پروژه پردازش فایل صوتی به زبان پایتون

استاد دکتر حسن پور

| 9524603 | سید محمد نصیر ستوده

شرح پروژه :

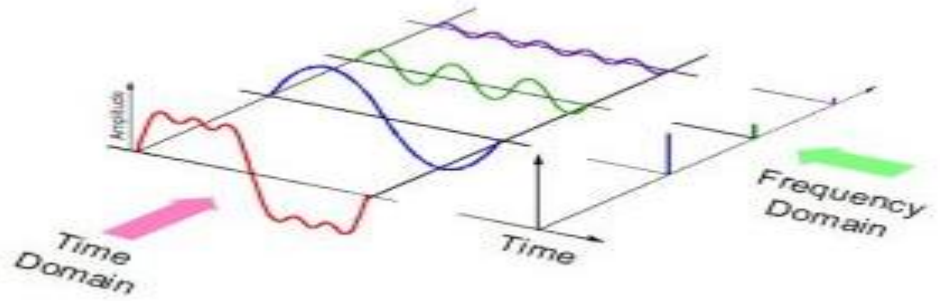
- قصد انجام این پروژه ضبط صدا و رسم نمودار دامنه فرکانسی آن
- و سپس اعمال اکو بر روی صدا
- و انتقال دامنه فرکانسی
- و سپس جمع سیگنال اکو با سیگنال انتقال داده شده می باشد

در این پروژه از زبان پایتون استفاده شده که دارای چندین کتاب خانه در حوضه رسم نمودار و گرفتن پارامتر ها می باشد .

این کتاب خانه ها متد ها و توابع زیادی در اختیار ما گذاشته تا بتوان سیگنال ورودی را به هر فرمت دلخواه خوانده و به هر فرمت دلخواه تبدیل کنیم .

گام نخست این است که داده‌ها را به طور عملی به فرمت قابل فهم برای ماشین بارگذاری کنیم. برای این، ما به سادگی مقادیر را بعد از هر گام زمانی خاص دریافت می‌کنیم؛ برای مثال در یک فایل صوتی 2 ثانیه‌ای، مقادیر را در نیم‌ثانیه استخراج می‌کنیم. این موضوع، **نمونه‌برداری از داده‌های صوتی (sampling of audio data)** نامیده می‌شود و نرخ آن، **نرخ نمونه‌برداری (sampling rate)** نامیده می‌شود.

روش دیگری برای نمایش داده‌های صوتی، تبدیل آن به یک نمایش متفاوت از دامنه داده‌ها، یعنی **دامنه فرکانس (frequency domain)** است. هنگامی که ما یک داده صوتی را به عنوان نمونه در نظر می‌گیریم، نیازمند نقاط داده‌ای بسیار زیادی برای نشان دادن کل داده‌ها هستیم و نرخ نمونه‌برداری نیز باید تا حد ممکن زیاد باشد. از سوی دیگر، اگر ما اطلاعات صوتی را به صورت دامنه فرکانس نمایش دهیم.



```
obj = wave.open('sound.wav','wb')
```

حالت 'rb' یک شی **Wave_read** را برمی گرداند ، در حالی که حالت **'wb'** یک شی **Wave_write** را برمی گرداند.

این عملکرد یک پرونده را برای خواندن / نوشتن داده های صوتی باز می کند.

این تابع به دو پارامتر نیاز دارد - ابتدا نام پرونده و دوم حالت. حالت می تواند برای نوشتن داده های صوتی "wb"
یا "rb"برای خواندن باشد.

close()	Close the file if it was opened by wave.
setnchannels()	Set the number of channels. 1 for Mono 2 for stereo channels
setsampwidth()	Set the sample width to n bytes.
setframerate()	Set the frame rate to n.
setnframes()	Set the number of frames to n.
setcomptype()	Set the compression type and description. At the moment, only compression type NONE is supported, meaning no compression.
setparams()	accepts parameter tuple (nchannels, sampwidth, framerate, nframes, comptype, compname)
tell()	Retrieves current position in the file
writeframesraw()	Write audio frames, without correcting.
writeframes()	Write audio frames and make sure they are correct.

Wave_read object methods:

close()	Close the stream if it was opened by wave module.
getnchannels()	Returns number of audio channels (1 for mono, 2 for stereo).
getsampwidth()	Returns sample width in bytes.
getframerate()	Returns sampling frequency.
getnframes()	Returns number of audio frames.
getcomptype()	Returns compression type ('NONE' is the only supported type).
getparams()	Returns a namedtuple() (nchannels, sampwidth, framerate, nframes, comptype, compname), equivalent to output of the get*() methods.
readframes(n)	Reads and returns at most n frames of audio, as a bytes object.
rewind()	Rewind the file pointer to the beginning of the audio stream.

گزارش کار :

ابتدا نیاز به ضبط چند فایل صوتی داریم که فایل ها صوتی ما به فرمت wav می باشد .

لازم به ذکر است هر فایل wav به دو صورت mono و stereo می توان ذخیره کرد .

تفاوت این دو در تعداد کانال های فرکانسی می باشد که صدای مونو، تک صوتی شناخته شده. در آن، تمام سیگنال‌های مختلف صوتی با هم آمیخته و به یک کانال صوتی منفرد تبدیل می‌شوند و از این رو، بر آن نام مونو (منفرد یا تک) گذاشته شد.

سیگنالهای صوتی در استریو به دو یا تعداد بیشتری کانال تقسیم می‌شوند که هر یک از آنها به یک اسپیکر متفاوتی فرستاده می‌شوند. این کار کمک می‌کند تا درک جهت و عمق صدا بهتر شبیه سازی شود.

setnchannels()	Set the number of channels. 1 for Mono 2 for stereo channels

با استفاده از این متد می توان تعداد کانال های فرکانسی را تشخیص داد .

ابتدا برای تشخیص و سپس رسم فایل صوتی کلاسی نوشتیم تحت عنوان showfile.py

این کلاس داری چهار تابع می باشد .

ما ابتدا باید فایل مورد نظر را گرفته و سپس تعداد کانال های آن را برسی کنیم .

تابعی نوشتیم تحت عنوان :()getfile

```
def getfile(p):
    # p = 'monow/1.wav'

    obj = wave.open(p, 'r')
    print("Number of channels", obj.getnchannels())
    print("Sample width", obj.getsampwidth())
    print("Frame rate.", obj.getframerate())
    print("Number of frames", obj.getnframes())
    print("parameters:", obj.getparams())
    if (obj.getnchannels() == 1):
        obj = wavfile.read(p, 'r')
        monoshow(obj)
    else:
        stereoshow(obj)
        obj.close()
```

این تابع علاوه بر برسی تعداد کانال ها مقادیر sample rate و دامنه و فریم Frame Per Second و دامنه و پارامتر ها را نمایش می دهد و اگر سیگنال منو باشد با تابع monoshow و استریو باشد با تابع stereoshow نمایش می دهد .

