# تکلیف سری پنجم درس DSP – بخش کامپیوتری

# «مثالهایی ساده از پردازش بلادرنگ»

**نویسنده: مسعود بابایی‌زاده**

**تاریخ: ۲۳ اسفند ۱۳۹۹**

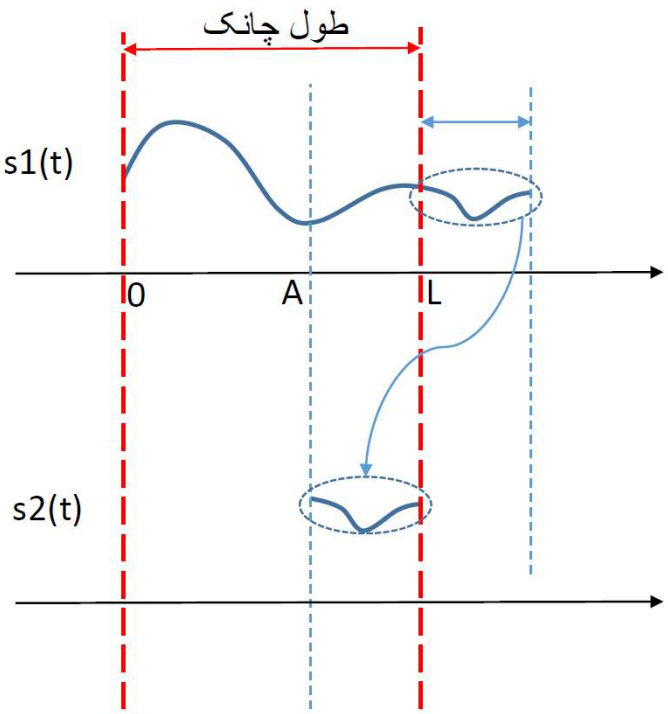
## ۱- مقدمه

این تکلیف در واقع ادامه تکلیف قبلی است. و می‌خواهیم در تابع signal_processing که در تکلیف قبل نوشتیم (و بهتر بود نام آن را chunk_processing می‌گذاشتیم، و در این تکلیف هم با این نام خوانده می‌شود) پردازشهای ساده‌ای (و به صورت Real-Time) انجام دهیم.

## ۲- صورت تکلیف

دو برنامه زیر را بنویسید:

**مساله اول)** در این مسأله، باید سیگنال صحبت را با دور کندتر پخش کنید. در واقع با درونیابی (خطی، مثلا استفاده از تابع interp در numpy ) نمونه‌های میانی را حساب کنید، به طوریکه طول سیگنال این چانک به اندازه پارامتر incPercent زیاد شود (درصد افزایش نسبت به طول چانک). سپس از آنجا که نمی‌خواهیم در خروجی از ورودی عقب بیفتیم می‌توانید به اندازه طول چانک از ابتدای این سیگنال ببرید و به خروجی تابع chunk_processing بفرستید. اما مشکل این روش آن است که بدلیل ناپیوستگیهای سیگنال در مرز چانکهای خروجی، در خروجی صدای تق‌تق (کمی شبیه صدای جرقه زدن) خواهید شنید. لذا روش بهتر آن است که مقدار سیگنال اضافه آمده را، بجای دور ریختن، ترکیب خطی کنیم با انتهای چانک. ضریب این ترکیب خطی هم متغیر است به طوریکه در اول این همپوشانی وزن سیگنال میانی بیشتر است و هرچه به انتها نزدیک می‌شویم وزن سیگنال اضافه. بطوریکه در انتها، نمونه نهایی همان نمونه سیگنال اضافه خواهد بود و ناپیوستگیها از بین خواهد رفت. شکل زیر را ببینید:




در شکل بالا سیگنال s2 همان اضافه سیگنال s1 نسبت به طول چانک است. در اینصورت خروجی کل تابع chunk_processing سیگنال s3 است که در آن s3(t) برای زمانهای قبل از A همان s1(t) است و برای زمانهای بین A تا L برابر است با:

$$S3(t) = alpha * s1(t) + (1-alpha) * s2(t)$$

که در آن ضریب alpha برای نقطه A یک است و سپس به تدریج کم می‌شود تا در نقطه L به صفر می‌رسد.

سعی کنید برنامه را کلی بنویسید که با هر incPercent کار کند. احتمالا تابع np.linspace به کارتان خواهد آمد (برای محاسبه زمانهای جدیدی که در آن به نمونه‌های سیگنال نیاز داریم). سپس خروجی صدای خود را برای مقادیر incPercent برابر ۳۰ درصد و ۵۰ درصد گوش کنید (به صورت Real-Time).

