

جمله انتخابی از کتاب رابینر:

ميانگين معمول نرخ صحبت كردن حدود ۱۵ نماد بر ثانие است.

The normal average rate of speaking is about 15 symbols per second.

(Chapter 1, Page 6, Line 12)

ابتدا، برای تخمین فرکانس گام، مراحل زیر را انجام می‌دهیم:

- فريمبندی و پنجره‌گذاري: سيگنان گفتار به قطعات کوچکتری تقسيم شده و هر قطعه با يك پنجره ضرب می‌شود.
- حذف مقدار DC: مقدار ميانگين قطعه‌ي گفتار برای هر فريم باید محاسبه و از آن حذف شود تا همه‌ي فريمه‌ها در سطح صفر قرار گيرند.
- تشخيص فريم سکوت: فريمه‌اي که حاوي سکوت يا نويز زمينه هستند، با توجه به انرژي فريمه‌اي اول و آخر كل سيگنان شناسايي می‌شوند و از تحليل گفتار حذف می‌شوند.
- تشخيص فريم حاوي گفتار: با محاسبه‌ي انرژي هر فريم و مقاييسه با حد آستانه، فريمه‌اي حاوي گفتار شناسايي می‌شوند. اين کار باید با توجه به نسبت سيگنان به نويز گفتار انجام شود.
- تشخيص واکدار بودن يا نبودن فريم: با استفاده از محاسبه‌ي نرخ عبور از صفر، فريمه‌اي واکدار شناسايي می‌شوند.

### روش :AMDF

تابع AMDF با نزديک شدن به مقدار پريود سيگنان، داري مينيم می‌شود و برای به دست آوردن فرکانس گام، از آن استفاده می‌شود. رابطه آن به شكل زير است:

$$AMDF(\eta) = \frac{1}{N} \sum_{n=0}^{N-1} |S(n) - S(n-\eta)| \quad 0 \leq \eta \leq N - 1$$

با استفاده از فرمول زير، می‌توان فرکانس پيچ را به دست آورد.

$$F_{pitch} = \frac{F_s}{I_{pos}}$$

در اين فرمول،  $F_{pitch}$  نمایانگر فرکانس گام است که برابر است با نسبت نرخ نمونه برداری سيگنان (F<sub>s</sub>) به فاصله اولين دره (I<sub>pos</sub>).

### روش :Autocorrelation

در روش اتوکوروليشن، ابتدا نمودار تغييرات مقدار تابع اتوکوروليشن رسم می‌شود. در اين نمودار، فاصله اولين قله (I<sub>pos</sub>) نشان‌دهنده مقدار پريود گام است. با توجه به اين خاصيت که تابع اتوکوروليشن در نزديکي مقدار پريود سيگنان، داري قله می‌شود، می‌توان از اين فاصله برای تخمين فرکانس گام استفاده کرد.

$$r(\eta) = \frac{1}{N} \sum_{n=0}^{N-1} S(n)^* S(n - \eta) \quad 0 \leq \eta \leq N - 1$$

$$F_{pitch} = \frac{F_s}{I_{pos}}$$

### روش Cepstrum

در روش کپستروم، ابتدا با استفاده از تبدیل فوریه، طیف فرکانسی سیگنال صوتی را به دست می آوریم. سپس با اعمال لگاریتم بر روی طیف فرکانسی، به طیف لگاریتمی می‌رسیم. در مرحله بعد، با اعمال تبدیل معکوس فوریه بر روی طیف لگاریتمی، به سیگنال کپستروم می‌رسیم. در نهایت، با محاسبه ضرایب کپستروم از سیگنال کپستروم، می‌توانیم اطلاعاتی درباره خصوصیات فرکانسی سیگنال صوتی دریافت کنیم. با محاسبه ضرایب کپستروم، اطلاعات مربوط به فرکانس‌ها و قله‌های طیف سیگنال صوتی قابل استخراج است.

