Лабораторная работа № 8. Фильтрация и свертка.

3530901/80201, Шелаев Н. Р.

31 мая 2021 г.

Оглавление

Список иллюстраций

Листинги

Глава 1

Сглаживание

Это операция, удаляющая быстрые изменения сигнала для выявления общих зависимостей. Попробуем это на чём-нибудь применить.

```
1       import pandas as pd
2
3       df = pd.read_csv('FB_2.csv', header=0, parse_dates=[0])
4       close = df['Close']
5       dates = df['Date']
6       days = (dates - dates[0]) / np.timedelta64(1,'D')
7       M = 30
8       window = np.ones(M)
9       window /= sum(window)
10      smoothed = np.convolve(close, window, mode='valid')
11      smoothed_days = days[M // 2:(len(smoothed) + M // 2)]
12      plt.plot(days, close, color = 'gray', alpha = 0.6,
    label = 'daily close')
13      plt.plot(smoothed_days, smoothed, color = 'C1', alpha =
     0.6, label = '30 day average')
14
```

Листинг 1.1: Воспользуемся акциями Facebook

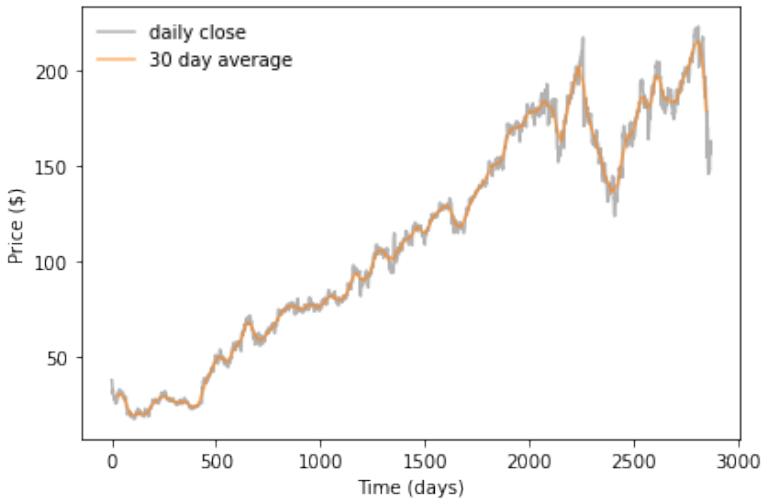

Рис. 1.1: Результат работы сглаживания

```
from thinkdsp import SawtoothSignal

signal = SawtoothSignal(freq = 440)
wave = signal.make_wave(duration=1.0, framerate=44100)
window = np.ones(11)
window /= sum(window)
plt.plot(window)
segment = wave.segment(duration = 0.01)
segment.plot()

```

Листинг 1.2: А теперь реальный сигнал

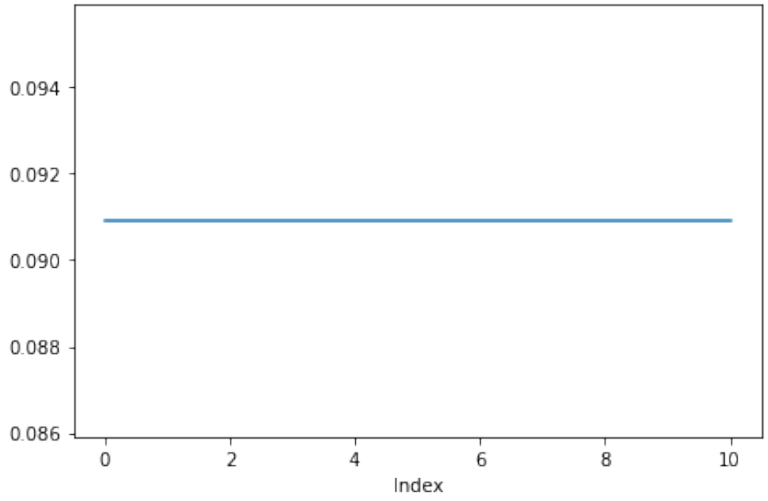

Рис. 1.2: Окно сигнала

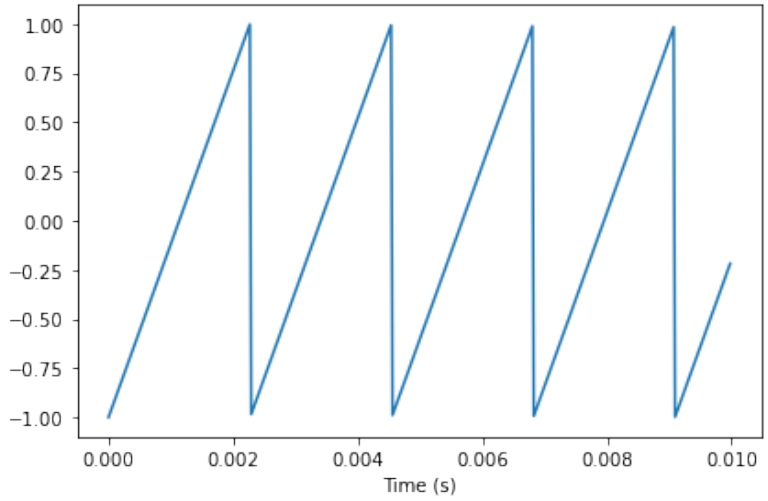

Рис. 1.3: Сегмент пилообразного сигнала

```
def zero_pad(array, n):
    res = np.zeros(n)
    res[:len(array)] = array
    return res

N = len(segment)
padded = zero_pad(window, N)
plt.plot(padded)
```

