به نام هستی بخش

واحد مشهد

دانشکده فنی و مهندسی، گروه کامپیوتر

درس پردازش زبان و گفتار

تجزیه و تحلیل سیگنال گفتار در پایتون

مدرس :

سید محمد حسین معطر

نگارش :

شیما شهرآئینی

پاییز 1402

4 Framing

8 Windowing

10 pre-emphasis

12 Energy

14 Zero Crossing Rate

16 Autocorrelation Coefficient

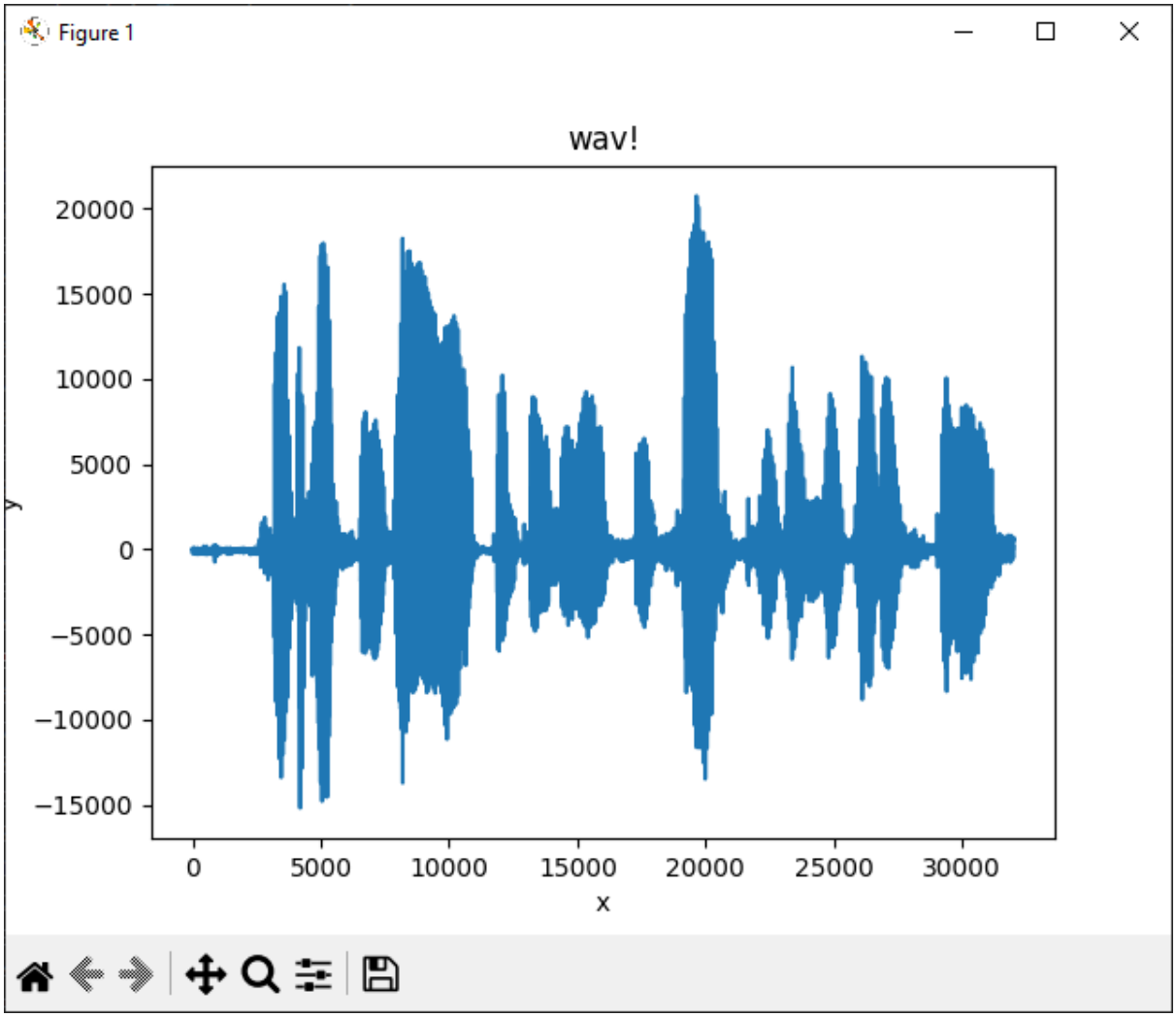
18 Average Magnitude Difference

20 Center Clipping

22 3level Center Clipping

24 Cepstral Coefficients

تجزیه و تحلیل سیگنال گفتار یک کار اساسی در زمینه پردازش سیگنال دیجیتال است. این شامل استخراج اطلاعات معنی دار از سیگنال های گفتاری، مانند انرژی، نرخ عبور صفر، ضرایب اتوکرلیشن، و ضرایب کپسترال است. در اینجا، تکنیک‌های مختلفی را برای تجزیه و تحلیل سیگنال‌های گفتاری با استفاده از پایتون بررسی می‌کنیم.



نمودار سیگنال خام

Framing

در پردازش سیگنال صوتی، فریمبندی به فرآیند تقسیم سیگنال صوتی پیوسته به یک سری فریم های همپوشانی اشاره دارد. هر فریم نشان دهنده بخش کوتاهی از سیگنال صوتی است و معمولاً برای تجزیه و تحلیل یا پردازش بیشتر استفاده می شود. فریم‌بندی یک مرحله ضروری در بسیاری از کارهای پردازش صدا، مانند تشخیص گفتار، تجزیه و تحلیل موسیقی و کدگذاری صدا است.

```python
def framing(sig, fs=16000, win_len=0.025, win_hop=0.01):
    """
    transform a signal into a series of overlapping frames.

    Args:
        sig            (array) : a mono audio signal (Nx1) from which to compute features.
        fs               (int) : the sampling frequency of the signal we are working with.
                                 Default is 16000.
        win_len        (float) : window length in sec.
                                 Default is 0.025.
        win_hop        (float) : step between successive windows in sec.
                                 Default is 0.01.

    Returns:
        array of frames.
        frame length.
        number of frames.
    """
    # compute frame length and frame step (convert from seconds to samples)
    frame_length = win_len * fs
    frame_step = win_hop * fs
    #signal_length = get_duration_pydub(file_path)
    signal_length = len(sig)
    frames_overlap = frame_length - frame_step

    # Make sure that we have at least 1 frame+
    num_frames = np.abs(signal_length - frames_overlap) // np.abs(frame_length - frames_overlap)
    rest_samples = np.abs(signal_length - frames_overlap) % np.abs(frame_length - frames_overlap)

    # Pad Signal to make sure that all frames have equal number of samples
    # without truncating any samples from the original signal
    if rest_samples != 0:
        pad_signal_length = int(frame_step - rest_samples)
        z = np.zeros((pad_signal_length))
        pad_signal = np.append(sig, z)
        num_frames += 1
    else:
        pad_signal = sig

    # make sure to use integers as indices
    frame_length = int(frame_length)
    frame_step = int(frame_step)
    num_frames = int(num_frames)

    # compute indices
    idx1 = np.tile(np.arange(0, frame_length), (num_frames, 1))
    idx2 = np.tile(np.arange(0, num_frames * frame_step, frame_step),(frame_length, 1)).T
    indices = idx1 + idx2

    frames = pad_signal[indices.astype(np.int32, copy=False)]
    #frames = indices.export(sig, format='wav')
    return frames, frame_length, num_frames


file_path = r'C:\Users\acer\Downloads\Music\h_orig.wav'
sample_rate , samples = wavfile.read(file_path)
plot_audio(np.arange(len(samples)), samples)

#show signal that has been framed
print('-> signal that has been framed:')
wav_frame, frame_length, num_frames = framing(samples, fs=sample_rate)
plot_audio(np.arange(len(wav_frame[185])), wav_frame[185])
plot_audio(np.arange(len(wav_frame)), wav_frame)
```

شرح کد:

تابع آرگومان های زیر را می گیرد:

sig: سیگنال صوتی مونو (Nx1) که از آن فریم ها محاسبه می شود.

fs: فرکانس نمونه برداری سیگنال. فرکانس نمونه برداری تعداد نمونه های گرفته شده در هر ثانیه برای نمایش سیگنال صوتی آنالوگ به صورت دیجیتالی را نشان می دهد. معمولاً با هرتز (هرتز) اندازه گیری می شود. مقدار پیش فرض استفاده شده در کد 16000 هرتز است.

win_len: طول پنجره بر حسب ثانیه. طول پنجره مدت زمان هر فریم را بر حسب ثانیه تعیین می کند. این یک پارامتر بسیار مهم است که بر وضوح زمانی سیگنال فریم شده تأثیر می گذارد. مقدار پیش فرض استفاده شده در کد 0.025 ثانیه (25 میلی ثانیه) است.

win_hop: مرحله بین پنجره های متوالی در چند ثانیه. پرش پنجره اندازه گام بین فریم های متوالی را در چند ثانیه تعیین می کند. همپوشانی بین فریم ها را کنترل می کند و بر وضوح فرکانس سیگنال قاب شده تأثیر می گذارد. مقدار پیش فرض استفاده شده در کد 0.01 ثانیه (10 میلی ثانیه) است.