تابع **monoshow(obj)**

برای رسم سیگنال تک صوتی از این تابع استفاده می کنیم .

بایستی قبل از فراخوانی با استفاده از آدرس و پارامتر خواندن سیگنال مورد نظر را خوانده و سپس به تابع فراخوانی کنیم.

obj = wavfile.read(p, 'r')
monoshow(obj)

```
def monoshow(f):
    # Load the data and calculate the time of each sample
    samplerate, data = f
    times = np.arange(len(data)) / float(samplerate)
    plt.figure(figsize=(15, 10))
    plt.fill_between(times, data)
    plt.xlim(times[1000], times[-1])
    plt.xlabel('time (s) mono')
    plt.ylabel('amplitude')
    # You can set the format by changing the extension
    #  like .pdf, .svg, .eps
    plt.savefig('plot.png', dpi=1000)
    plt.show()
```

ابتدا برای تشکیل محور زمان timeرا محاسبه کرده و سپس فریم پلات مد نظر را ساخته و مقادیر آرایه مربوط به دامنه و برد را تشکیل داده و پلات مورد نظر را با فرمت عکس چاپ و ذخیره می کنیم .

به همین ترتیب یک داده استریو را هم با تابع :stereoshow()

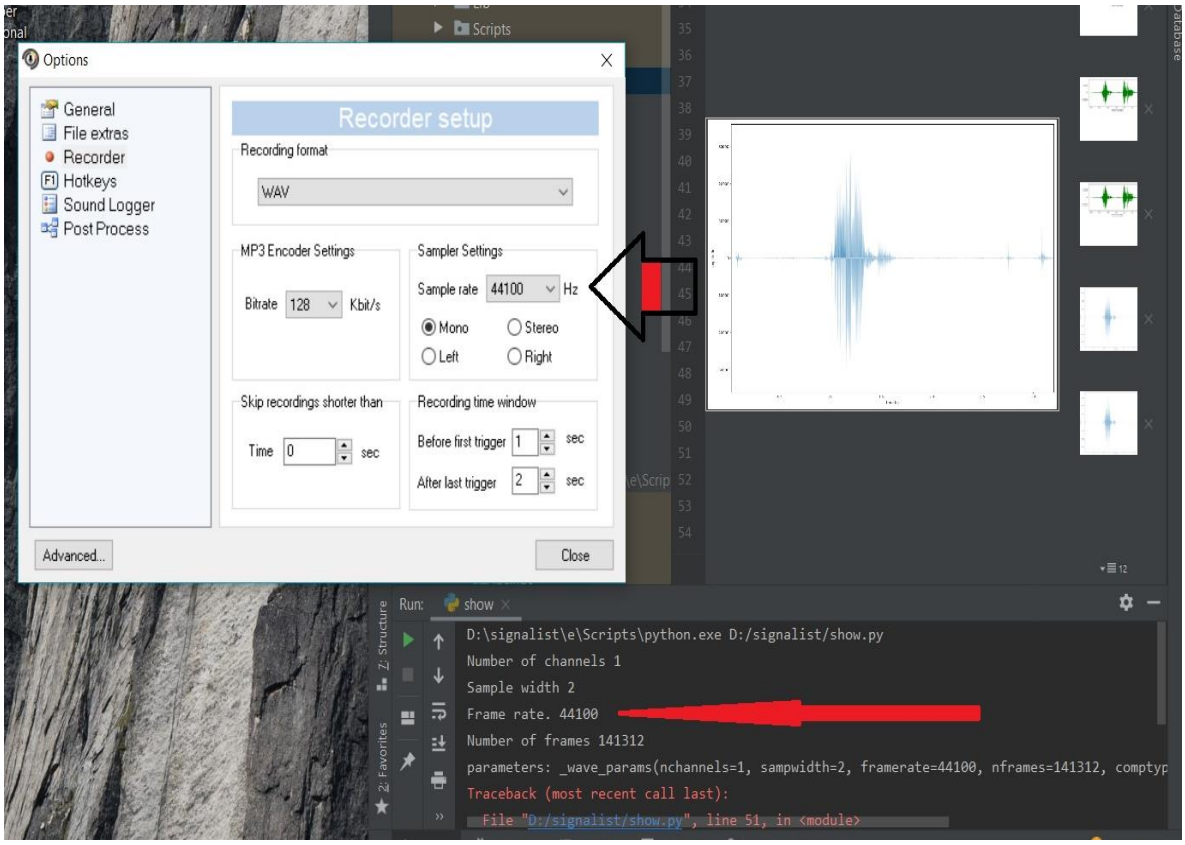
می توان رسم کرد منتها در استریو با متد

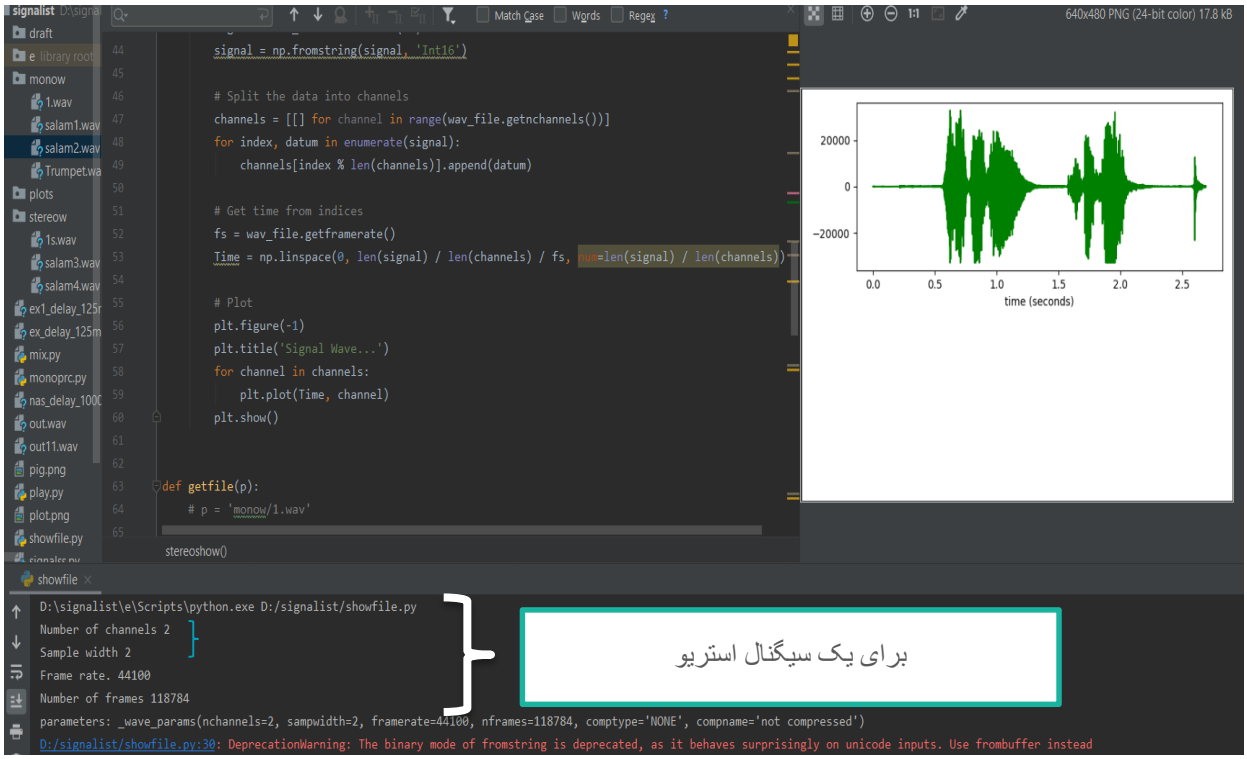
```
str_data = f.readframes(nframes)
wave_data = np.fromstring(str_data, dtype=np.short)
```

فریم مورد نظر را پیدا کرده و مقادیر را در قالب متن string در یک آرایه ذخیره می کنیم .

در اینجا مهم پیدا کردن دامنه زمانی فرکانس است که از فرمول زیر بدست آوردیم :

```
time = np.arange(0, nframes) * (1.0 / framerate)
```


Options
General
File extras
Recorder
Hotkeys
Sound Logger
Post Process
Recorder setup
Recording format
WAV
MP3 Encoder Settings
Bitrate 128 Kbit/s
Sampler Settings
Sample rate 44100 Hz
Mono
Stereo
Left
Right
Skip recordings shorter than
Time 0 sec
Recording time window
Before first trigger 1 sec
After last trigger 2 sec
Advanced...
Close
Run: show
D:\signalist\e\Scripts\python.exe D:/signalist/show.py
Number of channels 1
Sample width 2
Frame rate. 44100
Number of frames 141312
parameters: _wave_params(nchannels=1, sampwidth=2, framerate=44100, nframes=141312, comptyp
Traceback (most recent call last):
File "D:/signalist/show.py", line 51, in <module>


signalist
draft
library root
monow
1.wav
salam1.wav
salam2.wav
Trumpet.wa
plots
stereow
1s.wav
salam3.wav
salam4.wav
ex1_delay_125r
ex_delay_125m
mix.py
monoprc.py
nas_delay_100
out.wav
out11.wav
pig.png
play.py
plot.png
showfile.py
signalsr.py
signal = np.fromstring(signal, 'Int16')
# Split the data into channels
channels = [[] for channel in range(wav_file.getnchannels())]
for index, datum in enumerate(signal):
channels[index % len(channels)].append(datum)