Листинг 1.3: Сделали нужную ширину окна

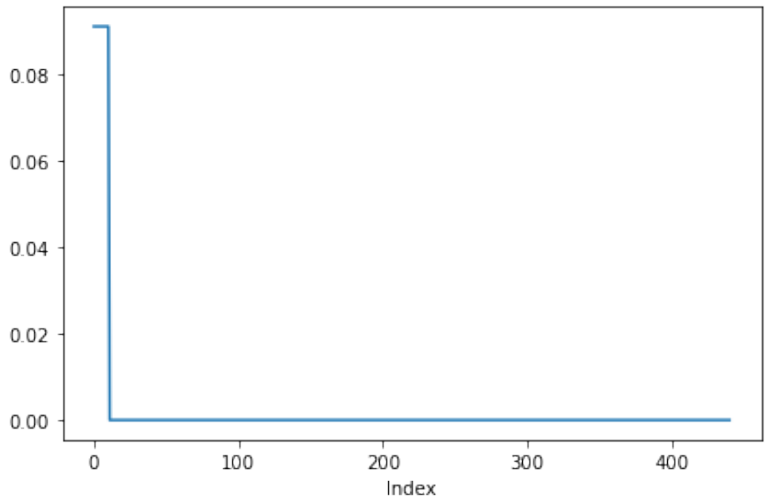

Рис. 1.4: Изменили ширину окна

```
prod = padded * segment.ys
smoothed = np.zeros(N)
rolled = padded.copy()
for i in range(N):
    smoothed[i] = sum(rolled * segment.ys)
    rolled = np.roll(rolled, 1)

plt.plot(rolled)

```

Листинг 1.4: Сдвинули окно вправо

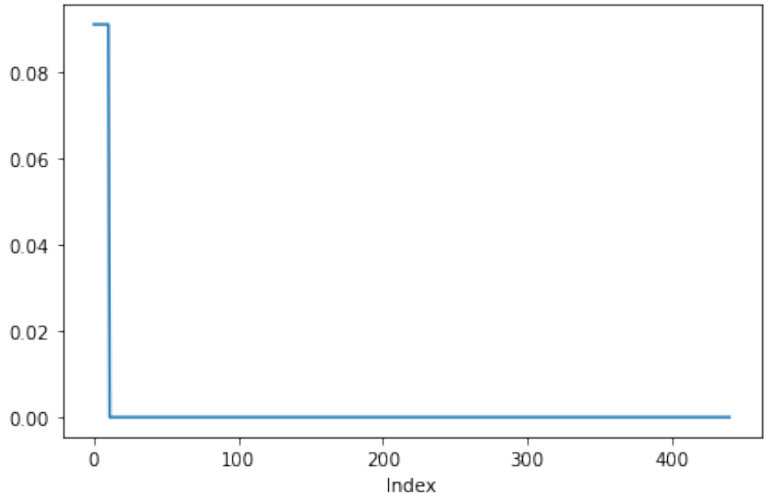

Рис. 1.5: Сдвиг окна

```
from thinkdsp import Wave

segment.plot(color = 'gray')
smooth = Wave(smoothed, framerate = wave.framerate)
smooth.plot()

```

Листинг 1.5: Сравнение с исходным сигналом

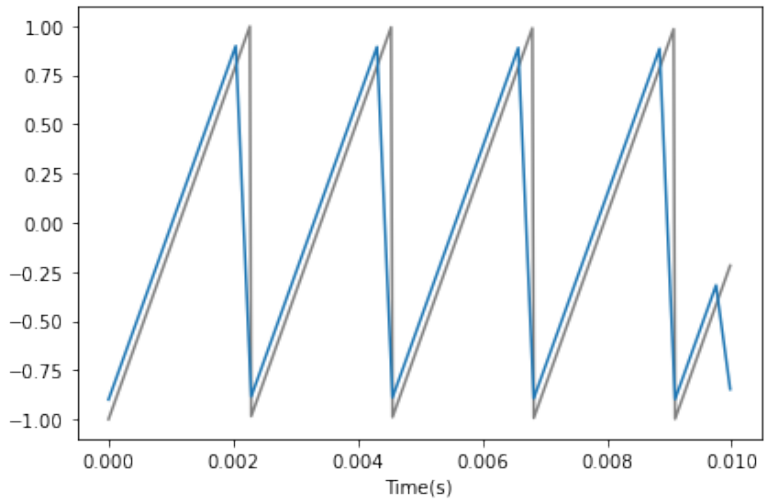

Рис. 1.6: В правой части что-то не так

```
segment.plot(color = 'gray')
ys = np.convolve(segment.ys, window, mode = 'valid')
smooth2 = Wave(ys, framerate = wave.framerate)
smooth2.plot()