تابع سه مقدار را برمی گرداند:

frames: آرایه‌ای از فریم‌ها که بخش‌های همپوشانی سیگنال ورودی را نشان می‌دهند.

frame_length: طول هر فریم در نمونه.

num_frames: تعداد کل فریم های تولید شده.

کد با محاسبه طول فریم و پرش فریم با ضرب طول فریم و پرش فریم(بر اساس زمان) در فرکانس نمونه‌برداری آغاز می‌شود. (ضرب طول فریم و پرش فریم بر اساس نمونه‌برداری) این مقادیر برای تعیین همپوشانی بین فریم ها استفاده می شود.

در مرحله بعد، کد با محاسبه تعداد فریم ها بر اساس طول سیگنال و همپوشانی بین فریم ها، تضمین می کند که حداقل یک فریم وجود دارد. همچنین نمونه های باقی مانده را که در فریم های کامل قرار نمی گیرند محاسبه می کند.

برای اطمینان از اینکه همه فریم‌ها دارای تعداد مساوی نمونه هستند بدون اینکه هیچ نمونه‌ای از سیگنال اصلی کوتاه شود، اگر نمونه‌های باقی مانده باشد، کد سیگنال را با صفر پر می‌کند. تعداد این صفرها با کم کردن نمونه های باقی مانده از میزات پرش فریم تعیین می شود.

سپس کد طول فریم، پرش فریم و تعداد فریم ها را برای اهداف ایندکس‌سازی به اعداد صحیح تبدیل می کند.

با استفاده از شاخص های محاسبه شده، کد فریم ها را از سیگنال پر شده استخراج می کند.

- شاخص های محاسباتی:

np.tile یک تابع NumPy است که یک آرایه را در ابعاد مشخص تکرار می کند. در این کد برای ایجاد دو آرایه idx1 و idx2 استفاده می شود.

np.arange دنباله ای از اعداد را در یک محدوده مشخص تولید می کند. در اینجا برای ایجاد اندیس برای فریم ها استفاده می شود.

شاخص ها با افزودن عناصر idx1 و idx2 محاسبه می شود. این نشان دهنده موقعیت نمونه ها در سیگنال ورودی برای هر فریم است.

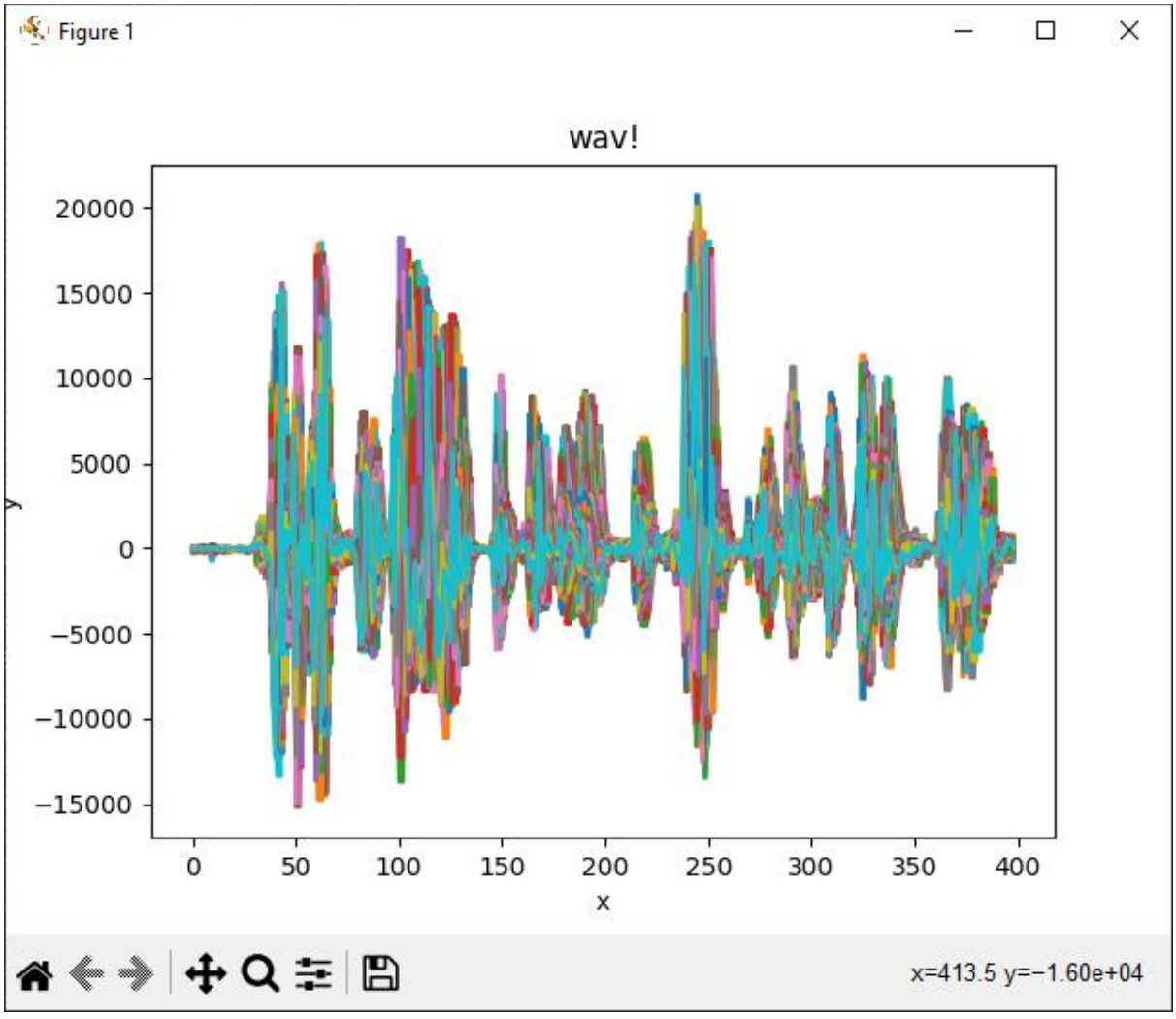
- استخراج فریم ها:

indices برای ایندکس کردن آرایه pad_signal استفاده می شود و فریم ها را از آن استخراج می کند.

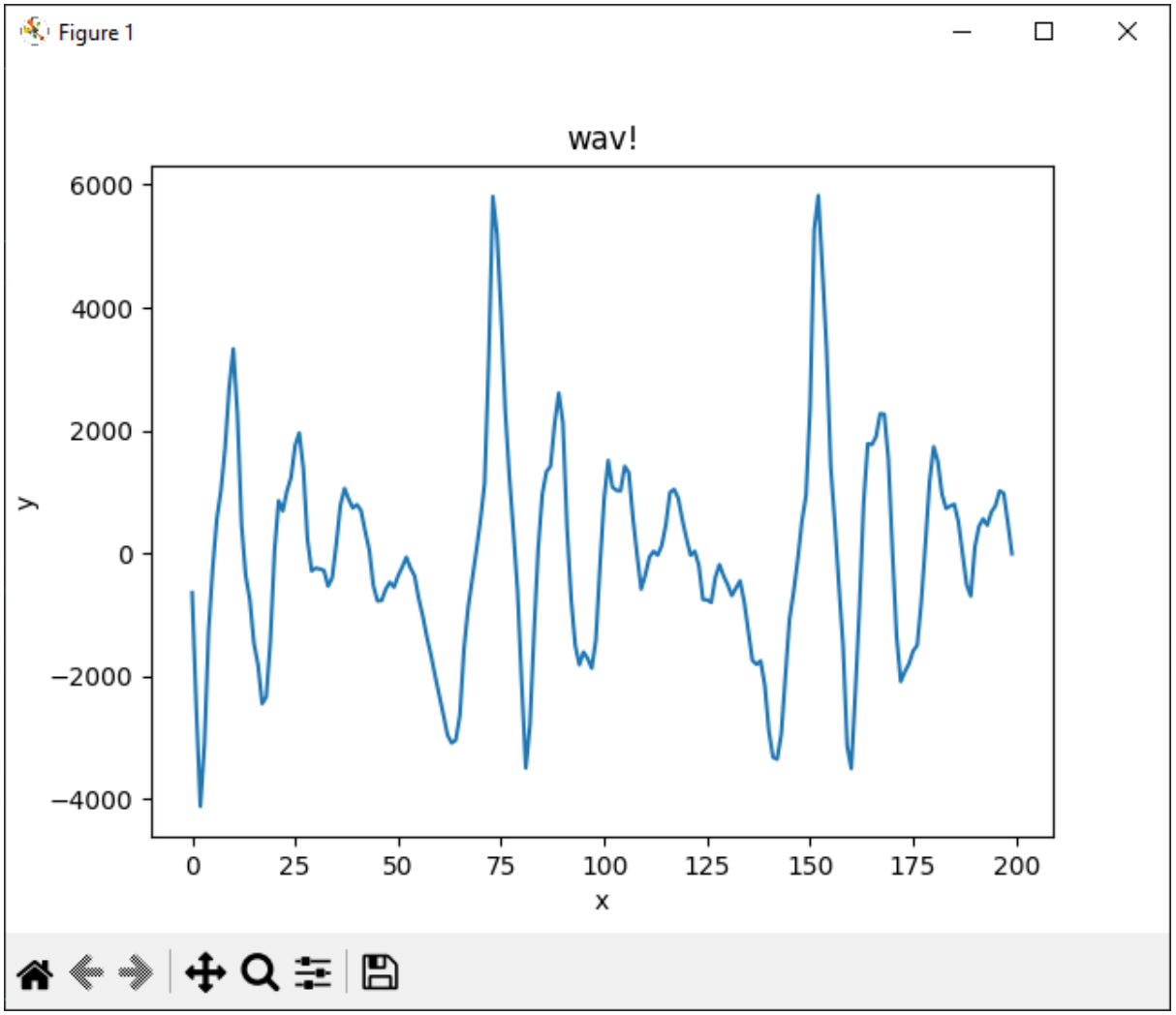
indices.astype(np.int32, copy=False)، شاخص ها را به اعداد صحیح تبدیل می کند و از نوع داده صحیح آنها اطمینان می دهد.