$$c[n] = \mathcal{F}^{-1}\{\log|\mathcal{F}\{x[n]\}|\}$$

پس از محاسبه ضرایب کپستروم، می‌توان از یک لیفترا زمان بالا (High-Lifter) بر روی ضرایب کپستروم استفاده کرد. لیفترا زمان بالا یک فیلتر است که برای تقویت قله‌های مهم و کاهش نویز در ضرایب کپستروم به کار می‌رود. با اعمال لیفترا زمان بالا، مقادیر ضرایب کپستروم تغییر می‌کنند و فاصله اولین نقطه اوج قابل توجه یعنی  $I_{pos}$  را می‌توان پیدا کرد. در واقع مقدار پریود گام را نشان می‌دهد. با استفاده از این مقدار، می‌توان فرکانس گام را با استفاده از فرمول زیر محاسبه کرد:

$$F_{pitch} = \frac{F_s}{I_{pos}}$$

با استفاده از این فرمول، می‌توان فرکانس گام را تخمین زد و اطلاعات مربوط به فرکانسی بودن سیگنال صوتی را استخراج کرد.

### توضیح کد:

#### main.m

```
% Read the recorded voice using audioread
[voice, sample_rate] = audioread('myrec.wav');
```

در این بخش، کد سیگنال صدای ضبط شده را از یک فایل با نام 'myrec.wav' با استفاده از تابع audioread می‌خواند. سیگنال در متغیر 'voice' ذخیره می‌شود و نرخ نمونه برداری در متغیر 'sample\_rate' ذخیره می‌شود.

```
% Frame size = 40ms (suitable for pitch detection tasks)
frame_size = 0.04 * sample_rate;
```

در اين بخش، اندازه فريم برای تجزيه و تحليل تعبيين می‌شود. اندازه فريم برابر با 40 ميلیثانیه تنظيم شده است که با ضرب 0.04 در نرخ نمونه برداری محاسبه می‌شود. اين اندازه فريم مناسب برای وظایف تشخیص گام می‌باشد.

% Apply rectangular framing to the voice signal

```
[~, frames] = rectangular_framing(voice, sample_rate, frame_size);
```

در اين بخش، سيگنال صدا با استفاده از تابع 'rectangular\_framing' به فريم‌های همپوشانی تقسیم می‌شود. اين تابع سیگنال 'frame\_size' و 'voice', 'sample\_rate' را به عنوان ورودی می‌گيرد و فريم‌های حاصل را برمی‌گرداند. فريم‌ها در متغیر 'frames' ذخیره می‌شوند.

% Detect voiced frames using short-time energy with a fixed threshold of 0.01

```
voiced_frames = voiced_frame(frames);
```

در اين بخش، فريم‌های داراي گفتار با استفاده از تابع 'voiced\_frame' تشخیص داده می‌شوند. اين تابع فريم‌ها را به عنوان ورودی می‌گيرد و تحليل بر پايه انرژي را اعمال كرده و فريم‌هایی که شامل گفتار هستند را شناسايی می‌کند. فريم‌های داراي گفتار در متغیر 'voiced\_frames' ذخیره می‌شوند.

```
methods = {'AMDF', 'Autocorrelation', 'Cepstrum'};
```

```
functions = {@amdf, @autocorrelation, @cepstrum};
```

```
for i = 1:numel(methods)
```

```
method = methods{i};
```

```
pitch_detection = functions{i};
```

% Call the appropriate pitch detection function

```
pitch = pitch_detection(frames, frame_size, voiced_frames);
```

```
disp(['Method: ' method]);
```

```
disp(['Estimated Pitch: ' num2str(pitch) ' Hz']);
```

```
end
```