```

Листинг 1.6: Встроенная функция свертки

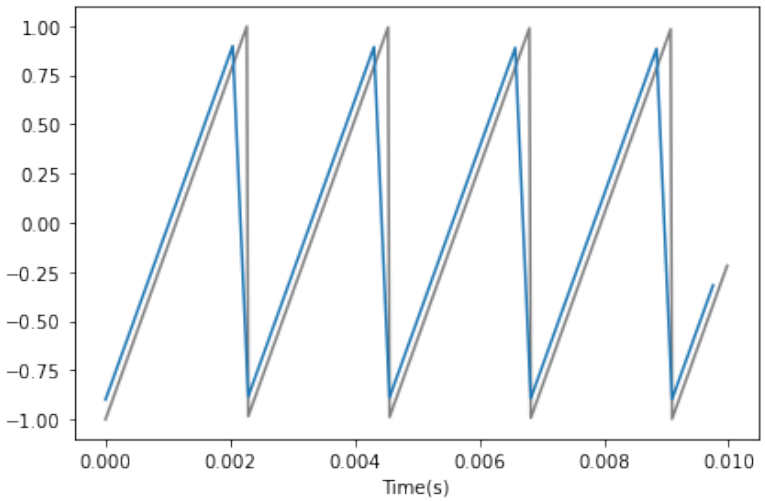

Рис. 1.7: Так-то лучше

Глава 2

Частотная область

Как операция сглаживания влияет на спектр сигнала?

```
convolved = np.convolve(wave.ys, window, mode = 'same')
smooth = Wave(convolved, framerate = wave.framerate)
spectrum = wave.make_spectrum()
spectrum.plot(color = 'gray')
spectrum2 = smooth.make_spectrum()
spectrum2.plot()

```

Листинг 2.1: Сравнение исходного и сглаженного сигнала

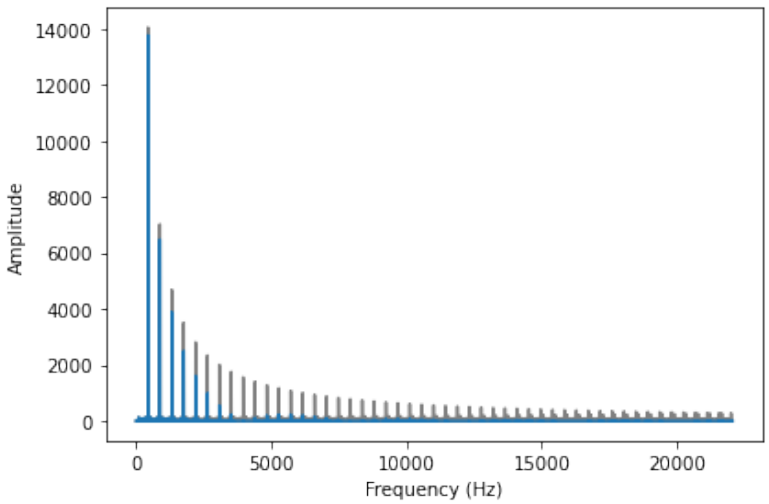

Рис. 2.1: Результат сглаживания

```
amps = spectrum.amps
amps2 = spectrum2.amps
```

```
3       ratio = amps2 / amps
4       ratio[amps < 280] = 0
5       padded = zero_pad(window, len(wave))
6       dft_window = np.fft.rfft(padded)
7       plt.plot(np.abs(dft_window), color = 'gray', label = '
    DFT(window)')
8       plt.plot(ratio, label = 'amplitude ratio')
9
```