فریم های به دست آمده در متغیر فریم ها ذخیره می شوند.

در نهایت، تابع فریم ها، طول فریم و تعداد فریم ها را برمی گرداند.



سیگنال های فریم‌بندی شده



نمودار یک فریم تصادفی، فریم 185

Windowing

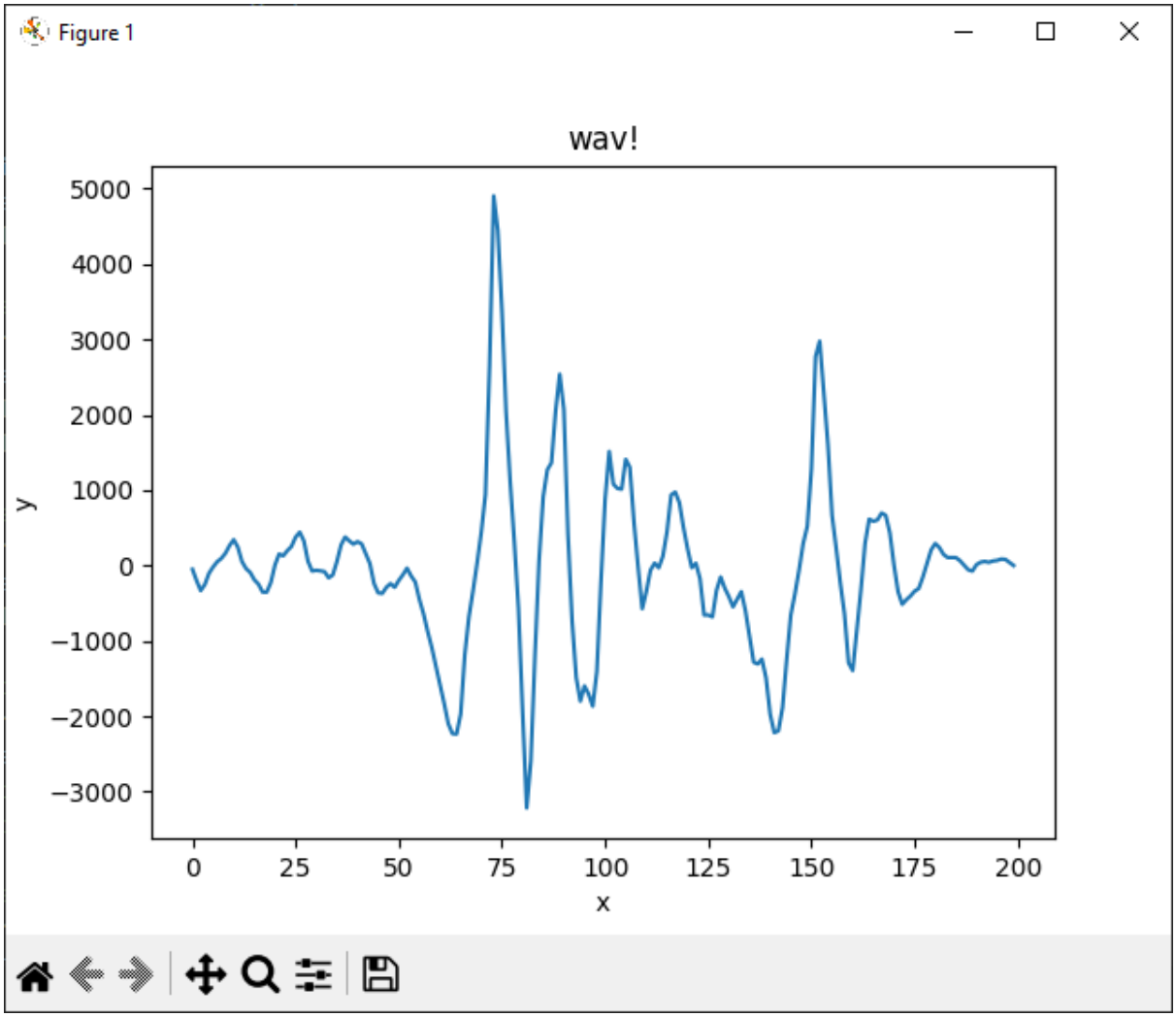
در پردازش سیگنال، پنجره سازی تکنیکی است که برای تجزیه و تحلیل و دستکاری سیگنال ها استفاده می شود. یکی از تابع های پنجره ای که معمولاً مورد استفاده قرار می گیرد، پنجره Hamming است برای کاهش نشت طیفی و بهبود دقت تجزیه و تحلیل، هر فریم در یک تابع پنجره، مانند پنجره Hamming ضرب می شود.

```
def haming_win(wav, frame_l):
    n = np.arange(0, frame_l)
    hw = 0.54 - (0.46 * np.cos(2*np.pi*n/frame_l))
    wav_windowed = wav * hw
    return wav_windowed

def windowing(frame_l, nframes, wav):
    windowed = []
    for i in range(nframes):
        w = haming_win(wav[i],frame_l)
        windowed.append(w)
    return windowed


#show signal that has been windowed(hamming)
print('-> signal that has been windowd by hamming function:')
wav_windowed = windowing(frame_length, num_frames, wav_frame)
plot_audio(np.arange(len(wav_windowed[185])), wav_windowed[185])
```

شرح کد:

- haming_win(wav, frame_l): این تابع پنجره Hamming را روی یک فریم سیگنال اعمال می کند. دو پارامتر نیاز دارد: wav که نشان دهنده سیگنال ورودی است و frame_l که طول فریم را مشخص می کند. تابع ضرایب پنجره همینگ را برای هر نمونه در قاب محاسبه می کند و آنها را با نمونه های مربوطه در سیگنال ورودی ضرب می کند. سیگنال پنجره ای به دست آمده برگردانده می شود.

- windowing(frame_l, nframes, wav): این تابع تابع haming_win را برای فریم های متعدد سیگنال ورودی اعمال می کند. این سه پارامتر نیاز دارد: frame_l که طول هر فریم را مشخص می کند، nframes که تعداد کل فریم ها را نشان می دهد و wav که سیگنال ورودی است. تابع روی هر فریم تکرار می‌شود، پنجره Hamming را با استفاده از تابع haming_win اعمال می‌کند و قاب پنجره‌دار را به یک لیست اضافه می‌کند. این لیست windowed برگردانده می شود.