در اين بخش، سه روش مختلف برای تشخیص گام وجود دارد: AMDF، اتوکورولیشن و کپستروم. متغیر 'methods' نام هر يك از اين روش‌ها را ذخیره می‌کند و متغیر 'functions' شامل اشاره‌گرهای تابع مربوط به هر روش می‌باشد. با استفاده از يك حلقه، برای هر روش عملیات تشخیص گام صورت می‌گيرد. در هر تكرار، متغیر 'method' نام روش فعلی را دریافت می‌کند و متغیر 'pitch\_detection' شامل اشاره‌گر تابع مربوطه می‌شود. تابع مناسب برای تشخیص گام فراخوانی می‌شود و با ارسال فريم‌ها ('frames')، اندازه فريم ('frame\_size') و فريم‌های داراي گفتار 'pitch' به عنوان آرگومان‌های ورودی. نتيجه حاصل از تشخیص گام در متغیر 'voiced\_frames'

ذخيره می‌شود. در نهايٰت، نام روش و مقدار گام تخمين زده با استفاده از تابع 'disp' نمایش داده می‌شوند.

### **amdf\_.m**

```
function pitch = amdf_(frames, window_size, voiced_frames)
```

این تابع با نام amdf\_ تعریف شده است و سه آرگومان ورودی به نام‌های frames، window\_size و voiced\_frames دریافت می‌کند. این تابع با استفاده از روش AMDF فرکانس گام را محاسبه می‌کند و مقدار تخمين زده شده را برمی‌گرداند.

```
figure(2);
```

```
clf; % Clear the figure before plotting
```

این بخش از کد یک شکل جدید با شماره 2 ایجاد می‌کند و هر محتوای موجود در شکل را پاک می‌کند.

```
pitch_freqs = amdf_pitch(frames, window_size, voiced_frames);
```

این خط کد تابع amdf\_pitch را فراخوانی می‌کند و آرگومان‌های frames، window\_size و voiced\_frames را به عنوان ورودی به آن می‌دهد. مقدار برجسته، pitch\_freqs، نشان‌دهنده فرکانس‌های گامی است که از تحلیل AMDF به‌دست آمده است.

```
% Plot original pitch frequencies
```

```
subplot(2,1,1);
```

```
hold on
```

```
% Plot scatter plot for each pitch candidate
```

```
for candidate_index = 1:5
```

```
candidate = pitch_freqs(candidate_index, :);
```

```
scatter(0:length(candidate)-1, candidate, 20, 'Marker', '.');
```

```
end
```

```
title('AMDF');
```

```
ylabel('Frequency (Hz)');
```

```
set(gca, 'xtick', 0:50:1000);
```

```
hold off
```

در اين قسمت از کد، فرکانس‌های اصلی گامی که از تحلیل AMDF به‌دست آمده‌اند رسم می‌شوند. یک زیرنمودار به ابعاد 2 در 1 ایجاد شده و زیرنمودار اول برای رسم انتخاب می‌شود. نمودار scatter برای نمایش کاندیداهای گام استفاده می‌شود. هر کاندیدا به وسیله یک نمودار scatter نمایش داده می‌شود که محور x نشان‌دهنده شاخص فریم و محور y نشان‌دهنده فرکانس گام است.

```

subplot(2,1,2);

% Smooth the pitch frequencies
smoothed = nan(5, length(pitch_freqs));
% Apply center-clipping and smoothing
for frame_index = 1:length(pitch_freqs)
    for candidate_index = 1:5
        if pitch_freqs(candidate_index, frame_index) <= 160
            smoothed(candidate_index, frame_index) = pitch_freqs(candidate_index,
frame_index);
        end
    end
end