Листинг 2.2: Соотношение спектров сигнала до и после сглаживания

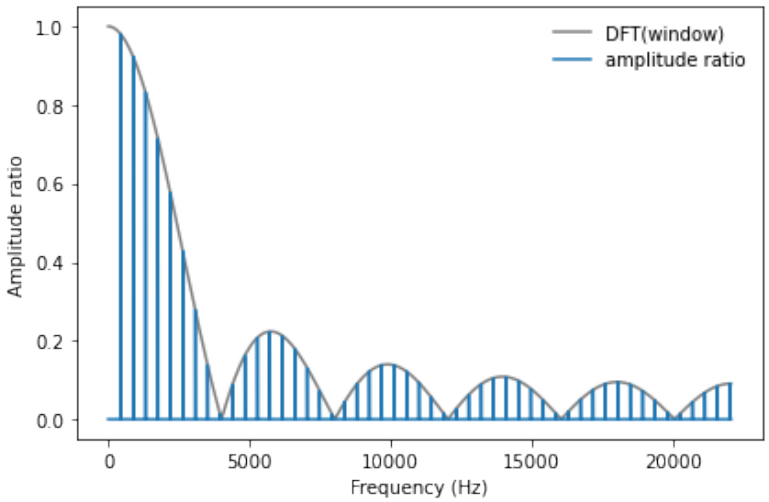

Рис. 2.2: График соотношения спектров до и после сглаживания

Глава 3

Гауссов фильтр

Протестируем работу гауссова филлтра на различных сигналах.

```
1       import scipy.signal
2
3       boxcar = np.ones(11)
4       boxcar /= sum(boxcar)
5       gaussian = scipy.signal.gaussian(M = 11, std = 2)
6       gaussian /= sum(gaussian)
7       plt.plot(boxcar, label = 'boxcar')
8       plt.plot(gaussian, label = 'Gaussian')
9
```

Листинг 3.1: Прямоугольное и гауссово окна

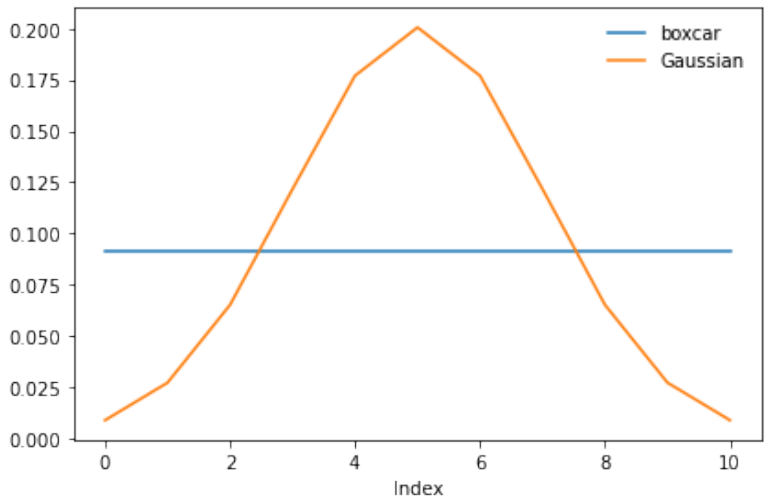

Рис. 3.1: Результат сравнения двух окон

```
    from thinkdsp import SquareSignal

    def plot_filter(M = 11, std = 2):
        signal = SquareSignal(freq = 440)
        wave = signal.make_wave(duration=1, framerate=44100)
        spectrum = wave.make_spectrum()
        gaussian = scipy.signal.gaussian(M = M, std = std)
        gaussian /= sum(gaussian)
        ys = np.convolve(wave.ys, gaussian, mode = 'same')
        smooth = Wave(ys, framerate = wave.framerate)
        spectrum2 = smooth.make_spectrum()
        amps = spectrum.amps
        amps2 = spectrum2.amps
        ratio = amps2 / amps
        ratio[amps < 560] = 0
        padded =  zero_pad(gaussian, len(wave))
        dft_gaussian = np.fft.rfft(padded)
        plt.plot(np.abs(dft_gaussian), color = 'gray', label
= 'Gaussian filter')
        plt.plot(ratio, label = 'amplitude ratio')

    slider = widgets.IntSlider(min=2, max=100, value=11)
    slider2 = widgets.FloatSlider(min=0, max=20, value=2)
    interact(plot_filter, M = slider, std = slider2);

```

Листинг 3.2: Собрали всё в одну функцию

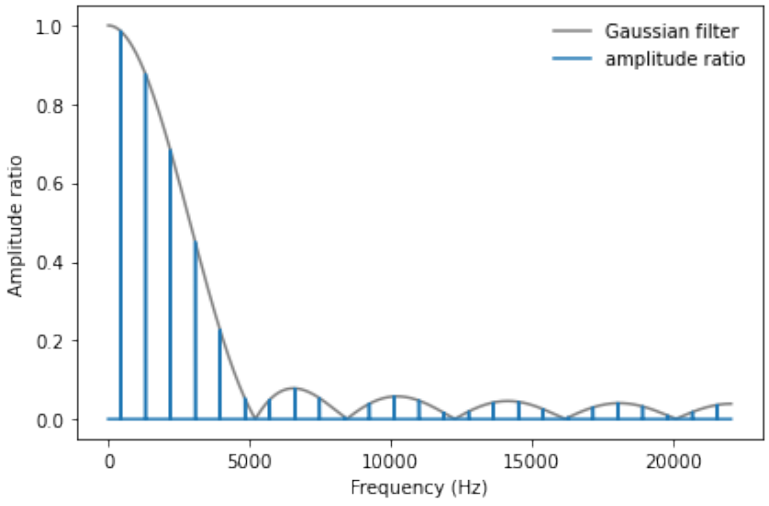

Рис. 3.2: Исследование зависимости результата от значений M и std

При увеличении `std` без увеличения `M` можно заметить, что «скачки»

снова появляются, как если бы мы взяли `Boxcar`-фильтр на `M` элементов.