# Get time from indices
fs = wav_file.getframerate()
Time = np.linspace(0, len(signal) / len(channels) / fs, num=len(signal) / len(channels))
# Plot
plt.figure(-1)
plt.title('Signal Wave...')
for channel in channels:
plt.plot(Time, channel)
plt.show()
def getfile(p):
# p = 'monow/1.wav'
stereoshow()
640x480 PNG (24-bit color) 17.8 kB
20000
0
−20000
0.0
0.5
1.0
1.5
2.0
2.5
time (seconds)
showfile
D:\signalist\e\Scripts\python.exe D:/signalist/showfile.py
Number of channels 2
Sample width 2
Frame rate. 44100
Number of frames 118784
parameters: _wave_params(nchannels=2, sampwidth=2, framerate=44100, nframes=118784, comptype='NONE', compname='not compressed')
D:/signalist/showfile.py:30: DeprecationWarning: The binary mode of fromstring is deprecated, as it behaves surprisingly on unicode inputs. Use frombuffer instead
برای یک سیگنال استریو

برای اکو دادن به فایل نیاز داریم تمامی پارامتر ها و فریم مورد نظر را گرفته تا تک تک بتوان در آن تغییراتی ایجاد کرد .

هدف ما فیلتر کردن سیگنال است همچنین ایجاد یک سفت در سیگنال و سپس رسم آن .

از آنجایی که ما دو نوع صدا داریم (تک صوتی و چند صوتی)

نیازمند یک تابع برای تبدیل چند صوتی به تک صوتی هستیم.

```
def convert_to_mono(channels, nChannels, outputType):
    if nChannels == 2:
        samples = np.mean(np.array([channels[0], channels[1]]), axis=0)  # Convert to
mono
    else:
        samples = channels[0]

    return samples.astype(outputType)
```

و نیازمند یک تابع برای گرفتن تمامی پارامتر ها همچنین ابتدای دامنه برای اعمال شیفت هستیم.

کلاس filter.py

در این کلاس چند تابع داریم

```
def extract_audio(fname, tStart=None, tEnd=None):
```

اسم و آدرس فایل را گرفته همچنین می توان مقدار اولیه سیگنال و مقدار پایانی آن را ارسال کرد.

```
def extract_audio(fname, tStart=None, tEnd=None):
    with contextlib.closing(wave.open(fname,'rb')) as spf:
        sampleRate = spf.getframerate()
        ampWidth = spf.getsampwidth()
        nChannels = spf.getnchannels()
        nFrames = spf.getnframes()

        startFrame, endFrame, segFrames = get_start_end_frames(nFrames, sampleRate,
tStart, tEnd)

        # Extract Raw Audio from multi-channel Wav File
        spf.setpos(startFrame)
        sig = spf.readframes(segFrames)
        spf.close()

        channels = interpret_wav(sig, segFrames, nChannels, ampWidth, True)

        return (channels, nChannels, sampleRate, ampWidth, nFrames)
```

تابع مورد نظر تمامی پارامتر هارا مانند فرکانس طول موج و تعداد کانال ها و فریم هارا گرفته و بر می گرداند .