نمودار یک فریم پنجره‌گذاری شده تصادفی، فریم 185

pre-emphasis

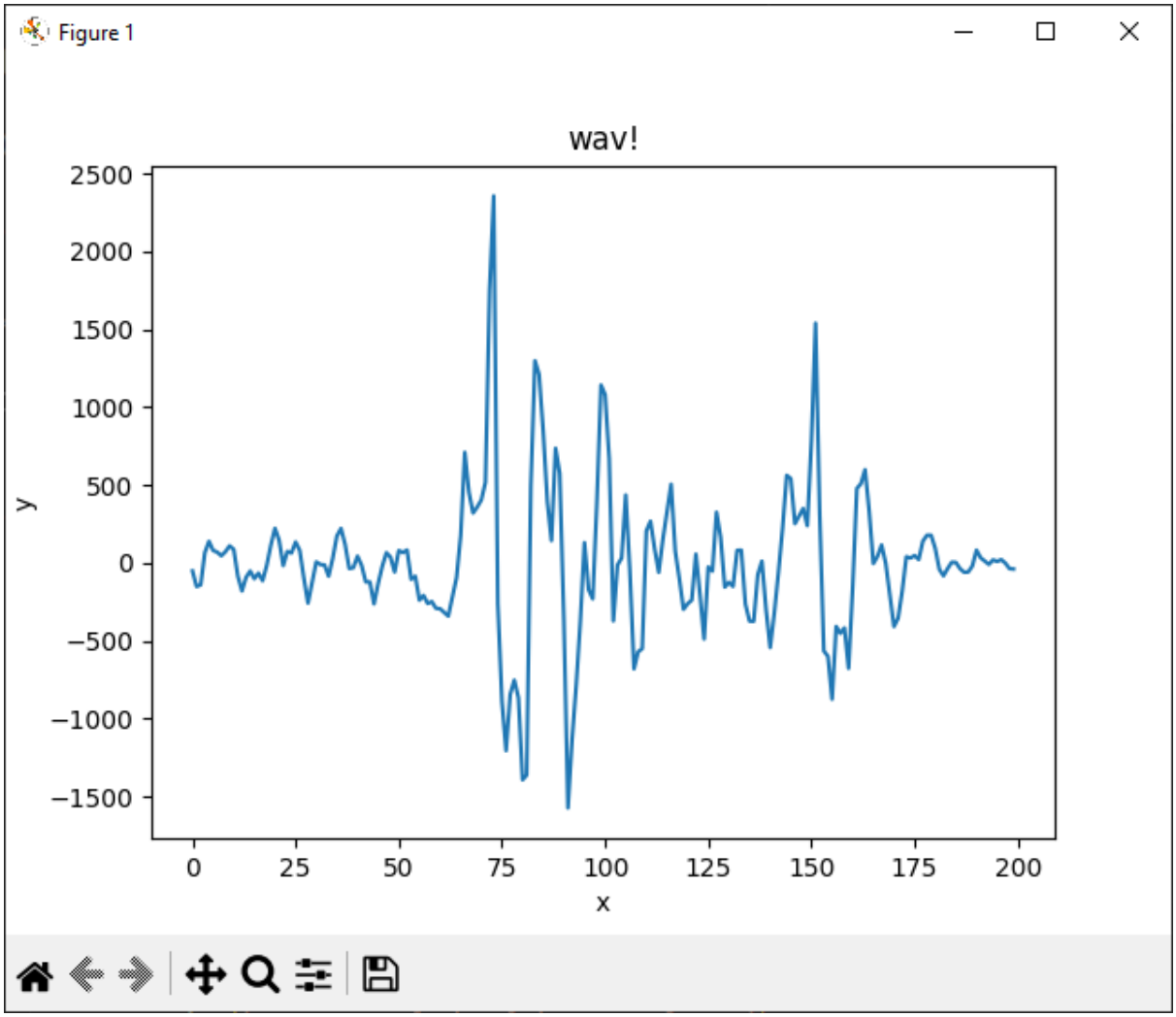
در پردازش صدا، پیش تاکید تکنیکی است که برای تأکید بر اجزای فرکانس بالا یک سیگنال قبل از ارسال یا پردازش بیشتر استفاده می‌شود. این تکنیک معمولاً در برنامه های گفتاری و صوتی برای بهبود کیفیت سیگنال و افزایش درک گفتار استفاده می شود.

```
def p_emphasis(signal, pre_emphasis):
    emphasized_signal = np.append(signal[0], signal[1:] - pre_emphasis * signal[:-1])
    return emphasized_signal

def pre_emphasis(nframes, wav, pre_emphasis=0.96):
    emphasis = []
    for i in range(nframes):
        e = p_emphasis(wav[i], pre_emphasis)
        emphasis.append(e)
    return emphasis


#pre_emphasise the signal
print('-> signal that has been pre_emphasis:')
wav_emphasis = pre_emphasis(num_frames, wav_windowed, pre_emphasis=0.96)
plot_audio(np.arange(len(wav_emphasis[185])), wav_emphasis[185])
```

شرح کد:

- p_emphasis(signal, pre_emphasis): این تابع بر روی یک فریم از سیگنال صوتی پیش تاکید را اعمال می کند. دو پارامتر به عنوان ورودی می گیرد: signal، که یک فریم از سیگنال صوتی را نشان می دهد، و pre_emphasis، که ضریب پیش تاکید است. تابع فرمول پیش تاکید را به سیگنال اعمال می کند و سیگنال تاکید شده را برمی گرداند.

- pre_emphasis(nframes, wav, pre_emphasis): این تابع بر روی فریم های متعدد سیگنال صوتی، پیش تاکید را اعمال می کند. سه پارامتر را به عنوان ورودی می گیرد: nframes که تعداد فریم های سیگنال صوتی را نشان می دهد، wav که سیگنال صوتی پنجره‌گذاری شده است و pre_emphasis که ضریب پیش تاکید است. این تابع روی هر فریم سیگنال صوتی تکرار می شود و تابع p_emphasis را برای هر فریم اعمال می کند. فهرستی از سیگنال های تاکید شده برای هر فریم را برمی گرداند.



نمودار یک فریم تصادفی بعد پیش‌تاکید، فریم 185

Energy

در پردازش سیگنال، تجزیه و تحلیل انرژی یک سیگنال می تواند بینش ارزشمندی در مورد ویژگی های آن ارائه دهد. انرژی اندازه گیری توان موجود در یک سیگنال است و می تواند برای شناسایی ویژگی ها یا الگوهای مهم استفاده شود.

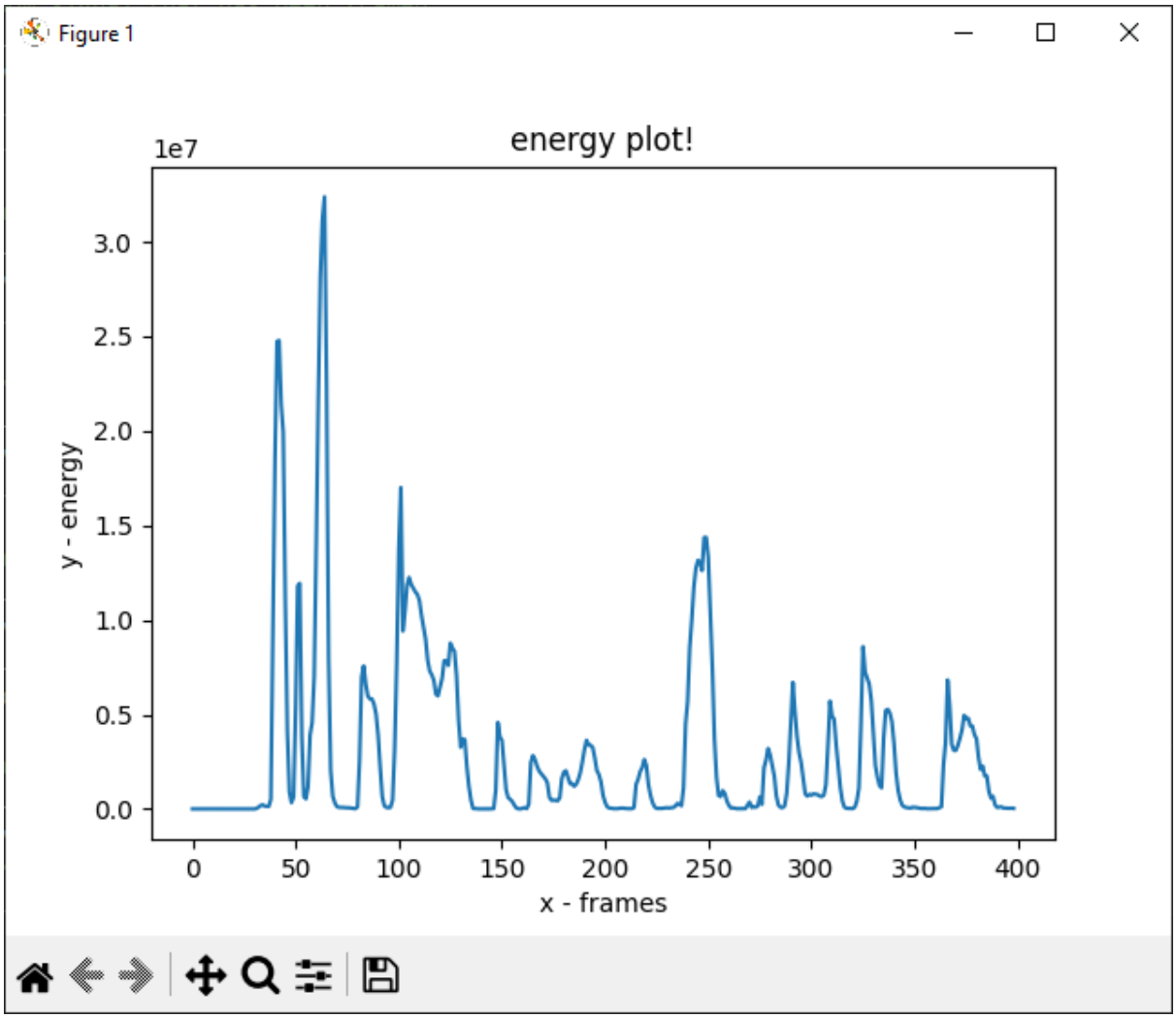
```
def energy(wav, frame_l):
    sum=0
    for i in range(frame_l):
        sum += pow(wav[i], 2)
    return sum/frame_l

def signal_energy(frame_l, nframes, wav):
    sig_energy = []
    for i in range(nframes):
        e = energy(wav[i],frame_l)
        sig_energy.append(e)
    return sig_energy

def plot_energy(x,y):
    plt.plot(x, y)
    plt.xlabel('x - frames')
    plt.ylabel('y - energy')
    plt.title('energy plot!')
    plt.show()


#show energy of every frame in signal
print('-> energy of signal in every frame:')
wav_energy = signal_energy(frame_length, num_frames, wav_windowed)
print(wav_energy,'\n')
plot_energy(np.arange(len(wav_energy)), wav_energy)
```

شرح کد:

- energy(wav, frame_l): این تابع انرژی یک فریم سیگنال را محاسبه می کند. دو پارامتر نیاز دارد: wav که نشان دهنده فریم سیگنال است و frame_l که طول فریم است. تابع بر روی نمونه های فریم تکرار می شود، هر نمونه را مربع می کند و آنها را جمع می کند. در نهایت انرژی متوسط قاب را برمی گرداند.