Глава 4

Эффективная свертка

Улучшаем алгоритм свертки.

```
window = scipy.signal.gaussian(M = 30, std = 6)
window /= window.sum()
smoothed = np.convolve(close, window, mode = 'valid')
plt.plot(close, color = 'gray')
plt.plot(smoothed)

```

Листинг 4.1: Исходная свертка

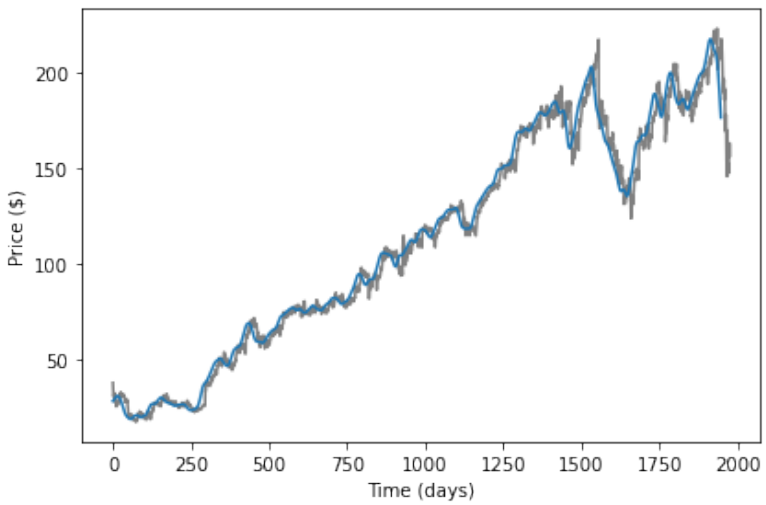

Рис. 4.1: Надо улучшить!

```
N = len(close)
padded = zero_pad(window, N)
fft_window = np.fft.fft(padded)
```

```
4         plt.plot(np.abs(fft_window))
5         fft_signal = np.fft.fft(close)
6         smoothed2 = np.fft.ifft(fft_signal * fft_window)
7         M = len(window)
8         smoothed2 = smoothed2[M-1:]
9         plt.plot(smoothed)
10        plt.plot(smoothed2.real)
11
```

Листинг 4.2: Улучшаем алгоритм свертки

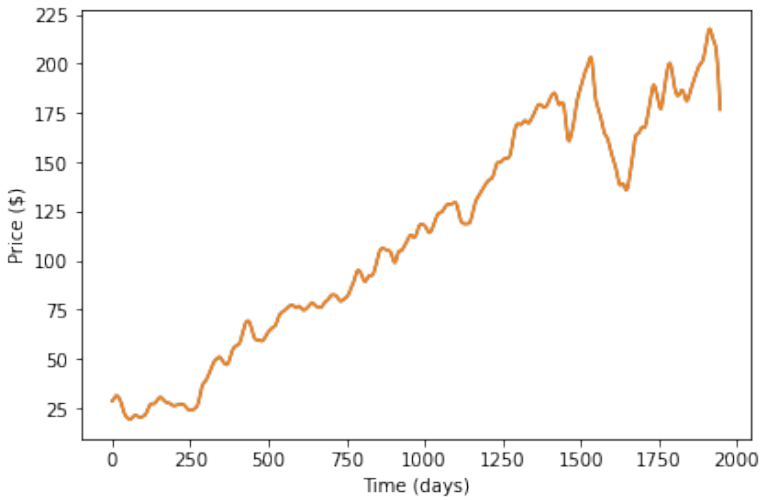

Рис. 4.2: Почти полное совпадение!

Сравнили этот алгоритм с встроенным алгоритмом `np.fft.fft`. Разница между результатами близка к 0.

Глава 5

Эффективная автокорреляция

А теперь улучшим алгоритм автокорреляции.

```
1       corrs = np.correlate(close, close, mode = 'same')
2
3       def fft_autocorr(signal):
4          N = len(signal)
5          signal = zero_pad(signal, 2 * N)
6          window = np.flipud(signal)
7          corrs = fft_convolve(signal, window)
8          corrs = np.roll(corrs, (N // 2 + 1))[:N]
9          return corrs
10
11      corrs2 = fft_autocorr(close)
12      lags = np.arange(N) - N // 2
13      plt.plot(lags, corrs, color = 'gray', alpha = 0.5,
    label = 'np.convolve')
14      plt.plot(lags, corrs2.real, color = 'C1', alpha = 0.5,
    label = 'fft_convolve')
15
```