لازم است در اینجا مقدار اولیه و مقدار پایانی سیگنال را محاسبه کنیم . برای این کار تابع زیر را داریم

```
def get_start_end_frames(nFrames, sampleRate, tStart=None, tEnd=None):
```

حال ما تمامی مقادیر مورد نیاز برای اعمال هر گونه تغییر در سیگنال داریم .

```
channels, nChannels, sampleRate, ampWidth, nFrames =
extract_audio('monow/1.wav',tStart, tEnd)

samples = convert_to_mono(channels, nChannels, np.int16)
```

فایل مورد نظر را خوانده و تمامی پارامتر ها را می گیرم.

به یک فایل تک صوتی تبدیل می کنیم.

و داده آن را در مقدار سمپل زخیره می کنیم .

هدف از فیلتر کردن این است مثلا اگر حوضه فرکانس ما بین 44 تا 1200 هرتز است آن را به مقدار دلخواه خودمان ببریم.

تابع فیلتر زیر را داریم :

```
def fir_band_pass(samples, fs, fL, fH, NL, NH, outputType):

    fH = fH / fs
    fL = fL / fs


یک فیلتر کم گذر را با فرکانس قطع محاسبه کنیم
    hlpf = np.sinc(2 * fH * (np.arange(NH) - (NH - 1) / 2.))
    hlpf *= np.blackman(NH)
    hlpf /= np.sum(hlpf)
    high-pass filter
    hhpf = np.sinc(2 * fL * (np.arange(NL) - (NL - 1) / 2.))
    hhpf *= np.blackman(NL)
    hhpf /= np.sum(hhpf)
    hhpf = -hhpf
    hhpf[int((NL - 1) / 2)] += 1
    # Convolve both filters.
    h = np.convolve(hlpf, hhpf)
    # Applying the filter to a signal s can be as simple as writing
    s = np.convolve(samples, h).astype(outputType)
```

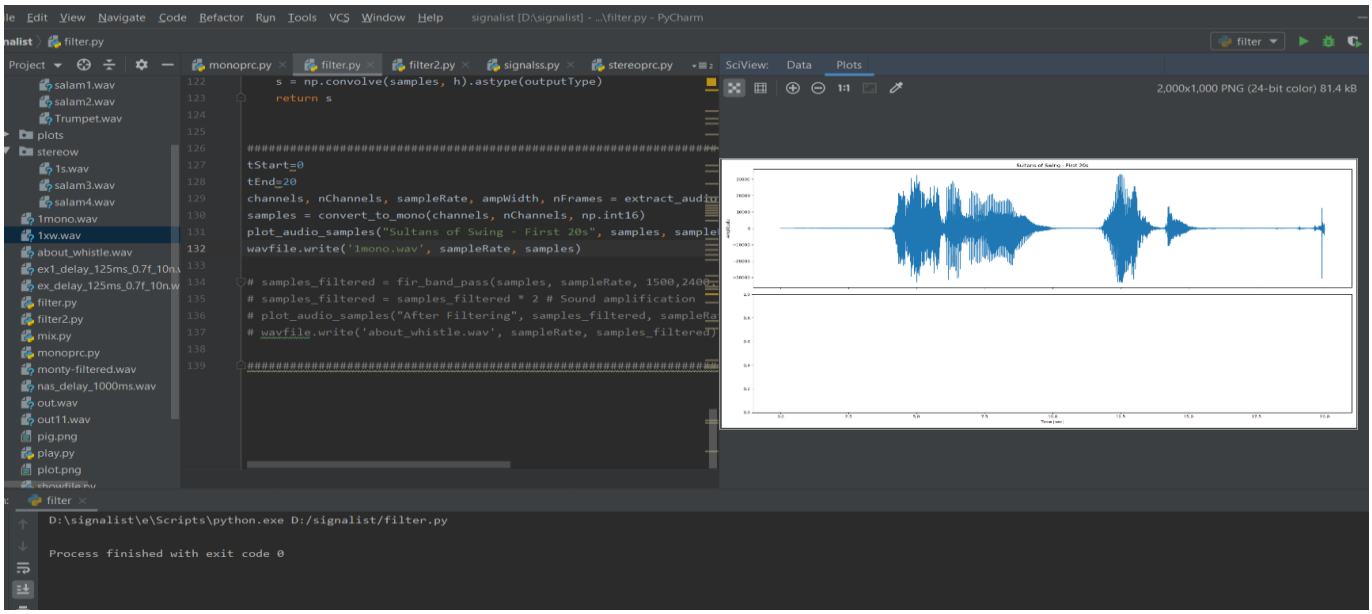
ما سیگنال پایین گذر و بالا گذر را محاسبه کرده و با هم جمع می کنیم.

تست :

```
tStart=0
tEnd=20
channels, nChannels, sampleRate, ampWidth, nFrames = extract_audio('monow/salam1.wav',
tStart, tEnd)
samples = convert_to_mono(channels, nChannels, np.int16)
plot_audio_samples("firstTest", samples, sampleRate, tStart, tEnd)
wavfile.write('1mono.wav', sampleRate, samples)
```

برای تست ابتدا زمان شروع و پایان را مشخص و سپس با تابع اکسرکت نوشته شده فایل مورد نظر را گرفته سپس پارامتر ها را زخیره می کینم. و سیگنال را به یک کانال تک صوتی تبدیل و مقدار آن را ذخیره می کنیم.

تابعی داریم برای رسم سیگنال از آن استفاده می کینم تا سیگنال اولیه خومان را ببینیم.سپس بدون هیچ تغییری فایل مورد نظر را به صورت مونو زخیره می کنیم.



