Санкт-Петербургский государственный политехнический
университет Петра Великого

**Высшая школа интеллектуальных систем и
суперкомпьютерных технологий**

Лабораторная работа №5

Автокорреляция

Выполнил студент 3-го курса
группа 3530901/80201
Матвеец Андрей Вадимович

Преподаватель:
Богач Наталья Владимировна

Санкт-Петербург

2021

Содержание

Список иллюстраций

Листинги

1 Часть №1: Оценка высоты тона вокального чирпа

В первом пункте лабораторной работы нам необходимо вычислить автокорреляцию для различных Lag и оценить высоты тона вокального чирпа.

Начнем с написания функции serial-corr и autocorr:

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

Листинг 1: Функция serial-corr

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

Листинг 2: Функция autocorr

После этого прочитам чирп с записью голоса и для проверки выведем его:

Рис. 1: Получение чирпа

После этого построим график автокорреляции:

```
segment = wave.segment(0, 0.01)
lags, corrs = autocorr(segment)
lagx = np.array(corrs[90:110]).argmax() + 90
thinkplot.plot(lags, corrs, color = 'blue')
thinkplot.config(xlabel='Lag', ylabel='Correlation')
```

Листинг 3: Построение графика автокорреляции

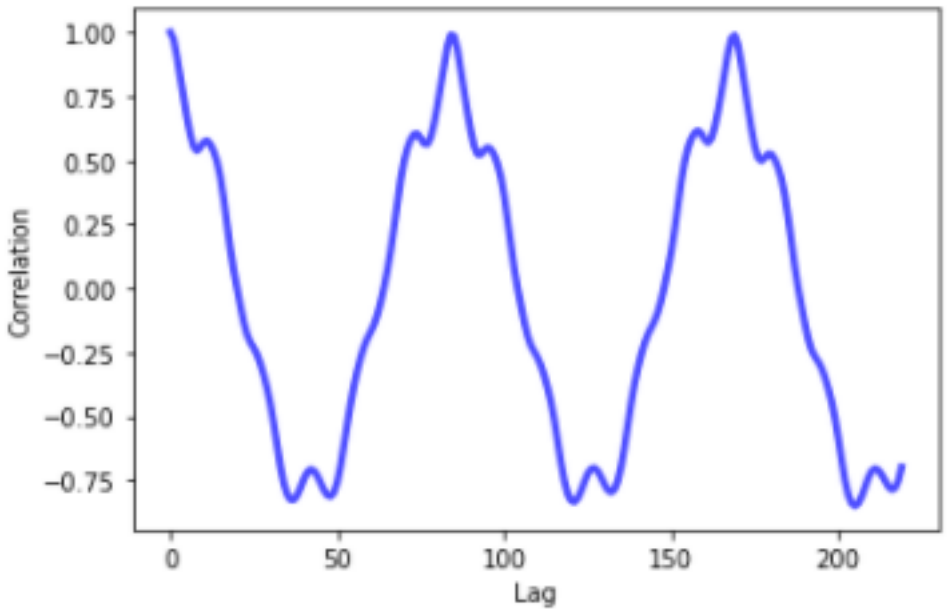


Рис. 2: Полученный график автокорреляции

По графику видно, что он периодический, период равен 90 `Lag`

2 Часть №2: Функция `estimate-fundamental`

Во втором пункте лабораторной рабооты нам необходимо написать функцию **`estimate-fundamental`**, отслеживающую высоту тона записанного звука. Также необходимо проверить ее работоспособность.