Листинг 5.1: Улучшенная автокорреляция

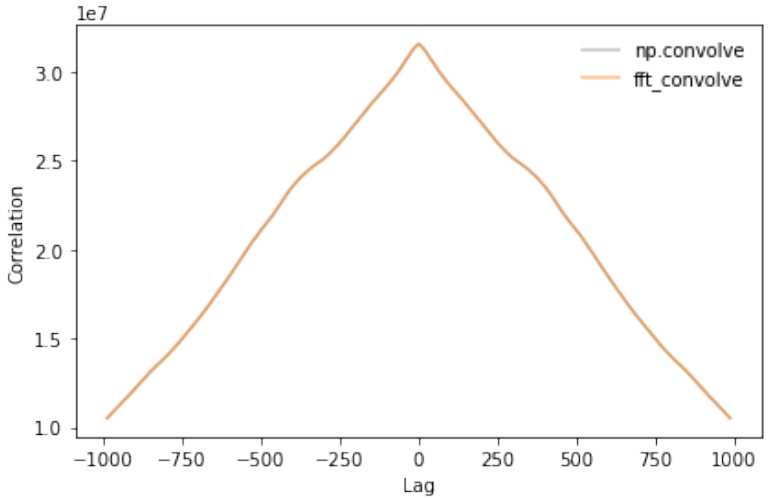

Рис. 5.1: Сравнение двух алгоритмов

Разница между этими двумя алгоритмами близка к 0.

Глава 6

Упражнения

6.1 Задание 2

Исследование преобразования Фурье над сигналами нормального распределения.

```
1       import scipy.signal
2
3       gaussian = scipy.signal.gaussian(M = 32, std = 2)
4       gaussian /= sum(gaussian)
5       plt.plot(gaussian)
6       fft_gaussian = np.fft.fft(gaussian)
7       plt.plot(abs(fft_gaussian))
8
```

Листинг 6.1: Получение сигнала

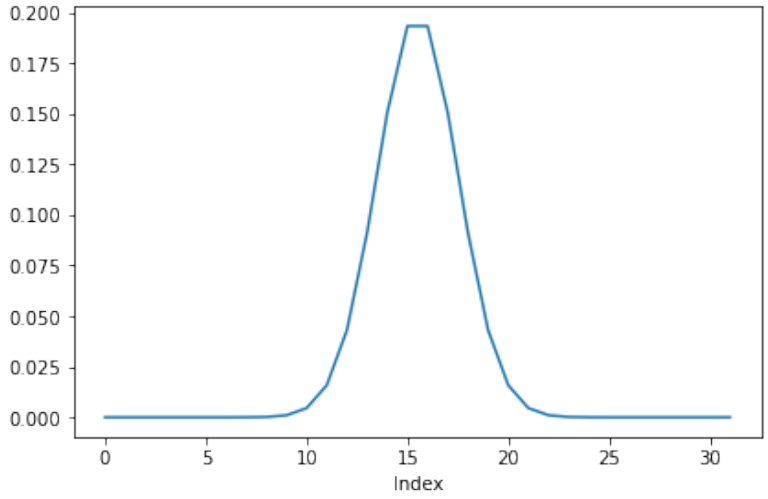

Рис. 6.1: Исходный сигнал

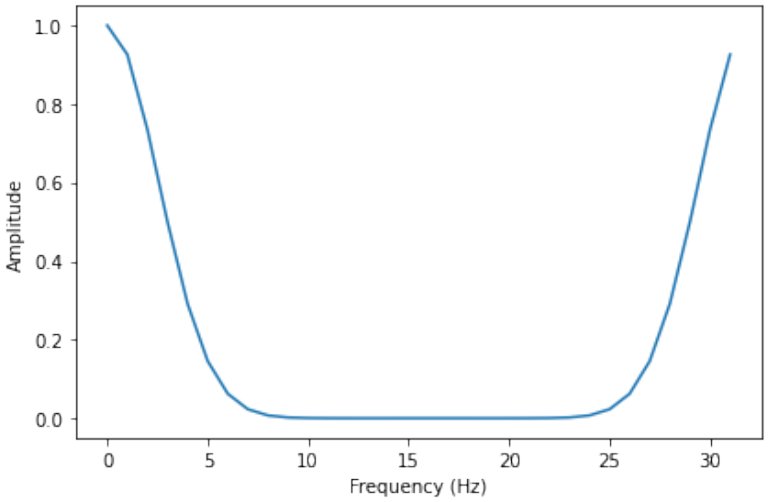

Рис. 6.2: Применение функции FFT

```
N = len(gaussian)
fft_rolled = np.roll(fft_gaussian, N // 2)
plt.plot(abs(fft_rolled))