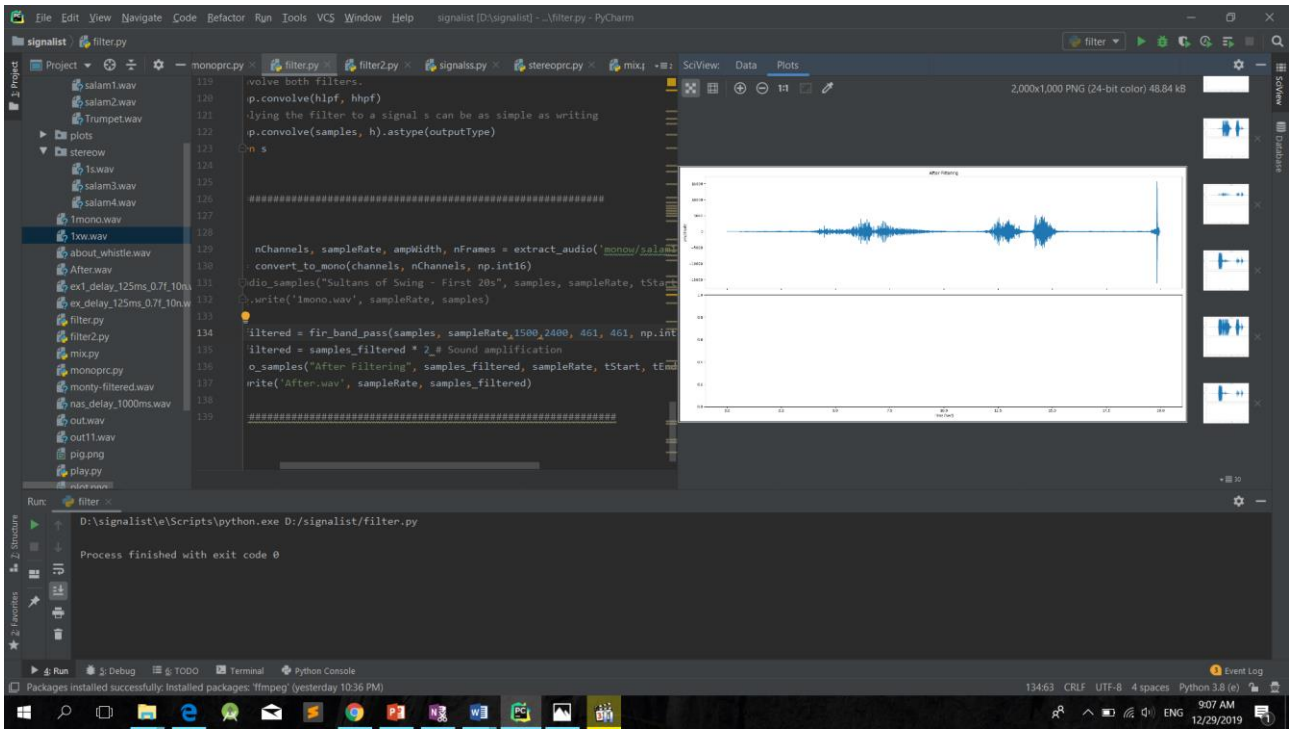
این سیگنال تک صوتی ماست .

حال آن را با تابع فیلتر نوشته شده و با توجه به مقدار دلخواه تغییر می دهیم و آن را ذخیره و رسم می کنیم.

```
samples_filtered = fir_band_pass(samples, sampleRate, 1500,2400, 461, 461, np.int16)
samples_filtered = samples_filtered * 2 # Sound amplification
plot_audio_samples("After Filtering", samples_filtered, sampleRate, tStart, tEnd)
wavfile.write('about_whistle.wav', sampleRate, samples_filtered)
```

در اینجا ما فرکانس را به بین 1500 تا 2400 انتقال دادیم .

ما پلات جدید فایل مورد نظر را با فیلتر اعمال در فرکانس بین 1500 تا 2400 می بینیم:



نتیجه گیری این است که با توجه به تست های انجام شده یعنی تغییر فرکانس بین بازه های مختلف مثلا 0 تا 6600 یا 1500 تا 6600 این است که هر چقدر بازه فرکانس را کمتر می کنیم صدا حالت ضعیف تر پیدا می کند و این طور مشاهده می شود که صدا های اضافی حذف می شود اما هر چقدر این بازه بیشتر می شود قدرت صدا بیشتر شده و صدا های اضافه بیشتری شنیده می شود .

برای اکو دادن به فایل نیاز داریم تمامی پارامتر ها و فریم مورد نظر را گرفته تا تک تک بتوان در آن تغییراتی ایجاد کرد

برای این کار کلاسی تحت عنوان monoprc نوشتیم که تنها برای فایل های تک صوتی کاربرد دارد .

با تابع input_wave(filename,f2) آدرس و بیشترین سقف فریم مورد نظر را میگیریم.

سپس پرامتر ها و فریم سیگنال را جدا و بر می گردنیم .

پارامتر ها شامل فرکانس و زمان و ... می باشد .

```
def input_wave(filename, frames=10000000):
    with wave.open(filename, 'rb') as wave_file:
        params = wave_file.getparams()
        audio = wave_file.readframes(frames)
        if params.nchannels != 1:
            raise Exception("The input audio should be mono for these examples")
    return params, audio
```

چنانچه تغییری در سیگنال اعمال کنیم نیازمند آن هستیم که سیگنال جدید را با پسوند و نام مورد نظر و پارامتر های جدید زخیره کنیم .

```
wave.open(filename, 'wb')
```

کتاب خانه wave پایتون در متد openعلاوه بر پارامتر "rb" که برای خواندن readمورد استفاده قرار می گیرد با پارامتر "wb " می توان نوشتن را نیز اعمال کرد تا مقادیر جدید را در سیگنال بنویسیم .

```
def output_wave(audio, params, stem, suffix):
    filename = stem.replace('.wav', '_{}.wav'.format(suffix))
    with wave.open(filename, 'wb') as wave_file:
        wave_file.setparams(params)
        wave_file.writeframes(audio)
```

با این تابع خروجی جدید را درست می کنیم.

برای اکو دادن به سیگنال لازم است یک شیفت بر حسب میلی ثانیه به آن اعمال شود .

که آن را از کاربر می گیریم.

تابع زیر را داریم :

که offset_ms در واقع مقدار تغییر در فرکانس بر حسب میلی ثانیه را گرفته و در تابع delay در تمامی سیگنال اعمال می شود .