Для начала напишем функцию **`estimate-fundamental`**. которая с помощью функции автокорреляции может помочь в отслеживании высоты тона:

```
    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 4: Функция **`estimate-fundamental`**

После этого возьмем запись чирпа:

Рис. 3: Получение чирпа

Теперь построим спектограмму полученной записи:

```
    wave.make_spectrogram(2048).plot(high=4200)
    decorate(xlabel='Time (s)', ylabel='Frequency (Hz)')
```

Листинг 5: Построение спектограммы

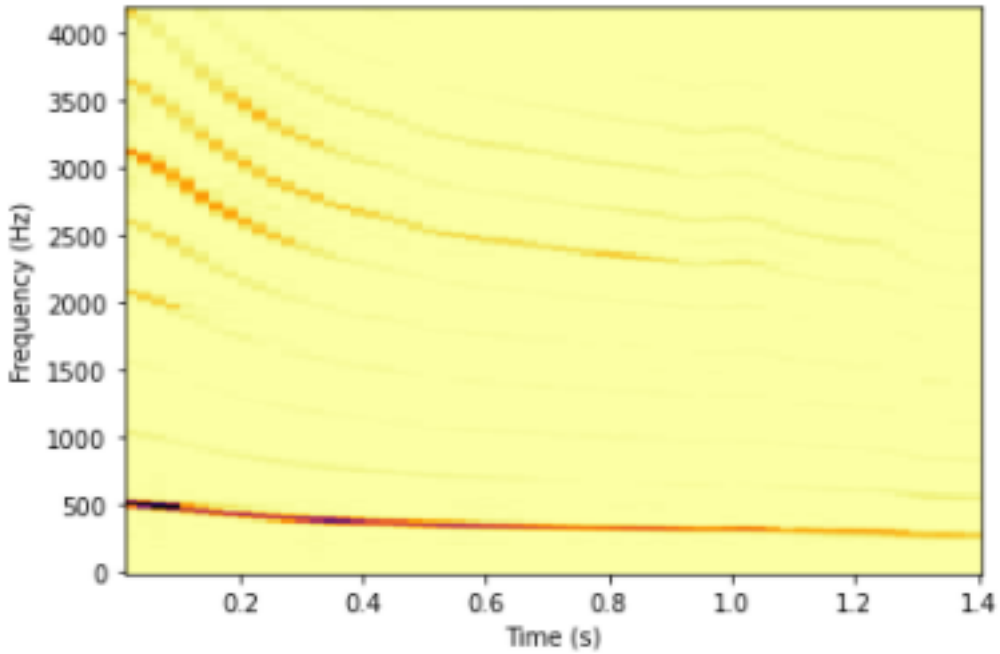


Рис. 4: Построенная спектограмма

Затем с помощью написанной ранее функции `estimate-fundamental` получим частоту минимального сегмента:

```
B [9]: segment = wave.segment(start=0.2, duration=0.01)
       freq = estimate_fundamental(segment)
       freq

Out[9]: 436.63366336633663
```

Рис. 5: Полученная частота минимального сегмента

После этого с шагом 0,05 найдем этот сегмент:

```
starts = np.arange(0.0, 1.4, 0.05)
ts = []
freqs = []
for start in starts:
    ts.append(start + 0.05/2)
    segment = wave.segment(start=start, duration=0.01)
    freq = estimate_fundamental(segment)
    freqs.append(freq)
```

Листинг 6: Нахождение нужного сегмента

И, наконец, выведем его спектограмму на экран:

```
wave.make_spectrogram(2048).plot(high=900)
plt.plot(ts, freqs, color='white')
decorate(xlabel='Time (s)', ylabel='Frequency (Hz)')
```

Листинг 7: Вывод спектограммы сегмента

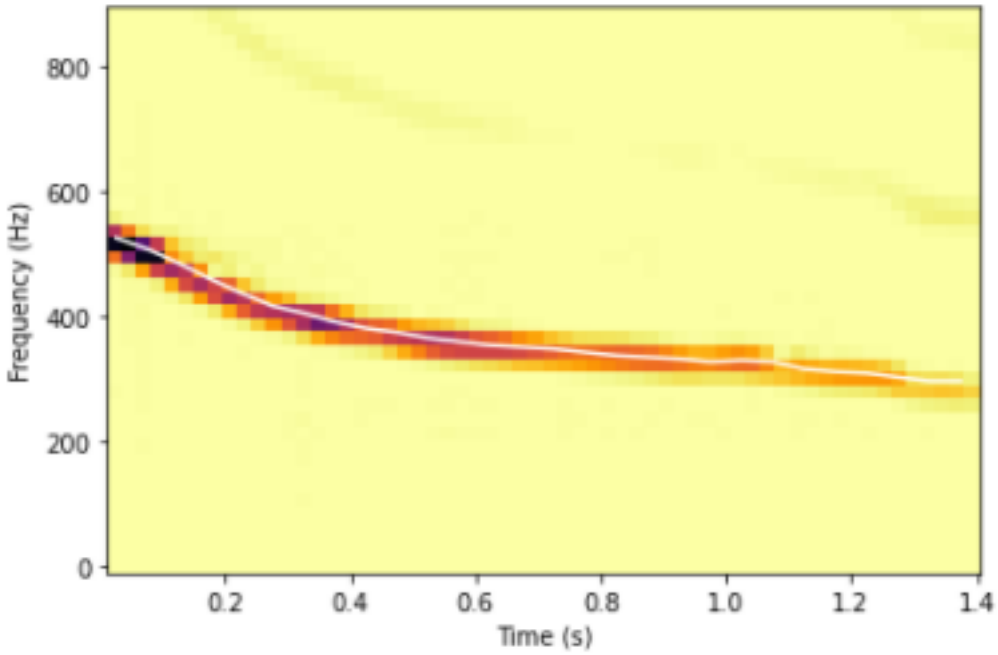


Рис. 6: Построенная спектограмма сегмента

В результате на полученной спектограмме сегмента можно увидеть частоту тона в каждый момент времени

3 Часть №3: BitCoin

В третьем пункте лабораторной работы нам необходимо использую данные цен BitCoin из прошлой лабораторной работы вычислить автокорреляцию цен на BitCoin.

Возмем тот же файл, что и в прошлой лабораторной работе и так же представим данные из него в виде графика:

```
1     df = pd.read_csv('BTC_USD_2013-10-01_2020-03-26-CoinDesk.csv', parse_dates
      =[0])
2     ys = df['Closing Price (USD)']
3     ts = df.index
4     wave = Wave(ys, ts, framerate=1)
5     wave.plot()
6     decorate(xlabel='Time (days)', ylabel='Price of BitCoin ($)')
```

Листинг 8: Представление данных файла в виде графика

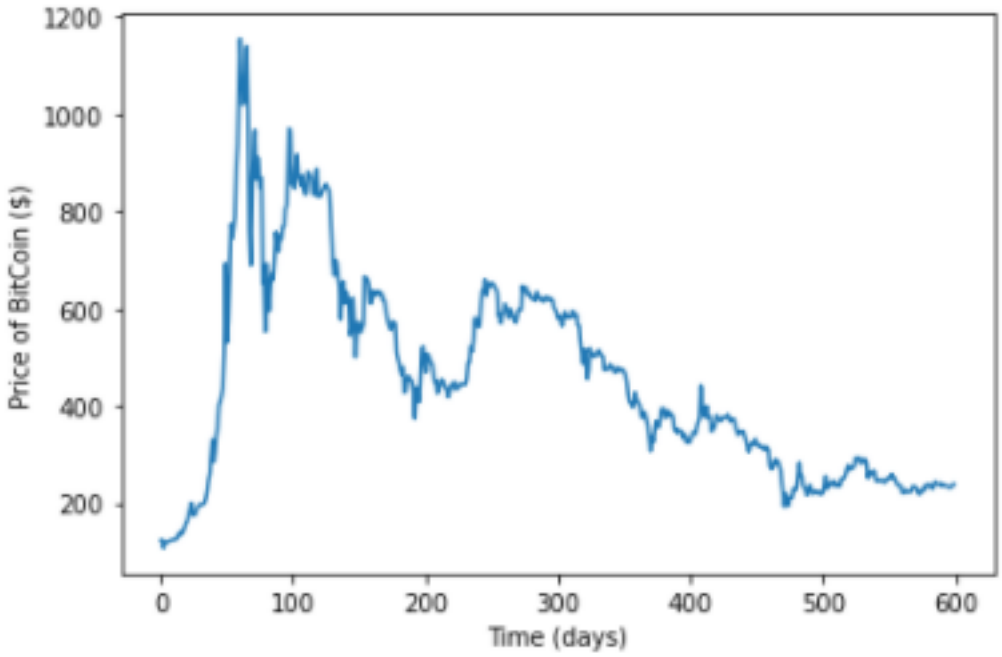


Рис. 7: Данные из файла в виде графика

После этого с помощью функции автокорреляции получим еще один график:

```
1     lags, corrs = autocorr(wave)
2     thinkplot.plot(lags, corrs)
3     decorate(xlabel='Lag', ylabel='Correlation')
```

Листинг 9: Получение нового графика после автокорреляции

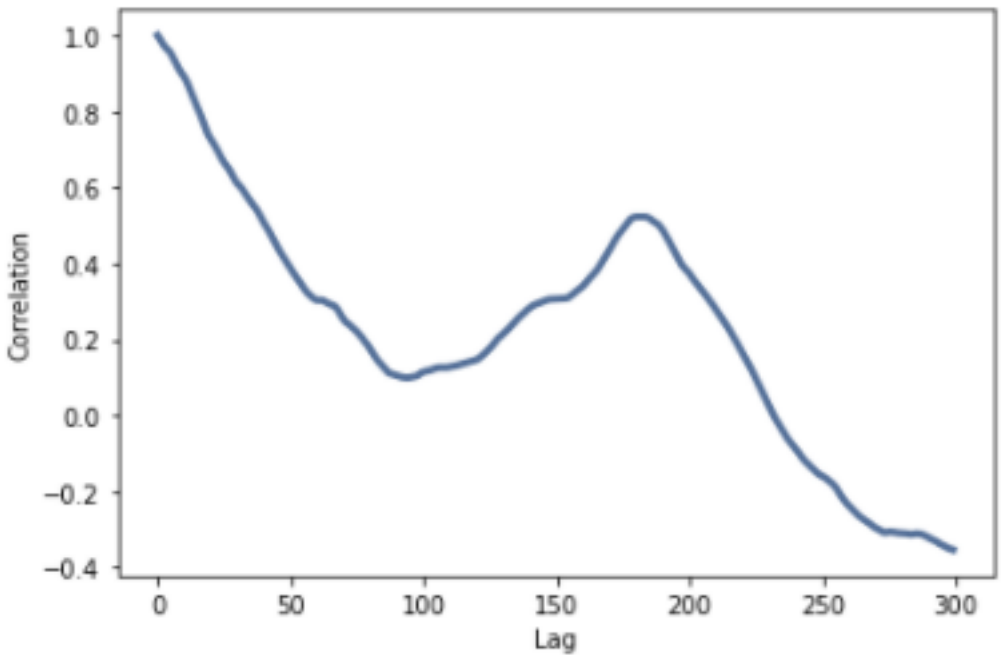


Рис. 8: Полученный график после автокорреляции

В результате можно сказать о том, что полученный график повторяет график цены `BitCoin`. Периодичность в графике не наблюдается.

4 Часть №4: `saxophone.ipynb`

В чертвертом пункте лабораторной работы нам необходимо прочитать блокнот `saxophpne.ipynb`, пройтись по всем примерам, затем выбрать другой сегмент записи и поработать с ним.

Будем использовать звук саксофона, который уже был использован:

```
In [14]: wave = read_wave('100475__iluppai__saxophone-weep.wav')
         wave.normalize()
         wave.make_audio()
```

Out[14]: ▶ 0:00 / 0:11

Рис. 9: Получение звука саксофона

После этого построим спектограмму данного звука:

```
1  gram = wave.make_spectrogram(seg_length=1024)
2  gram.plot(high=3000)
3  decorate(xlabel='Time (s)', ylabel='Frequency (Hz)')
```

Листинг 10: Построение спектограммы

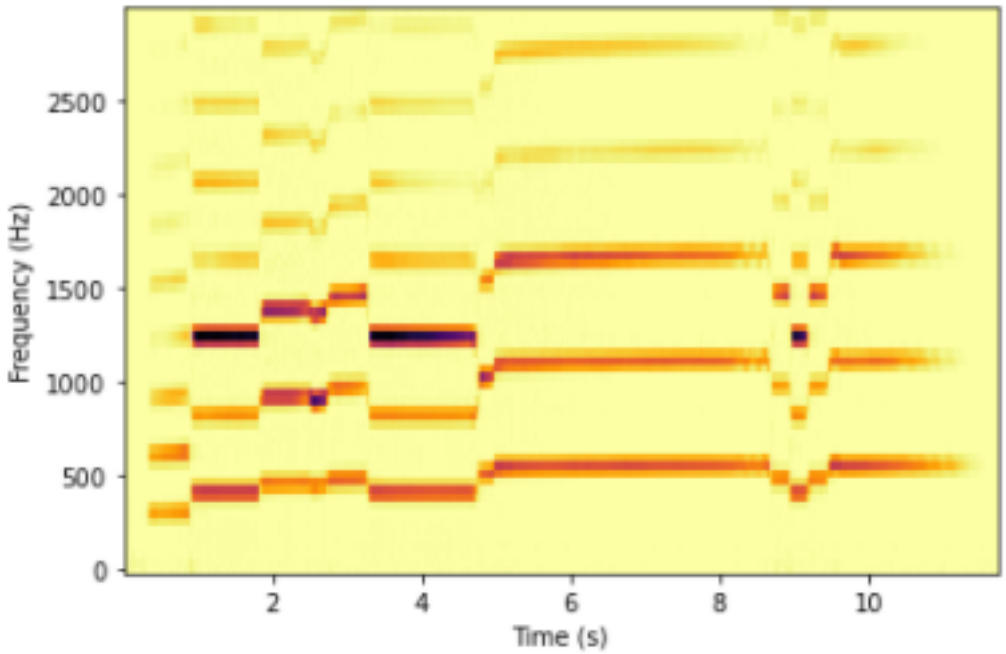


Рис. 10: Полученная спектограмма

После этого выделим сегмент из исходной записи с 6 секунды длительностью 0,5 секунды:

```
B [16]: segment = wave.segment(start=6.0, duration=0.5)
        segment.make_audio()
```

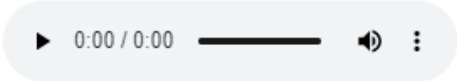
Out[16]: 

Рис. 11: Получение сегмента из записи

Далее получим спектр из данного сегмента:

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

Листинг 11: Получение спектра сегмента

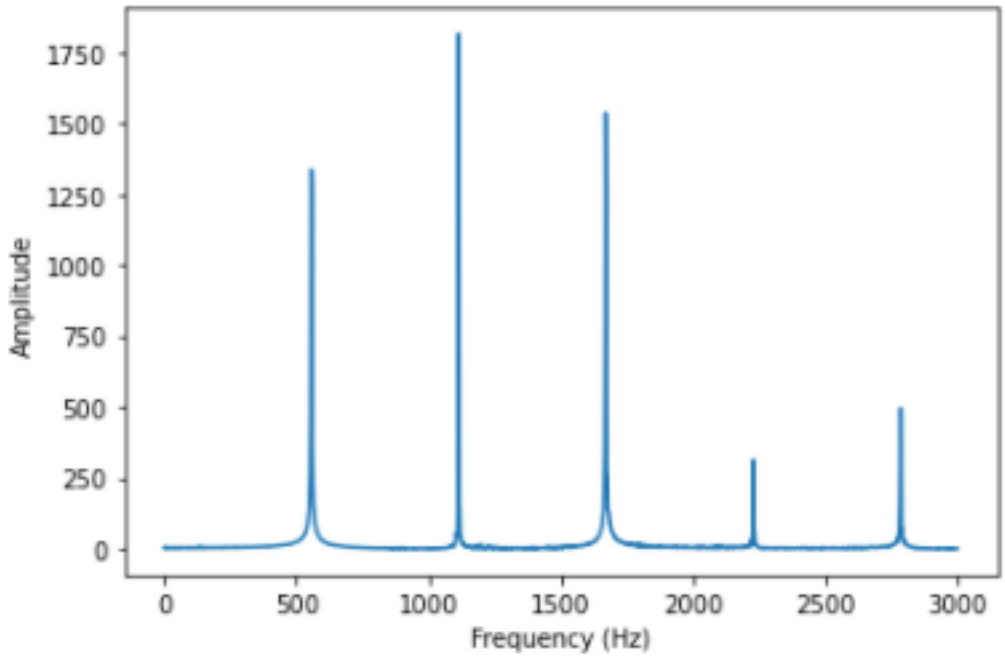


Рис. 12: Полученный спектр сегмента

Сразу же получим все пики полученного спектра:

```
B [18]: spectrum.peaks()[:10]
```

```
Out[18]: [(1815.189472282375, 1114.0),
          (1536.1422239239648, 1670.0),
          (1337.2214321092208, 556.0),
          (1276.178380563102, 1672.0),
          (1274.2203137022834, 558.0),
          (706.3189084856051, 1668.0),
          (498.87612764141124, 554.0),
          (496.1139335893734, 2784.0),
          (375.3414270174492, 2786.0),
          (366.17346455840516, 560.0)]
```

Рис. 13: Пики полученного спектра

Для сравнения получим треугольный сигнал той же частоты:

```
B [19]: TriangleSignal(freq=560).make_wave(duration=0.5).make_audio()
Out[19]:
```

Рис. 14: Треугольный сигнал

Теперь воспользуемся автокорреляцией для нашего сегмента. Для этого сначала напишем функцию:

```
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 12: Функция для автокорреляции

Теперь обратимся к этой функции и выведем на экран полученный график:

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

Листинг 13: Получение графика автокорреляции

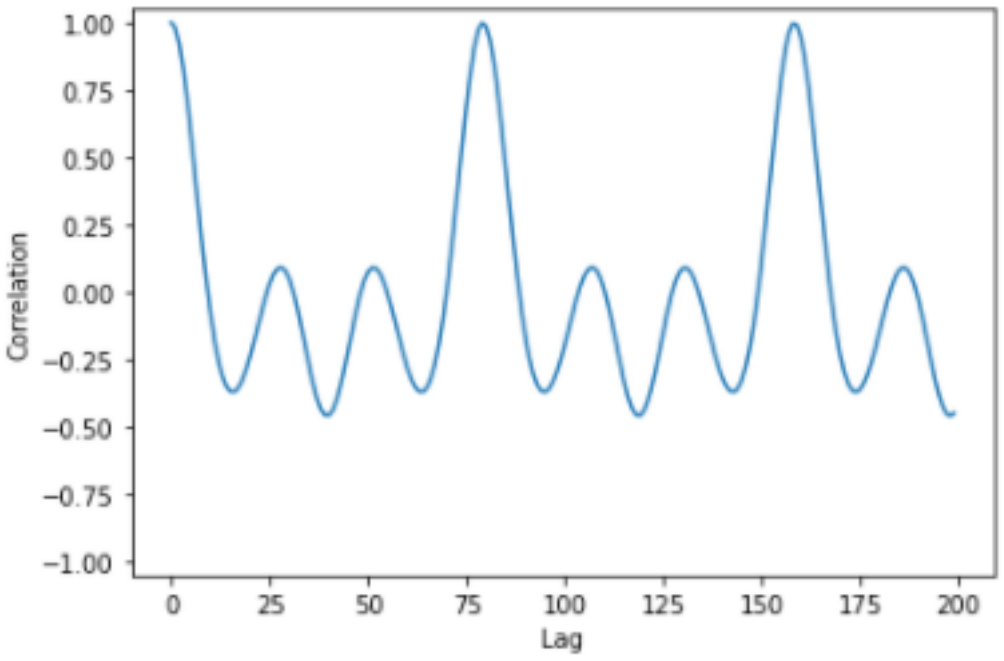


Рис. 15: Полученный график автокорреляции

По графику видно, что первый наибольший пик начинается приверно около `lag` = 65. Для нахождения частоты в данном `lag` напишем функцию:

```
    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
```