- signal_energy (frame_l, nframes, wav): این تابع انرژی هر فریم را در یک سیگنال محاسبه می کند. این به سه پارامتر نیاز دارد: frame_l که طول هر فریم است، nframes که تعداد کل فریم های سیگنال است و wav که کل سیگنال را نشان می دهد. تابع بر روی هر فریم تکرار می شود، تابع energy را برای محاسبه انرژی آن فراخوانی می کند و آن را به لیستی اضافه می کند. در نهایت، لیستی از انرژی های فریم را برمی گرداند.

- plot_energy(x,y): این تابع مقادیر انرژی هر فریم در سیگنال را ترسیم می کند. دو پارامتر نیاز دارد: x، که مقادیر محور x (شاخص‌های فریم) را نشان می‌دهد، و y، که مقادیر محور y (انرژی‌های فریم) را نشان می‌دهد. این تابع از کتابخانه matplotlib برای ایجاد نمودار خطی از مقادیر انرژی استفاده می کند.



نمودار انرژی سیگنال

Zero Crossing Rate

در پردازش سیگنال، نرخ عبور صفر (ZCR) اندازه گیری تعداد دفعاتی است که یک سیگنال علامت خود را در یک بازه زمانی معین تغییر می دهد. این یک معیار مفید برای تجزیه و تحلیل محتوای فرکانس و ویژگی های یک سیگنال است.

```
def sgn(x):
    if x>=0:
        return 1
    return -1

def ZCR(wav, frame_l):
    sum=0
    for i in range(1, frame_l):
        sum += (abs(sgn(wav[i]) - sgn(wav[i-1])) / 2)
    return sum

def signal_zcr(frame_l, nframes, wav):
    sig_zcr = []
    for i in range(nframes):
        z = ZCR(wav[i],frame_l)
        sig_zcr.append(z)
    return sig_zcr

def plot_zcr(x,y):
    plt.plot(x, y)
    plt.xlabel('x - frame')
    plt.ylabel('y - zcr')
    plt.title('zero crossing rate plot!')
    plt.show()


#show zero crossing rate of every frame in signal
print('-> zero crosing rate of signal in every frame:')
wav_zcr = signal_zcr(frame_length, num_frames, wav_windowed)
print(wav_zcr, '\n')
plot_zcr(np.arange(len(wav_zcr)), wav_zcr)
```

شرح کد:

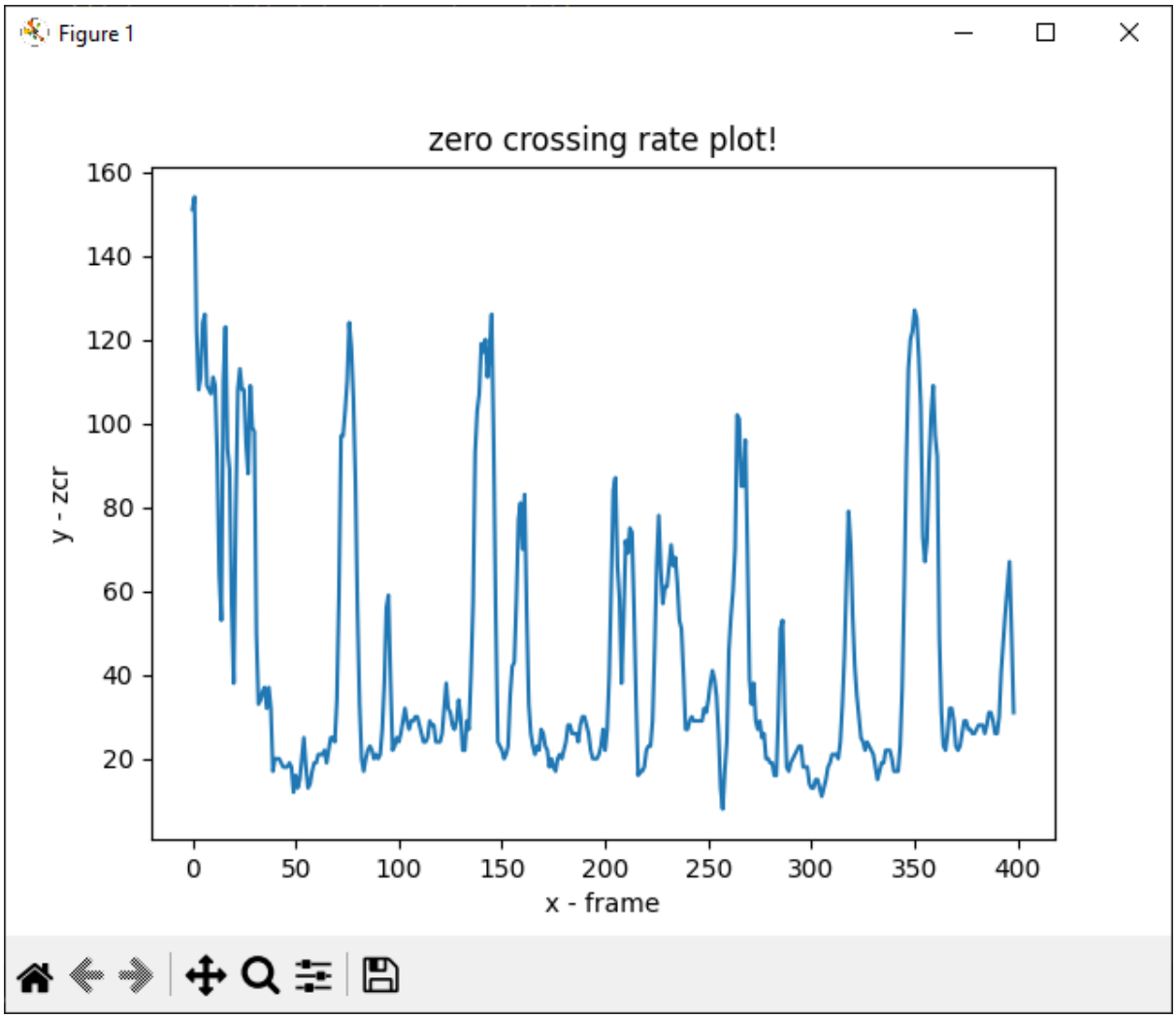
- sgn(x): این تابع یک عدد x را به عنوان ورودی می گیرد و اگر x بزرگتر یا مساوی صفر باشد، 1 و در غیر این صورت -1 را برمی گرداند. برای تعیین علامت هر نمونه در سیگنال استفاده می شود.

- ZCR(wav, frame_l): این تابع نرخ عبور صفر را برای یک فریم از سیگنال محاسبه می کند. دو پارامتر نیاز دارد: wav که نشان دهنده فریم سیگنال است و frame_l که طول فریم است. تابع بر روی نمونه های فریم تکرار می

شود و تعداد تغییرات علامت را می شمارد و آنها را در متغیر sum جمع می کند. در نهایت ZCR محاسبه شده را برمی گرداند.

- signal_zcr(frame_l ،nframes ،wav): این تابع ZCR را برای هر فریم در سیگنال محاسبه می کند. این به سه پارامتر نیاز دارد: frame_l که طول هر فریم را نشان می دهد، nframes که تعداد کل فریم های سیگنال را مشخص می کند و wav که کل شکل موج را نشان می دهد. تابع بر روی هر فریم تکرار می شود و تابع ZCR را برای محاسبه ZCR برای آن فریم فراخوانی می کند و نتیجه را به لیست sig_zcr اضافه می کند. در نهایت، لیستی از مقادیر ZCR را برای هر فریم برمی گرداند.

- plot_zcr(x,y): این تابع مقادیر ZCR را در برابر شماره فریم رسم می کند. این دو پارامتر نیاز دارد: x که نشان دهنده اعداد فریم است و y که مقادیر ZCR را نشان می دهد. این تابع از کتابخانه matplotlib برای ایجاد یک نمودار خطی استفاده می کند و محور x را به عنوان "frame" و محور y را به عنوان "zcr" برچسب گذاری می کند. همچنین عنوانی به داستان اضافه می کند. در نهایت، طرح را نمایش می دهد.