مقادیر جمع سیگنال چپ و راست را با توجه به تغییر مورد نظر به ما می دهد.

```
delay(audio_bytes, params, offset_ms, factor=1, num=1):
#delays after 'offset_ms' milliseconds amplified by 'factor'
    if factor >= 1:
        warn("These settings may produce a very loud audio file. \
              Please use caution when listening")
    # calculate the number of bytes which corresponds to the offset in milliseconds
    offset = params.sampwidth * offset_ms * int(params.framerate / 1000)
    # add extra space at the end for the delays
    audio_bytes = audio_bytes + b'\0' * offset * (num)
    # create a copy of the original to apply the delays
    delayed_bytes = audio_bytes
    for i in range(num):
        # create some silence
        beginning = b'\0' * offset * (i + 1)
        # remove space from the end
        end = audio_bytes[:-offset * (i + 1)]
        # multiply by the factor
        multiplied_end = mul(end, params.sampwidth, factor ** (i + 1))
        delayed_bytes = add(delayed_bytes, beginning + multiplied_end,
params.sampwidth)
    return delayed_bytes
```

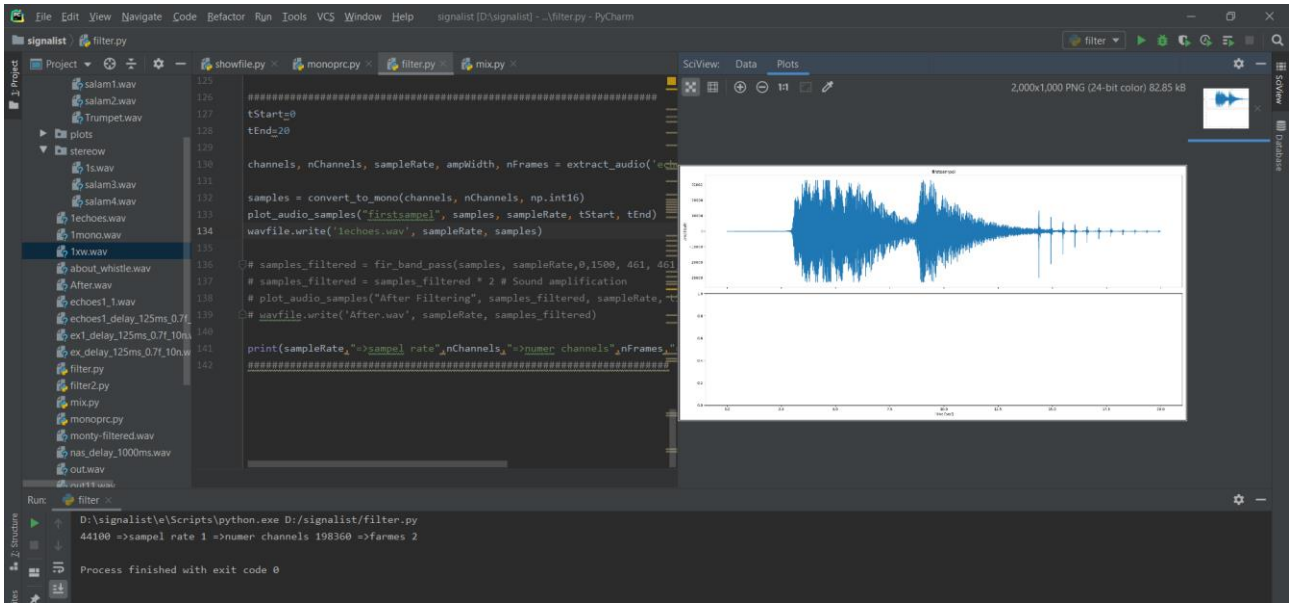
حال باید خروجی مورد نظر را بسازیم .

```
def delay_to_file(audio_bytes, params, offset_ms, file_stem, factor=1, num=1):
    echoed_bytes = delay(audio_bytes, params, offset_ms, factor, num)
    output_wave(echoed_bytes, params, file_stem,
                1'.format(offset_ms, factor, num))
```

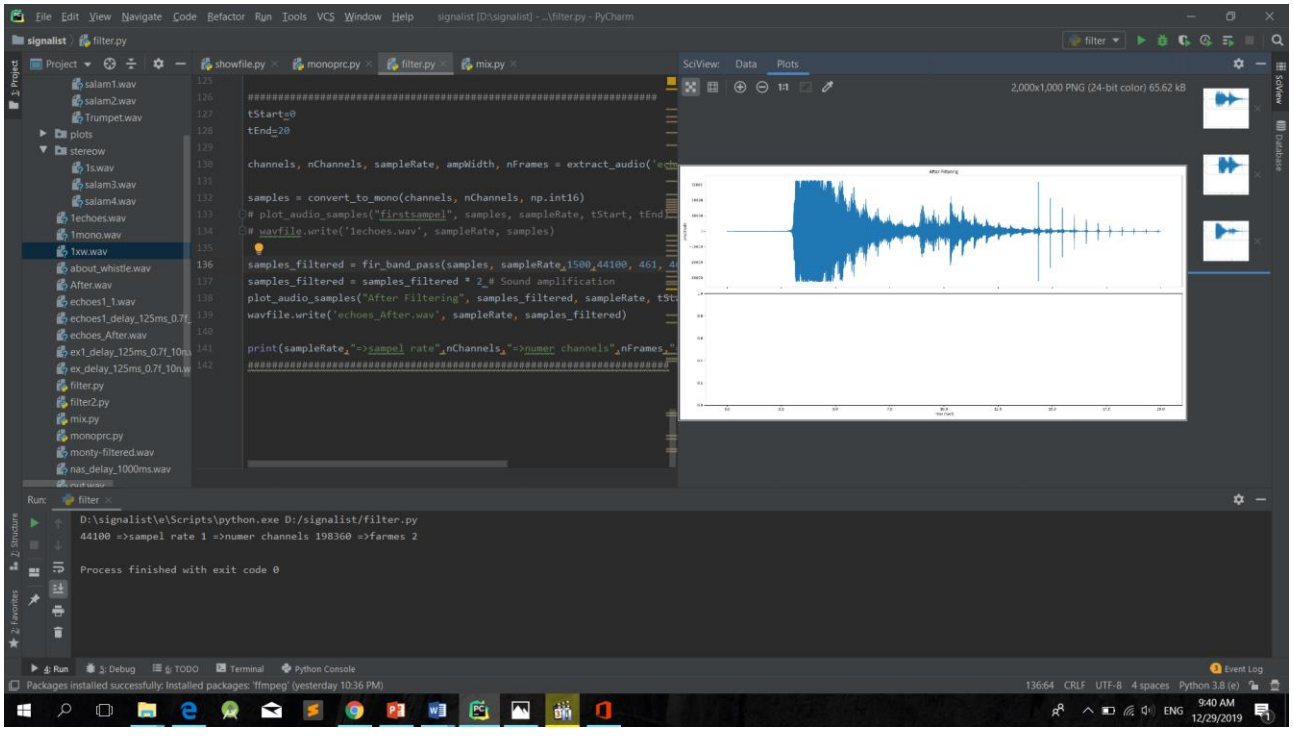
در اینجا این تابع پارامتر های ما را گرفته و تغییرات را یک به یک اعمال می کند و با تابع output_wave خروجی مورد نظر را می سازد.