**مساله دوم)** در مساله قبل، کند (تند) پخش کردن سیگنال نه تنها باعث بم (زیر) شدن صدای شما می‌شد، بلکه خود ریتم بیان صحبت نیز کاملا عوض می‌شد. در این مسأله می‌خواهیم صدای شما را زیر یا بم کنیم (مثلا صدای یک مرد به یک زن تبدیل شود یا برعکس) بدون اینکه ریتم بیان صحبت تغییر کند. برای اینکار باید سیگنال تحریک را از اثر فیلترینگ گلو و دهان جدا کنیم. توضیح آنکه سیگنال صحبت انسان، در اثر عبور سیگنالی که توسط ارتعاشات تارهای صوتی تولید می‌شود از یک فیلتر

(اثرات گلو و دهان) به وجود می‌آید. اگر ما این سیگنال تحریک (سیگنال ورودی فیلتر) را از خود فیلتر جدا کنیم، و سپس سیگنال تحریکی با فرکانس (pitch) متفاوت را از همان فیلتر قبلی عبور دهیم، سیگنال صحبت جدیدی به دست می‌آید که pitch متفاوتی دارد (فرکانس pitch آقایان معمولا در محدوده حدود ۸۵ تا ۱۵۵ هرتز است و فرکانس pitch خانمها در محدوده ۱۶۵ تا ۲۵۵ هرتز). البته این فرکانس برای یک فرد ثابت نیست و حرف به حرف (فونم به فونم) کم و زیاد می‌شود.

مثلا در پردازش سیگنال صحبت یکی از روشهای جداسازی سیگنال تحریک از فیلتر استفاده از LPC یا Linear Predictive Coding است. جهت اطلاعات بیشتر در مورد این مدلسازی سیگنال صحبت و استفاده از LPC می‌توانید به لینک زیر مراجعه کنید:

http://isle.illinois.edu/speech_web_lg/coursematerials/ece417/19fall/slides/lec14_lpc10.pdf

اما در این تکلیف نمی‌خواهیم این تغییر pitch را خودمان پیاده‌سازی کنیم و می‌خواهیم از پکیج آماده parselmouth که در واقع یک اینترفیس پایتون است به کتابخانه praat (برای کار پردازش صحبت) استفاده کنیم.

- لینک parcelmouth: https://github.com/YannickJadoul/Parselmouth
- لینک praat: https://www.fon.hum.uva.nl/praat/

در واقع در لینک زیر نشان داده شده است که تغییر pitch با parcelmouth چگونه انجام می‌شود:

https://parselmouth.readthedocs.io/en/stable/examples/pitch_manipulation.html

و در قسمتهای انتهایی آن لینک یک تابع `change_pitch(sound, factor)` نوشته است که شما هم از همان تابع آماده استفاده کنید (و آن را مستقیما روی هر چانک به کار ببرید). در این تابع sound یک آبجت است از نوع parselmouth.Sound (و سیگنال درون آن با sound.values به دست می‌آید)، و factor هم ضریبی است که با آن pitch باید تغییر کند. شما مقادیر factor بین ۰,۲۵ تا حدود ٤ را می‌توانید امتحان کنید (البته توجه داشته باشید که در آن تابع، pitch بین ۷۵ تا ٦۰۰ هرتز فرض شده و اگر pitch جدید خارج از این محدوده باشد، تابع کار خاصی انجام نمی‌دهد).

## ۳- چیزهایی که باید تحویل دهید

- یک برنامه پایتون (یک فایل با پسوند py ) برای مسأله اول.
- یک برنامه پایتون (یک فایل با پسوند py ) برای مسأله دوم.

همه این موارد را به صورت یک فایل فشرده zip تحویل دهید.

## ٤- نکات و راهنمایی‌ها

۱) توابع زیر احتمالا برای شما مفید خواهند بود:

- تبدیل یک آرایه numpy از نوع np.int16 به np.float64 :

```
x = x.astype(np.float64) / 32768
```

(تقسیم بر ۳۲۷۶۸ برای بردن رنج اعداد به محدوده منفی یک تا یک است، که البته در این تکلیف ضرورتی ندارد. اگر انجام دادید، موقع برگشتن به np.int16 هم ضرب در ۳۲۷۶۸ را فراموش نکنید).