نمودار نرخ عبور صفر سیگنال

Autocorrelation Coefficient

در پردازش سیگنال، خودهمبستگی یک ابزار ریاضی است که برای اندازه‌گیری شباهت بین سیگنال و نسخه تاخیری خود استفاده می‌شود. معمولاً در کاربردهای مختلفی مانند تشخیص گفتار، پردازش صدا و تجزیه و تحلیل سری های زمانی استفاده می شود. ضریب خود همبستگی قدرت و جهت رابطه خطی بین سیگنال و نسخه تاخیری آن را کمیت می کند.

```python
def autocorrelation_coefficient(signal):
    n = len(signal)
    autocorr = []

    for lag in range(n):
        sum_product = 0
        for i in range(n - lag):
            sum_product += signal[i] * signal[i + lag]
        autocorr.append(sum_product/n)

    return autocorr

def plot_autocorrelation(x,y):
    plt.plot(x, y)
    plt.xlabel('x - lag')
    plt.ylabel('y - autocorrelation Coefficient')
    plt.title('autocorrelation Coefficient plot for 1 frame!')
    plt.show()


#show autocorrelation of 1 random frame in signal
print('-> autocorrelation of signal in 1 frame:')
frame_autocorrelation = autocorrelation_coefficient(list(wav_emphasis[185]))
print(frame_autocorrelation, '\n')
plot_autocorrelation(np.arange(len(frame_autocorrelation)), frame_autocorrelation)
```

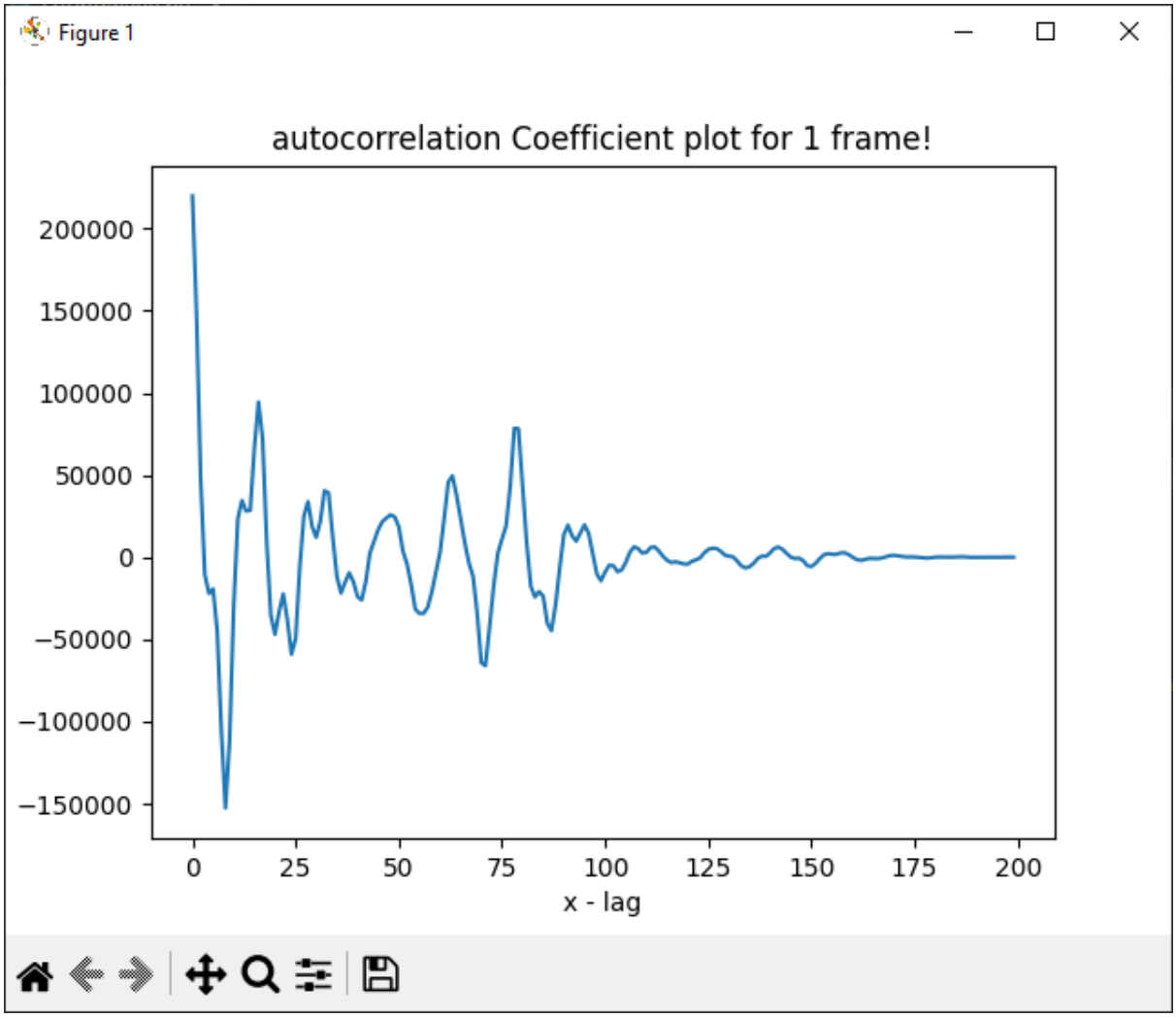
شرح کد:

- autocorrelation_coefficient(signal): این تابع ضریب همبستگی خود را برای یک سیگنال مشخص محاسبه می کند. سیگنال را به عنوان ورودی می گیرد و لیستی از مقادیر ضریب همبستگی خود را برمی گرداند.

در اینجا نحوه عملکرد کد آمده است:

1. برای ذخیره مقادیر ضریب همبستگی خودکار یک لیست خالی autocorr را راه اندازی می کند.
2. با تاخیرهای زمانی مختلف از 0 تا طول سیگنال تکرار می شود.
3. برای هر تأخیر زمانی، مجموع حاصل ضرب مقادیر سیگنال در اندیس i و i + lag را محاسبه می کند.
4. مجموع را بر طول سیگنال تقسیم می کند و نتیجه را به لیست خودکار اضافه می کند.
5. در نهایت، فهرست autocorr حاوی مقادیر ضریب همبستگی خودکار را برمی‌گرداند.

- plot_autocorrelation(x, y): این تابع مقادیر ضرایب خودهمبستگی را رسم می کند. دو آرگومان نیاز دارد: x (وقفه زمانی) و y (مقادیر ضریب همبستگی خودکار). با استفاده از تابع plt.plot مقادیر ضرایب خودهمبستگی را رسم می کند.



نمودار اتوکورلیشن یک فریم تصادفی، فریم 185

Average Magnitude Difference

AMDF یک ابزار مفید در پردازش سیگنال برای تجزیه و تحلیل تناوب و تخمین گام در سیگنال های صوتی است. با محاسبه میانگین اختلاف بین نمونه‌ها در وقفه‌های زمانی مختلف، می‌توانیم بینشی در مورد تناوب یک سیگنال به دست آوریم.

```
def AMDF(signal):
    n = len(signal)
    amdf = []
    for lag in range(n):
        sum_product = 0
        for i in range(n - lag):
            sum_product += abs(signal[i] - signal[i + lag])
        amdf.append(sum_product/n)
    return amdf

#show avarage magnitude differene function of 1 random frame in signal
print('-> varage magnitude differene function of signal in 1 frame:')
frame_amdf = AMDF(list(wav_emphasis[185]))
print(frame_amdf, '\n')
plot_audio(np.arange(len(frame_amdf)), frame_amdf)
```

شرح کد:

تابع AMDF یک سیگنال را به عنوان ورودی می گیرد و مقادیر AMDF را برای تاخیرهای مختلف محاسبه می کند. در اینجا یک تفکیک گام به گام کد آمده است:

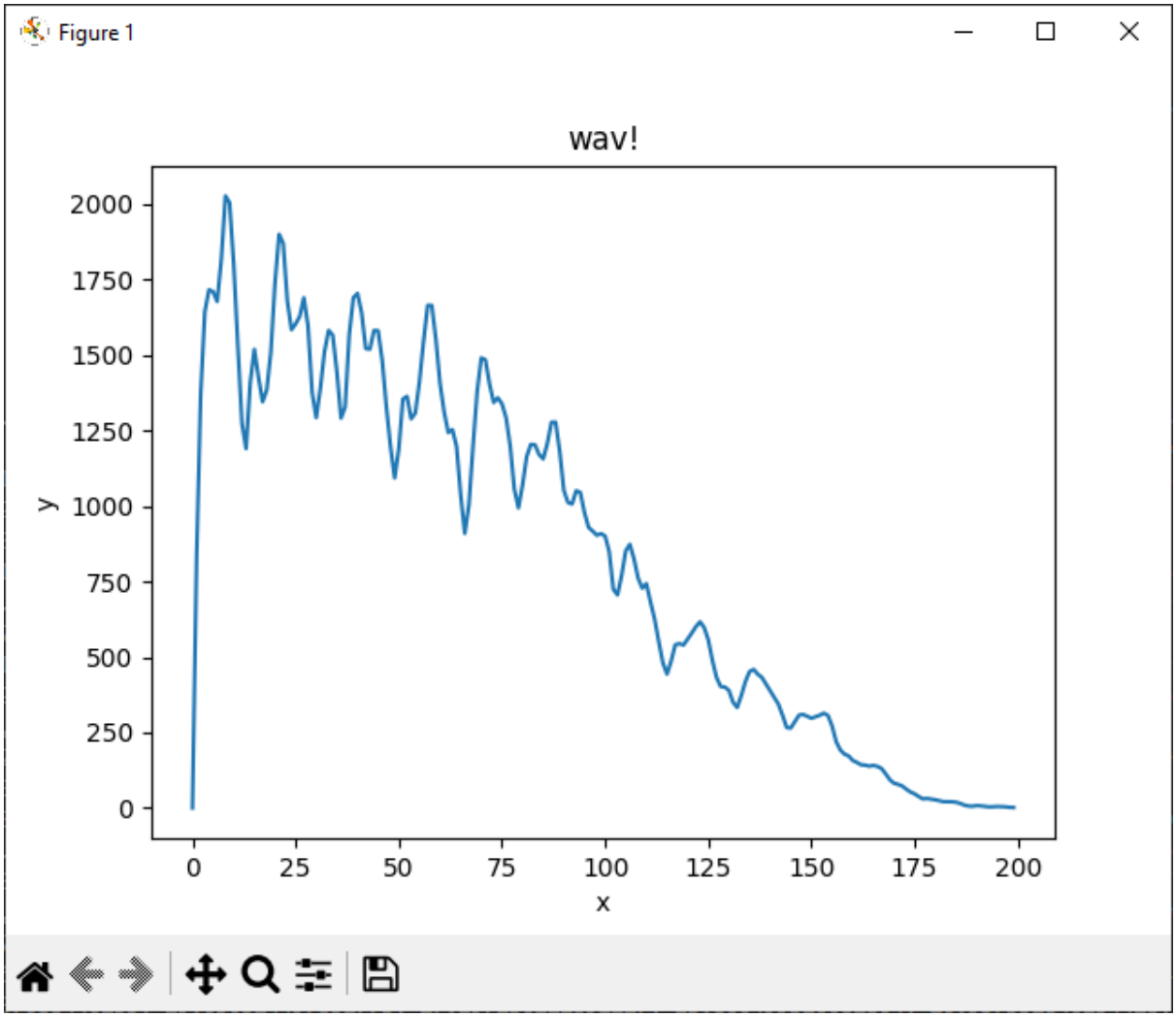
این تابع یک لیست خالی به نام amdf را برای ذخیره مقادیر AMDF مقداردهی می کند.