حال می توانیم فایل اکو داده شده را که زخیره کردیم با کلاس فیلتر در آن فیلتر لازم را اعمال کنیم .

ابتدا سیگنال مورد نظر را مشاهده می کنیم :



حال تغییرات را اعمال و دوباره سیگنال را رسم و به آن گوش می دهیم این بار فرکانس را بین 1500 تا 44100 تنظیم می کنیم :



در پوشه monow فایل های ضبط شده موجود است .

فایل salam1.wav را با کلاس monoprc گرفته

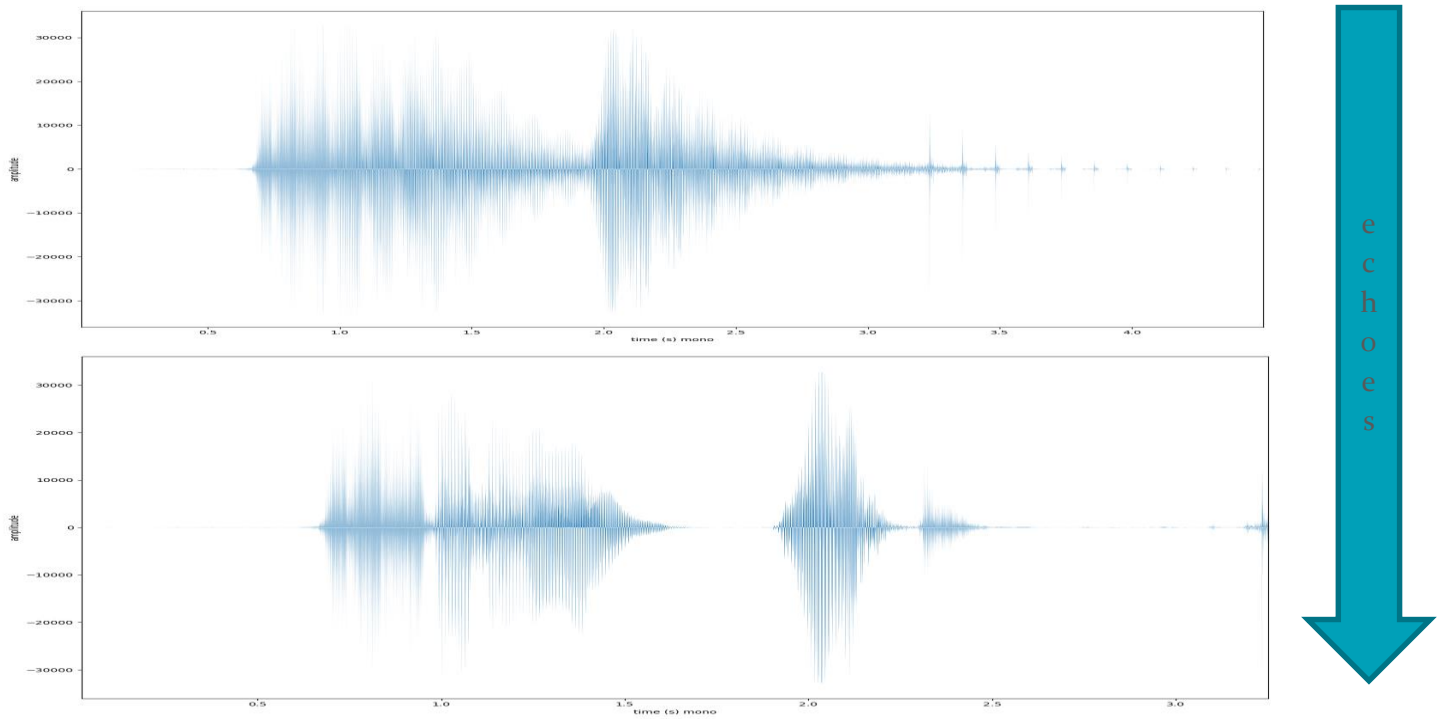
```
tr_params, tr_bytes = input_wave('monow/salam1.wav')
```

پارامتر های مورد نظر را تغییر می دهیم :

```
delay_to_file(tr_bytes, tr_params, offset_ms=125, file_stem='echsalam.wav', factor=.7,
num=10)
print("First 10 bytes:", tr_bytes[:10], sep='\n')
```

در واقع با یک تغییر 125 میلی ثانیه و ضرب عدد 10 در هر بیت پر سکنت صدا اکو دار شده و در پوشه اصلی با اسم echsalam_1.wav ذخیره می شود .

حال می توانیم با کلاس های نوشته شده فایل اکو دار را رسم کنیم.



فایلی داریم در پوشه monow به نام soot که می خواهیم آن را 8000(hz) شیفت دهیم .

لازم به زکر است

برای این کار کلاس filter.py را نوشتیم که توضیح داده شد .

برای رسم تابع در حوضه فرکانسی تابعی در کلاس showfile اضافه کردیم که فرکانس را نمایش می دهد به نام stshow(file)

این تابع از دو کتاب خانه به نام های

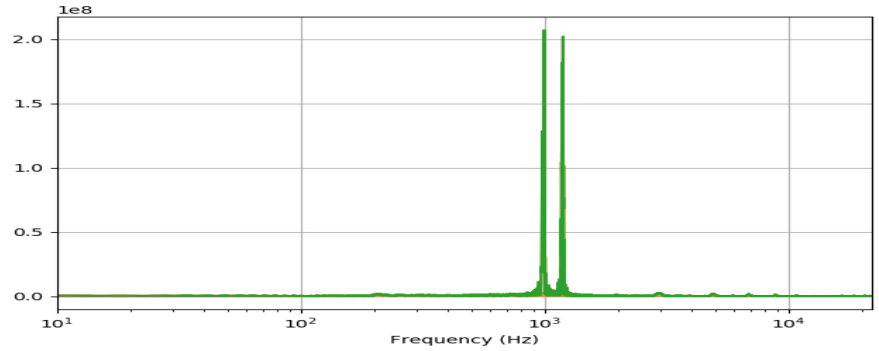
```
import matplotlib.pyplot

from scipy.fftpack import fft,fftfreq
```

استفاده می کند که در واقع برای تبدیل فوریه از آن استفاده می کیم.

```
stshow('monow/soot.wav')
```

فایل را رسم می کنیم :



چون ما دو بار سوت می زنیم و نت دو سوت نزدیک به هم است مشاهده می شود که دو فرکانس نزدیک به هم داریم .