```

Листинг 6.2: Сдвинули отрицательные частоты влево

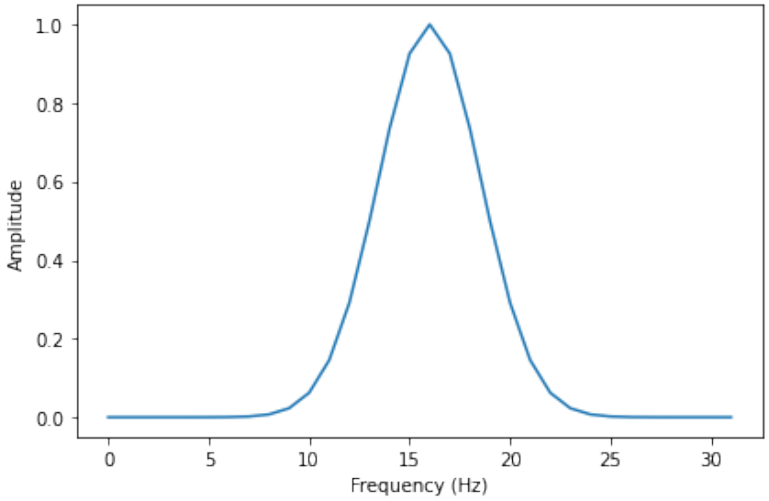

Рис. 6.3: Немного измененный сигнал

```
def plot_gaussian(std):
    M = 32
    gaussian = scipy.signal.gaussian(M = M, std = std)
    gaussian /= sum(gaussian)
    plt.subplot(1, 2, 1)
    plt.plot(gaussian)
    decorate(xlabel = 'Time')
    fft_gaussian = np.fft.fft(gaussian)
    fft_rolled = np.roll(fft_gaussian, M // 2)
    plt.subplot(1, 2, 2)
    plt.plot(np.abs(fft_rolled))

slider = widgets.FloatSlider(min=0.1, max=10, value=2)
interact(plot_gaussian, std = slider);

```

Листинг 6.3: Исследование влияния значения std

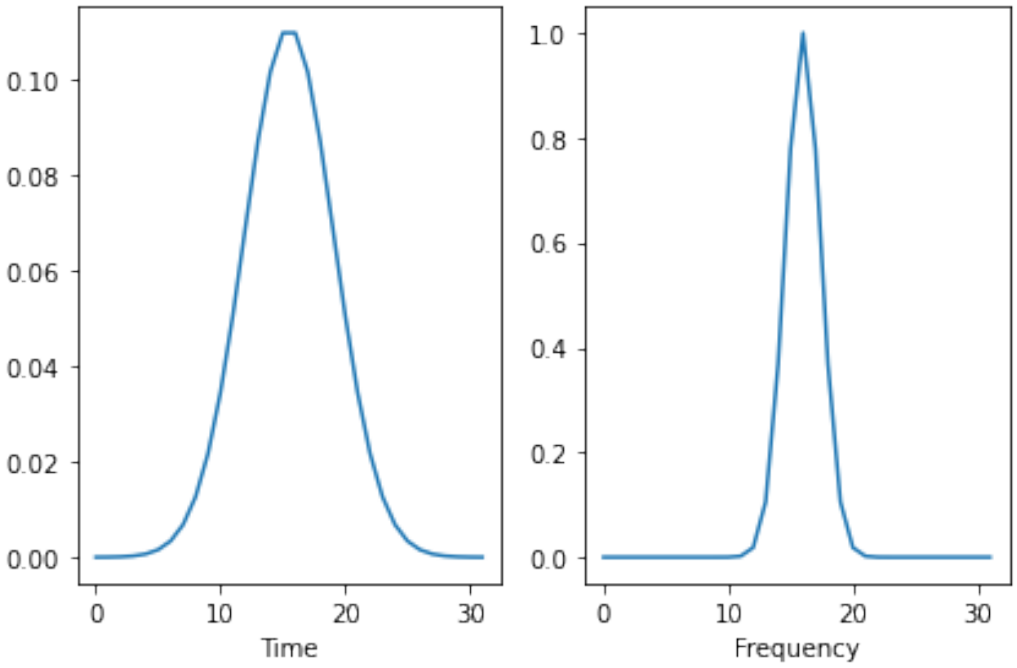

Рис. 6.4: Изменяем значение std

При увеличении значения `std` левый «колокол» расширяется, а правый - сужается, и наоборот.

6.2 Задание 3

Более подробно исследуем различные окна.

```
from thinkdsp import SquareSignal

signal = SquareSignal(freq = 440)
wave = signal.make_wave(duration=1.0, framerate=44100)
M = 25
std = 5.0
gaussian = scipy.signal.gaussian(M = M, std = std)
bartlett = np.bartlett(M)
blackman = np.blackman(M)
hamming = np.hamming(M)
hanning = np.hanning(M)
windows = [blackman, gaussian, hanning, hamming]
names = ['blackman', 'gaussian', 'hanning', 'hamming']

for window in windows: window /= sum(window)

```

```
    for window, name in zip(windows, names):
        plt.plot(window, label=name)