با استفاده از متغیر تاخیر در محدوده طول سیگنال تکرار می شود. این تاخیر، تاخیر اعمال شده به سیگنال را تعیین می کند.

در داخل حلقه بیرونی، یک حلقه داخلی وجود دارد که مجموع تفاوت های مطلق بین مقادیر سیگنال و نسخه های تاخیری آنها را محاسبه می کند.

مجموع محاسبه شده بر طول سیگنال n تقسیم می شود و به لیست amdf اضافه می شود.

در نهایت، تابع لیست amdf حاوی مقادیر AMDF را برمی گرداند.



نمودار AMDF یک فریم تصادفی، فریم 185

Center Clipping

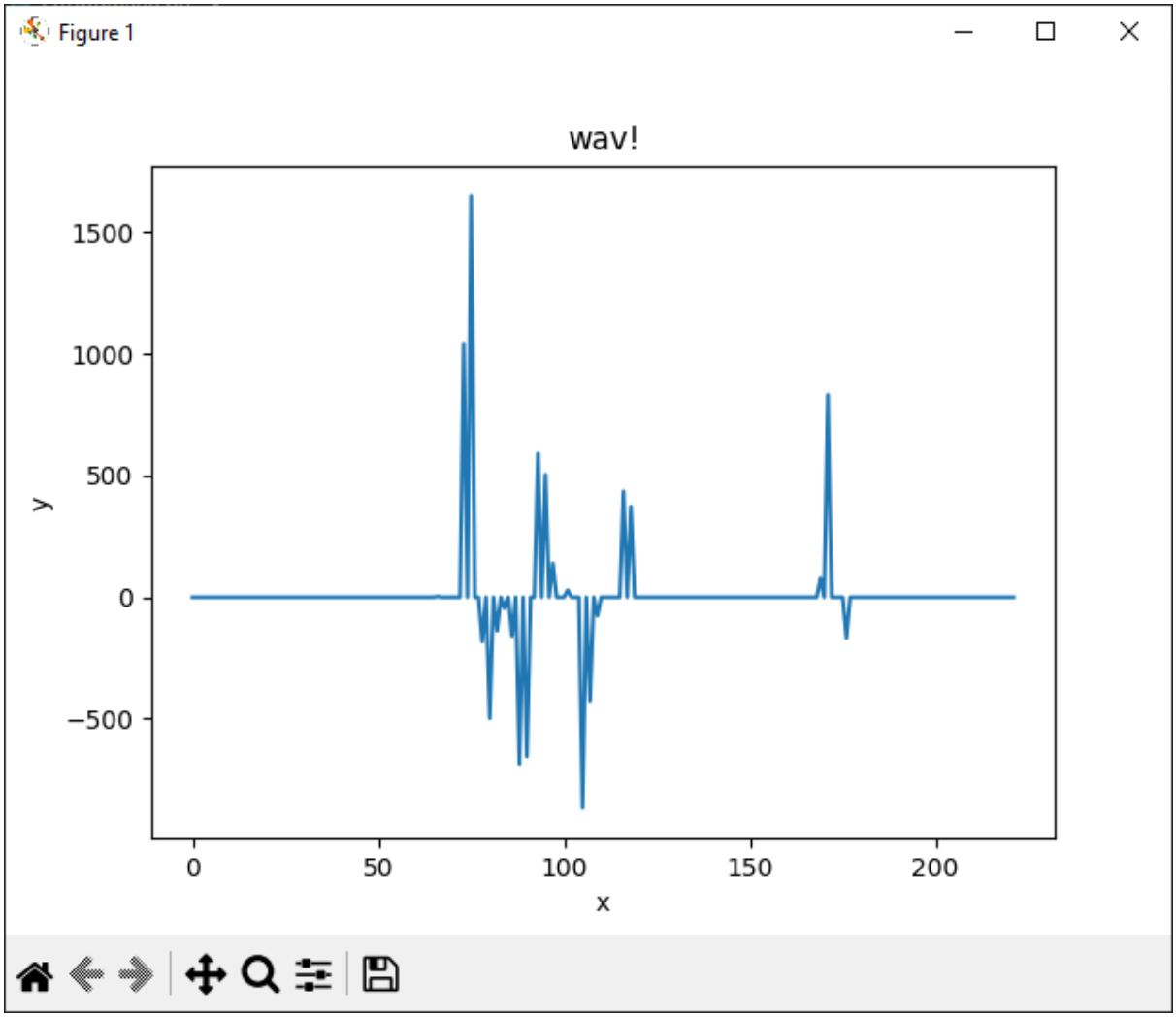
برش مرکزی تکنیکی است که در پردازش سیگنال برای حذف یا کاهش دامنه سیگنال هایی که از یک آستانه خاص فراتر می روند استفاده می شود. معمولاً در پردازش صدا برای جلوگیری از اعوجاج ناشی از سیگنال های بیش از حد بلند استفاده می شود.

```
def center_clip(S, cut):
    max_value = np.amax(S)
    c = cut * max_value

    C = []
    for value in S:
        if value > c:
            C.append(value - c)
        elif value <-c:
            C.append(value + c)
        C.append(0)
    return C

#center clipping of 1 random frame in signal
print('-> signal after center clipping:')
cc_frame = center_clip(wav_emphasis[185], 0.3)
print(cc_frame, '\n')
plot_audio(np.arange(len(cc_frame)), cc_frame)
```

شرح کد:

تابع center_clip دو پارامتر دارد: S (سیگنال ورودی) و cut (ضریب برش). تابع حداکثر مقدار در سیگنال ورودی را با استفاده از np.amax از کتابخانه NumPy محاسبه می کند. سپس ضریب برش را با حداکثر مقدار ضرب می کند تا آستانه (c) را تعیین کند.

سپس، تابع به اندازه مقادیر در سیگنال ورودی تکرار می شود. اگر مقدار از آستانه (c) بیشتر شود، آستانه را از مقدار کم کرده و به لیست خروجی C اضافه می کند. اگر مقدار کمتر از آستانه منفی (-c) باشد، آستانه را به مقدار اضافه می کند. اگر مقدار در محدوده آستانه باشد، 0 را به لیست خروجی اضافه می کند.