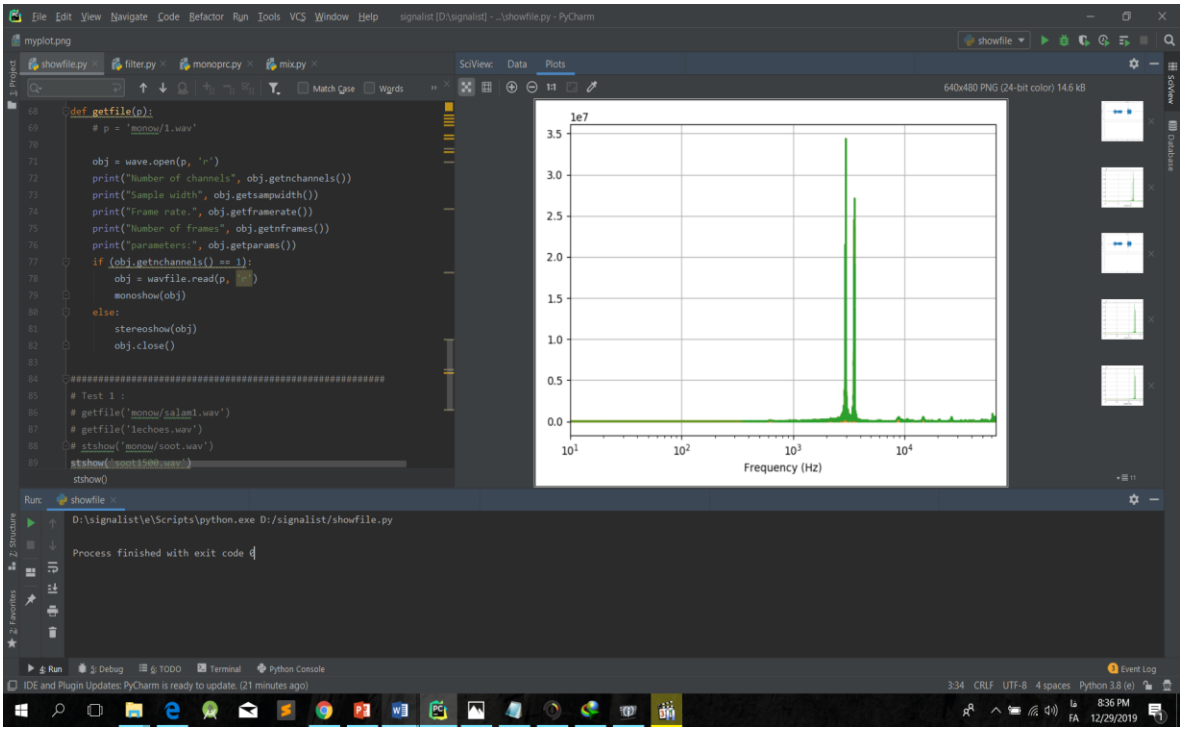
حال صدای صوت را 8000 هرتز شیفت می هدیم :در کلاس filter.py

فایل با soot1500.wav ذخیره می شود.

```
samples_filtered = fir_band_pass(samples, sampleRate,1500,44100,70,70, np.int16)
samples_filtered = samples_filtered * 3 # Sound amplification
sampleRate +=80000
wavfile.write('soot1500.wav', sampleRate +8000, samples_filtered)
plot_audio_samples("After Filtering", samples_filtered , sampleRate, tStart, tEnd)
```

```
sampleRate +=80000
```

مشاهده می شود که سیگنال شیفت داده شد.



پس از گوش دادن به فایل متوجه شدم که با شیفت فرکانس گام صدا تغییر می کند.

حال می خواهیم دو صدا را با هم جمع کنیم :

برای این کار کلاس mix.py نوشتیم .

که دو صدای سلام و سوت را بیت به بیت جمع می کند و به عنوان یک فایل چند صوتی خروجی می دهد .

salam1+soot.wav صدای جمع دو صدای تغییر داده شده است .

حال می خواهیم صدای سوت را حذف کنیم

برای این کار ابتدا با فیلتر نوشته شده یک سیگنال پایین گذر به فایل صوتی می دهیم و صدا فیلتر می شود . سیگنال پایین گذر ما باید پارامتر هایی نزدیک به صدای فیلتر شده سوت داشته باشد .

```
channels, nChannels, sampleRate, ampWidth, nFrames = extract_audio('salam1+soot.wav',
tStart, tEnd)
samples = convert_to_mono(channels, nChannels, np.int16)
lp_samples_filtered = fir_low_pass(samples, sampleRate, 70,461, np.int16)
# First pass
lp_samples_filtered = fir_low_pass(lp_samples_filtered, sampleRate, 70, 461, np.int16)
# Second pass

hp_samples_filtered = fir_high_pass(samples, sampleRate, 1500, 461, np.int16)
# First pass
hp_samples_filtered = fir_high_pass(hp_samples_filtered, sampleRate,1500, 461,
np.int16) # Second pass

samples_filtered = np.mean(np.array([lp_samples_filtered, hp_samples_filtered]),
axis=0).astype(np.int16)
plot_audio_samples("Sultans of Swing - After Filtering 1", samples_filtered,
sampleRate, tStart, tEnd)

wavfile.write('sultans_novoice1.wav', sampleRate, samples_filtered)
```

خروجی بالا فایلی تحت عنوان 'sultans_novoice1.wav

است که به شدت صدای کلی تضعیف شده اما به علت پارامتر های پایین گذر و بالا گذر نزدیک به صدای سوت صدای سوت تا حد امکان حذف شده است .

حالا با همین فیلتر صدا را قدرتمند تر می کنیم تا حدالمکان به صدای اول نزدیک شود.

```
 channels, nChannels, sampleRate, ampWidth, nFrames =
extract_audio('sultans_novoice1.wav', tStart, tEnd)
```

صدای ذخیره شده را دوباره خوانده و پارامتر های آن را افزایش می دهیم :

```
lp_samples_filtered = fir_low_pass(samples, sampleRate, 44100,461, np.int16)
# First pass
lp_samples_filtered = fir_low_pass(lp_samples_filtered, sampleRate, 441000, 461,
np.int16) *4   # Second pass

hp_samples_filtered = fir_high_pass(samples, sampleRate, 66000, 461, np.int16)
# First pass
hp_samples_filtered = fir_high_pass(hp_samples_filtered, sampleRate,66000, 461,
np.int16)
```

```
wavfile.write('nosooot2.wav', sampleRate, samples_filtered *2)
```

خروجی فیلتر افزایش `nosooot2.wav'`

است که صدای صوت به طور کامل حذف شده است .