- تبدیل یک آرایه numpy به یک آبجکت از نوع parselmouth.Sound :

```
x = parselmouth.Sound(x, Sampling_freq)
```

- تبدیل یک صدای موجود در یک آبجکت از نوع parselmouth.Sound به یک آرایه یک بعدی numpy:

```
y = y.values.reshape(-1)
```

- تبدیل یک آرایه numpy از نوع np.float64 به np.int16 :

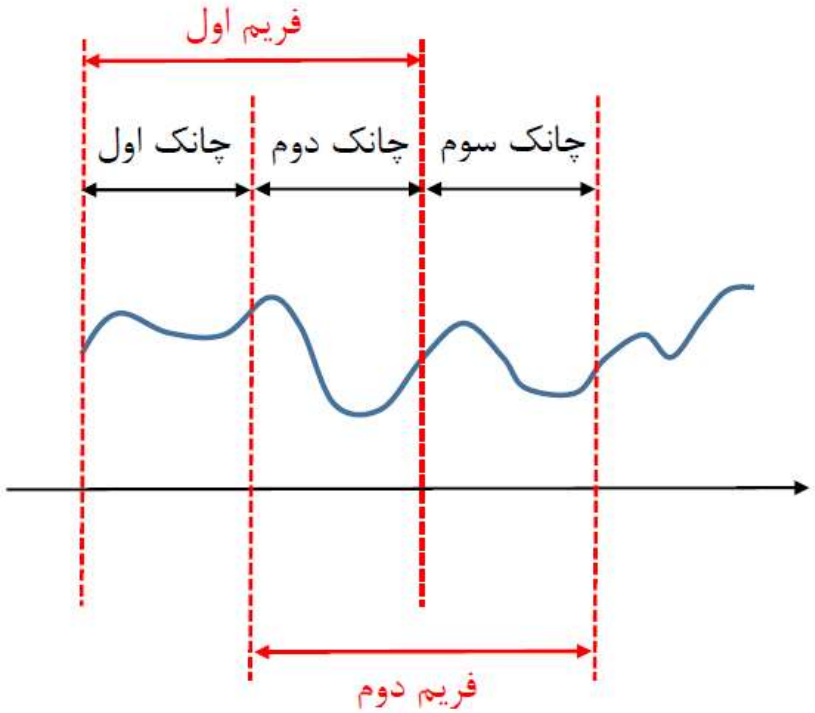
```
y = y.astype(np.int16)
```

۲) در این مسأله هر چانک را جداگانه پردازش کردیم و در نتیجه هر پنجره یا فریم[۱] سیگنال صحبت به صورت جداگانه پردازش شد (در نتیجه بخصوص در مسأله دوم به دلیل ناپیوستگیهای مرز چانکها، کیفیت نهایی خیلی بالا نخواهد شد. بخصوص که در اینجا از تابع آماده‌ای استفاده کردیم که خودش هم فریمهای خودش را دارد و برای کار پردازش فریم به فریم طراحی نشده بوده است[۲]). کلا در پردازش فریم به فریم سیگنال صحبت، خیلی بهتر است فریمهایی را که پردازش می‌کنیم با هم همپوشانی داشته باشند (تا در مرز بین فریمها ناپیوستگی به وجود نیاید). مثلا در حالت ۵۰ درصد همپوشانی، در اینجا اینطور خواهد شد که هر فریم، دو چانک باشد. یعنی همواره باید چانک قبلی ورودی را در حافظه نگه داریم. سپس با رسیدن چانک جدید، چانک قبلی و چانک جدید روی هم یک فریم از سیگنال ما را تشکیل می‌دهند که آن را پردازش می‌کنیم و یک فریم از سیگنال خروجی (که طول آن هم دو چانک است) به دست می‌آید. سپس با رسیدن چانک بعدی، چانک فعلی و چانک بعدی روی هم فریم بعدی ما را تشکیل می‌دهند، که آن هم پردازش می‌شود. در نتیجه سیگنال خروجی متناظر با چانک ورودی فعلی، یک بار با پردازش آن با چانک قبلی محاسبه می‌شود و یک بار هم با پردازش آن با چانک بعدی. چانک خروجی نهایی

[۱] «فریم» در اینجا به معنی متداول آن در پردازش سیگنال است، یعنی یک قسمت (پنجره‌ای) از سیگنال. نه به معنی‌ای که در پکیج pyaudio به کار می‌رود.

[۲] اگر طبق روشی که در اینجا گفته شده عمل کنید، کیفیت کمی بهتر خواهد شد، ولی نه کاملا (بخاطر آن تابع آماده و فریمهای خودش، که برای پردازش فریم به فریم نوشته نشده است).

متناظر با چانک ورودی فعلی، در واقع از ترکیب خطی این دو سیگنال با وزن متغیر (از یک تا صفر) از روی این دو سیگنال خروجی به دست می‌آیند (مشابه ضریب alpha در فرمول مسأله اول این تکلیف). شکل زیر را ببینید:




دقت کنید که با این روش، برای اینکه خروجی متناظر با چانک ورودی فعلی را محاسبه کنیم، باید تا اتمام چانک ورودی بعدی منتظر بمانیم. یعنی در خروجی، یک چانک نسبت به ورودی عقب خواهیم بود.

موفق باشید

مسعود بابایی‌زاده