Листинг 14: Функция для нахождения частоты

После этого вызовем ее, передав на вход примерные значения начала и конца `lag`:

```
B [23]: find_frequency(corrs, 65, 95)
        79
Out[23]: 558.2278481012657
```

Рис. 16: Использование написанной функции

Как можно увидеть, пик находится на `lag` = 79

Теперь нам необходимо отфильтровать наш сегмент через фильтр низких частот:

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

Листинг 15: Фильтрация сегмента

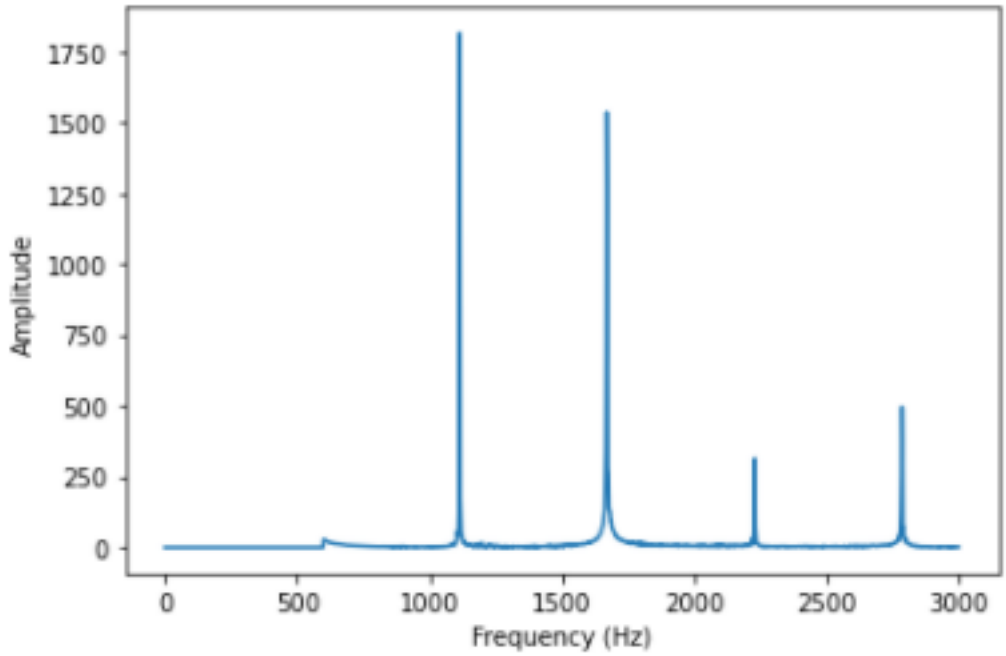


Рис. 17: Полученный график после фильтрации

Как можно увидеть по графику, главная частота была убрана.

Теперь нам осталось преобразовать сегмент в аудио для сравнения с исходным сегментом:

```
B [25]: segment2 = spectrum2.make_wave()
        segment2.make_audio()
Out[25]:
```

Рис. 18: Получение аудио из отфильтрованного сегмента

Наконец, прослушав исходный сегмент и отфильтрованный можно сказать, что эти сигналы очень похожи, но отфильтрованный звучит более приглушённо.

5 Выводы

В результате выполнения лабораторной работы мы изучили, что такое автокорреляция, научились вычислять ее для различных`lag`. Была создана функция `estimate-fundamental` для отслеживания высоты тона звука. Также вычислили автокорреляцию цен за `BitCoin` из прошлой лабораторной работы. Наконец, мы поработали с сегментами записи звуков саксофона и вычислили для них автокорреляцию.