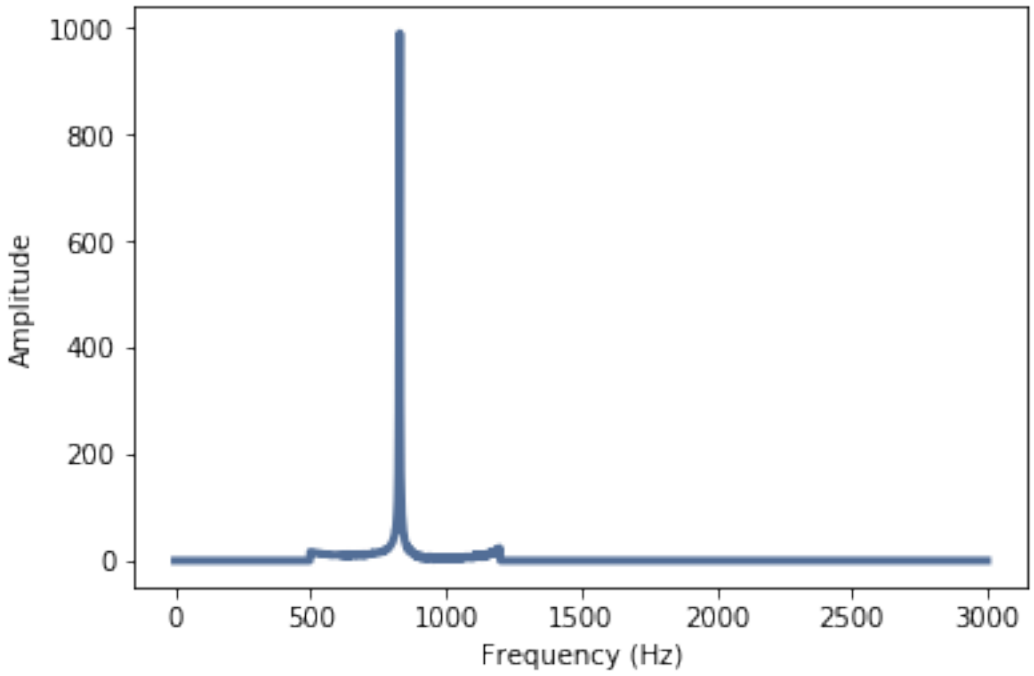
[38]:
```
621.1267605633802
```

Так почему же мы не воспринимаем ни одну из этих высот вместо 414 Гц? Причина в том, что высшие компоненты сигнала являются гармониками с частотой 414 Гц и не являются гармониками с частотой 621 или 1225 Гц.

Наше ухо интерпретирует высокие гармоники как свидетельство того, что «правильный» фундаментальный сигнал имеет частоту 414 Гц.

Если мы избавимся от высших гармоник, эффект исчезнет. Вот спектр с удаленными гармониками выше 1200 Гц:

[39]:
```
spectrum4 = segment.make_spectrum()
spectrum4.high_pass(500)
spectrum4.low_pass(1200)
spectrum4.plot(high=3000)
thinkplot.config(xlabel='Frequency (Hz)', ylabel='Amplitude')
```



Теперь воспринимаемая высота - 830 Гц.

[40]:
```
segment4 = spectrum4.make_wave()
segment4.make_audio()
```

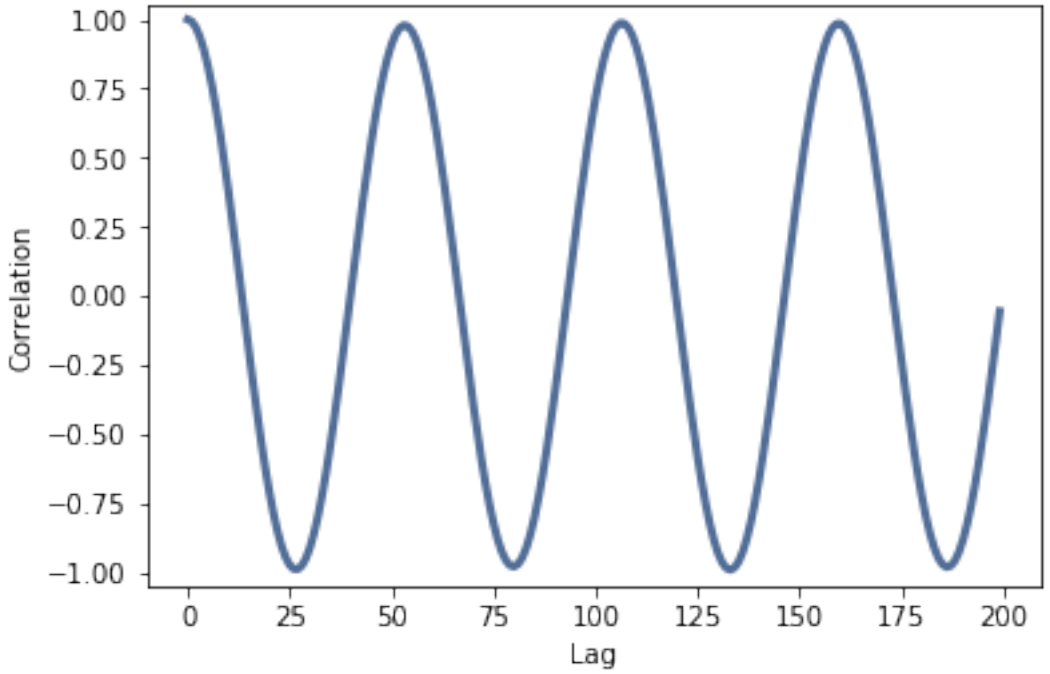
[40]:
```
<IPython.lib.display.Audio object>
```

[41]:
```
thinkdsp.TriangleSignal(freq=830).make_wave(duration=0.5).make_audio()
```

[41]:
```
<IPython.lib.display.Audio object>
```

И если мы посмотрим на функцию автокорреляции, мы найдем самый высокий пик при `lag` = 53, что соответствует 832 Гц.

[42]:
```
corrs = autocorr(segment4)
thinkplot.plot(corrs[:200])
thinkplot.config(xlabel='Lag', ylabel='Correlation', ylim=[-1.05, 1.05])
```



[43]:
```
find_frequency(corrs, 30, 60)
```

```
53
```

[43]:
```
832.0754716981132
```

Для удобства, все версии вместе:

Треугольный сигнал при 414 Гц:

[44]:
```
thinkdsp.TriangleSignal(freq=414).make_wave(duration=0.5).make_audio()
```

[44]:
```
<IPython.lib.display.Audio object>
```

Исходный сегмент:

[45]:
```
segment.make_audio()
```

[45]:
```
<IPython.lib.display.Audio object>
```

После удаления основной частоты:

[46]:
```
segment2.make_audio()
```

[46]:
```
<IPython.lib.display.Audio object>
```

После удаления высших гармоник:

[47]:
```
segment4.make_audio()
```

[47]:
```
<IPython.lib.display.Audio object>
```

Синусоидальный сигнал с частотой 832 Гц:

[48]:
```
thinkdsp.SinSignal(freq=928).make_wave(duration=0.5).make_audio()
```

[48]:
```
<IPython.lib.display.Audio object>
```

4 Вывод

В ходе выполнения данной лабораторной работы были рассмотрены понятия корреляццции, последовательной корреляции, автокорреляции. Была рассмотрена автокорреляционая функция и её практическое применение.