```

Листинг 6.4: Начало исследования

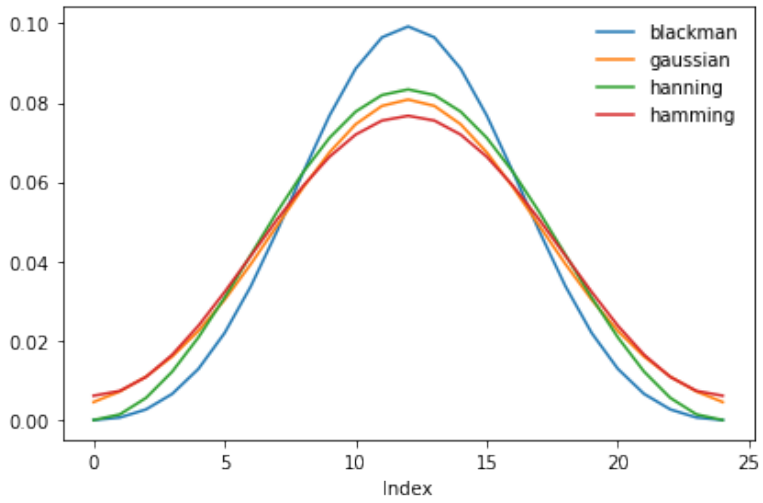

Рис. 6.5: Применение окон к исходному сигналу

```
def zero_pad(array, n):
    res = np.zeros(n)
    res[:len(array)] = array
    return res

def plot_window_dfts(windows, names):
    for window, name in zip(windows, names):
        padded = zero_pad(window, len(wave))
        dft_window = np.fft.rfft(padded)
        plt.plot(abs(dft_window), label = name)

plot_window_dfts(windows, names)

```

Листинг 6.5: Применение функции DFT

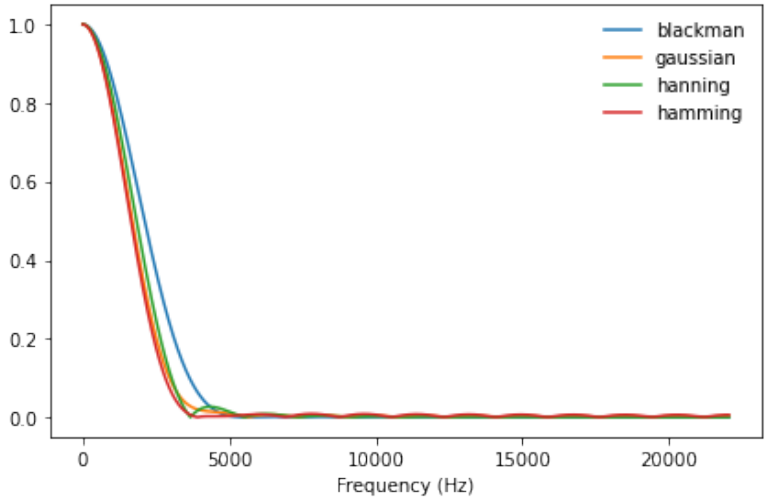

Рис. 6.6: Результаты сравнения

Все функции очень похожи, однако, можно отметить, что функция Хэмминга падает быстрее всех, а Блэкмана - медленнее всех.

```
1       plot_window_dfts(windows, names)
2       decorate(xlabel = 'Frequency (Hz)', yscale = 'log')
3
```

Листинг 6.6: Логарифмический масштаб

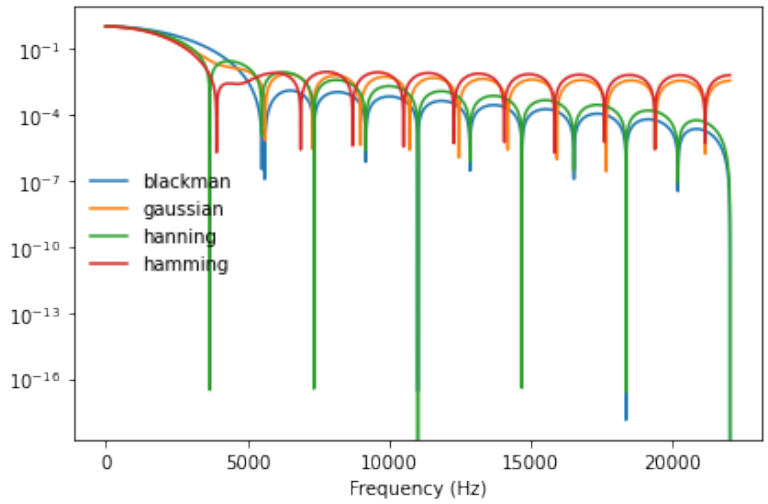

Рис. 6.7: Результаты сравнения в логарифмическом масштабе

В этом масштабе мы заметили, что функции Хэмминга и Хамминга пада-

ют быстрее всех. Окна Хэмминга и Гаусса имеют самые стойкие боковые лепестки.

Глава 7

Вывод

В данной работе мы рассмотрели схлаживание сигналов, улучшили алгоритмы свертки и автокорреляции. В последнем упражнении сравнили работу 4-х различных окон.