نمودار یک فریم تصادفی بعد از center-clipping ، فریم 185

3level Center Clipping

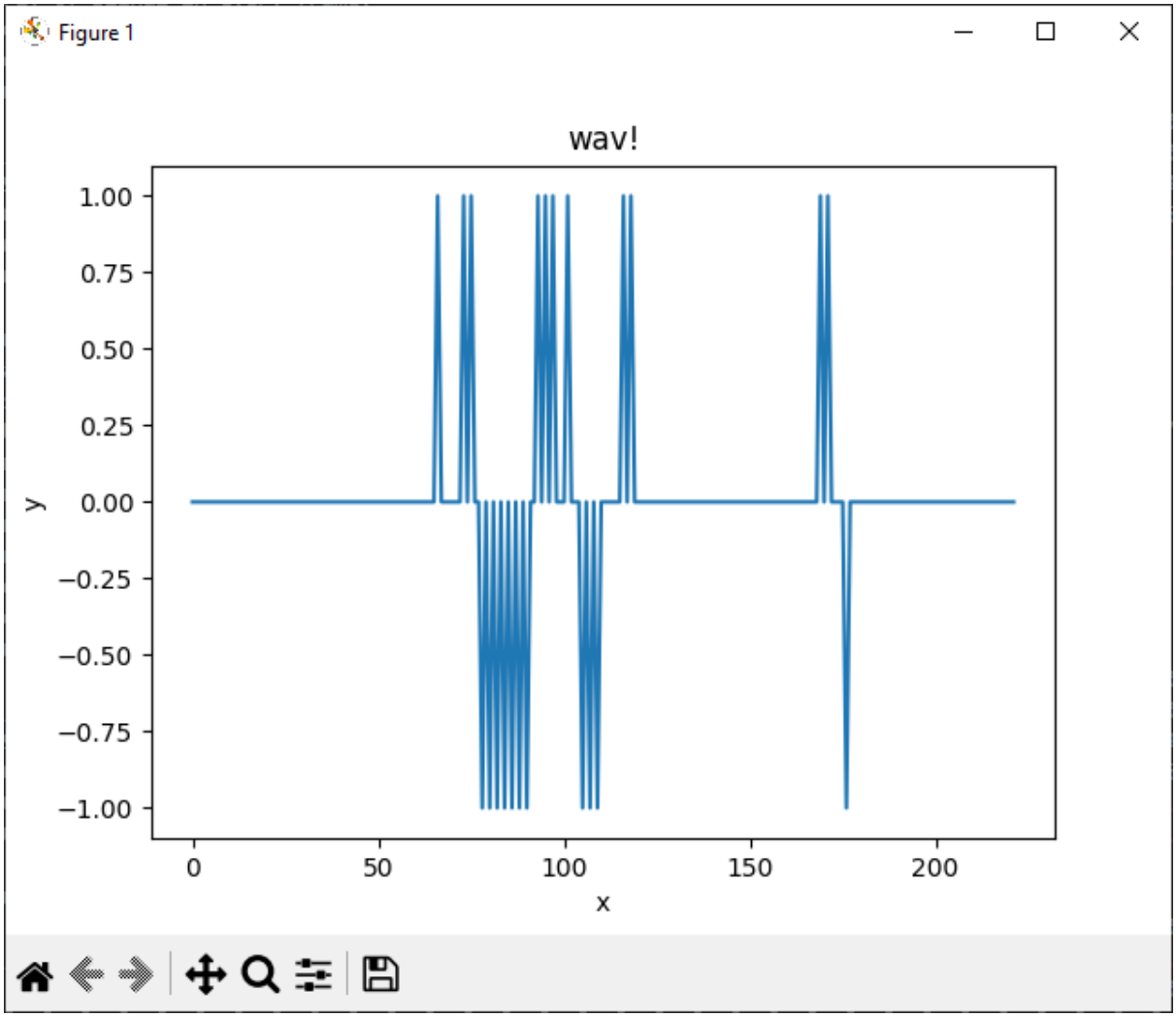
عملیات برش مرکز سه سطحی در این کد، مقادیر سیگنال را در سه سطح 1، -1 و 0 دسته بندی می کند.

```
def threelevel_center_clip(S, cut):
    max_value = np.amax(S)
    c = cut * max_value

    C = []
    for value in S:
        if value > c:
            C.append(1)
        elif value <-c:
            C.append(-1)
        C.append(0)
    return C

#3 level center clipping of 1 random frame in signal
print('-> signal after 3 level center clipping:')
three_cc_frame = threelevel_center_clip(wav_emphasis[185], 0.3)
print(three_cc_frame, '\n')
plot_audio(np.arange(len(three_cc_frame)), three_cc_frame)
```

شرح کد:

تابع threelevel_center_clip دو پارامتر دارد: S (سیگنال ورودی) و cut (آستانه برای برش مرکزی). عملیات برش مرکزی سه سطحی را روی سیگنال ورودی انجام می دهد و سیگنال قطع شده را برمی گرداند.

سپس، تابع به اندازه مقادیر در سیگنال ورودی تکرار می شود. اگر مقدار از آستانه (c) بیشتر شود، آستانه مقدار 1 به لیست خروجی C اضافه می کند. اگر مقدار کمتر از آستانه منفی (-c) باشد، مقدار 1- اضافه می کند. اگر مقدار در محدوده آستانه باشد، 0 را به لیست خروجی اضافه می کند.



نمودار یک فریم تصادفی بعد از 3level-center-clipping، فریم 185

Cepstral Coefficients

آنالیز Cepstral تکنیکی است که در پردازش سیگنال برای تجزیه و تحلیل محتوای طیفی سیگنال استفاده می شود. این شامل تبدیل سیگنال از حوزه فرکانس به حوزه کپسترال است که اطلاعاتی در مورد ویژگی های طیف سیگنال ارائه می دهد.

```
def cepstral_analysis(signal):
    # FFT
    spectrum = []
    for frame in signal:
        spectrum.append(np.abs(fft(frame)))

    # Logarithm
    log_spectrum = np.log(np.array(spectrum))

    # IFFT
    cepstrum = []
    for frame in log_spectrum:
        cepstrum.append(np.real(ifft(frame)))

    return cepstrum


#show Cepstral Coefficients of 1 random frame in signal
print('-> Cepstral Coefficients  of signal in 1 frame:')
cepstral = cepstral_analysis(wav_emphasis)
print(cepstral_analysis, '\n')
plot_audio(np.arange(len(cepstral[185])), cepstral[185])
```

شرح کد:

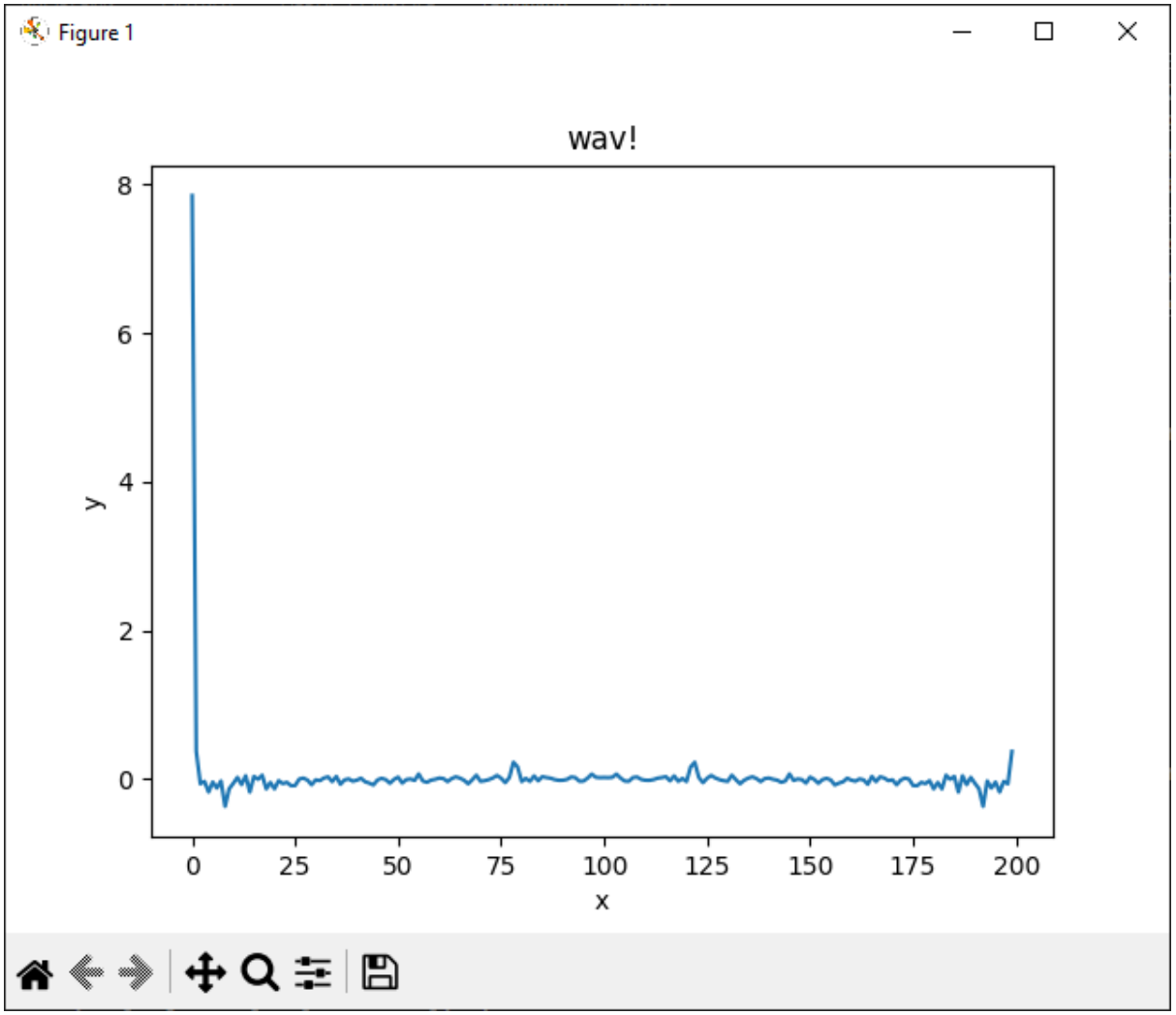
تابع cepstral_analysis یک سیگنال را به عنوان ورودی دریافت می کند و مراحل زیر را انجام می دهد:

FFT: تبدیل فوریه سریع هر فریم در سیگنال را با استفاده از تابع fft از کتابخانه NumPy محاسبه می کند. مقادیر مطلق طیف حاصل در لیست طیف ذخیره می شود.

لگاریتم: لگاریتم طیف با استفاده از تابع np.log محاسبه می شود. این مرحله محدوده دینامیکی سیگنال را فشرده می کند.

IFFT: تبدیل فوریه معکوس به هر فریم از طیف لگاریتمی با استفاده از تابع ifft اعمال می شود. بخش واقعی مقادیر پیچیده حاصل در لیست cepsrum ذخیره می شود.

در نهایت، تابع cepstral_analysis کسپستروم را برمی گرداند که نشان دهنده ضرایب کپسترال سیگنال است.



نمودار ضرایب کپسترال یک فریم تصادفی، فریم 185