```

در اين قسمت، يك زيرنومدار جديد ايجاد شده و برای رسم انتخاب می‌شود. ماترييس smoothed با مقادير NaN مقداردهي اوليه می‌شود. سپس يك حلقه تمامی فريمها و کانديداهاي گام را تکرار می‌کند و بررسی می‌کند که آيا فرکانس گام کمتر یا مساوی ۱۶۰ است یا خير. اگر کمتر یا مساوی ۱۶۰ باشد، مقدار متناظر در ماترييس smoothed با مقدار فرکانس گام بروزرسانی می‌شود.

```

smoothed = smooth2a(smoothed, 5, 5);
max_values = nan(1, length(smoothed));
% Find the maximum value in each frame
for frame_index = 1:length(smoothed)
    for candidate_index = 1:5
        if ~isnan(smoothed(candidate_index, frame_index))
            max_values(frame_index) = max(smoothed(candidate_index, frame_index));
        end
    end
end

```

در اين بخش، ماترييس smoothed بيشتر پردازش می‌شود. آن را به تابع smooth2a با پaramترهاي ۵ و ۵ برای smooth کردن مقادير ارسال می‌کنيم. سپس، يك حلقه تمامی فريمها و کانديداهاي گام را تکرار می‌کند و بيشينه‌ی مقدار در هر فريم را از مقادير انعکاس‌یافته پيدا می‌کند. مقادير بيشينه در آرایه max\_values ذخیره می‌شوند.

```

pitch = nanmedian(max_values);

```

این خط کد با گرفتن میانه آرایه max\_values، مقدار گام را محاسبه می‌کند. متغیر pitch مقدار تخمین زده شده گام را نگهداری می‌کند.

```
% Plot the smoothed contour
plot(0:length(pitch_freqs)-1, max_values, 'linewidth', 2);
title('AMDF (Smoothed Contour)');
xlabel('Frame number');
set(gca, 'xtick', 0:50:1000);
ylabel('Frequency (Hz)');
ylim([0 400]);
```

در اين قسمت، نمودار مربوط به خطی از مقادیر smooth شده پیچ کانتور را رسم می‌کند. يك نمودار خطی با محور x نشان‌دهنده شماره فریم و محور y نشان‌دهنده بیشینه‌ی مقادیر گام رسم می‌شود. محدوده محور y به [0 400] تنظیم شده است.

### amdf\_pitch.m

```
function pitch_freqs = amdf_pitch(frames, window_size, voiced)
% Computes pitch candidates using AMDF (Average Magnitude Difference Function)
method.

% Initialize pitch_freqs matrix to store pitch candidates
pitch_freqs = zeros(5, length(voiced));

% Iterate over frames
for i = 1:length(voiced)
    if voiced(i) == 1
        % Extract current frame and apply center-clipping
        clipped_frame = center_clipping(frames(i, 1:window_size));
        % Compute AMDF for the clipped frame
        amdf_result = amdf(clipped_frame);
        % Assign pitch candidates to pitch_freqs matrix
        pitch_freqs(:, i) = five_pitch(amdf_result);
    end
end

% Remove pitch candidates above 500 Hz and below 75 Hz
for i = 1:length(pitch_freqs)
    for j = 1:5
```

```

if pitch_freqs(j, i) >= 600 || pitch_freqs(j, i) <= 75
    pitch_freqs(j, i) = nan;
end
end
end
end

```

این تابع amdf\_pitch مقادیر کاندید گام را با استفاده از روش AMDF محاسبه می‌کند. ابتدا یک ماتریس به نام pitch\_freqs برای ذخیره کاندیدهای گام ایجاد می‌کند. سپس بر روی فریم‌ها تکرار می‌کند و اگر فریم صدایی باشد (بر اساس آرایه voiced)، فریم فعلی را استخراج کرده، center-clipping را اعمال می‌کند، AMDF را برای فریم برش‌داده شده محاسبه می‌کند و کاندیدهای گام را به ماتریس pitch\_freqs اختصاص می‌دهد. در نهایت، کاندیدهای گام بالای 500 هرتز و پایین‌تر از 75 هرتز حذف می‌شوند.

```

function amdf_result = amdf(frame)
% Computes the AMDF (Average Magnitude Difference Function) for a given frame.
amdf_result = zeros(1, length(frame));
% Compute AMDF values
for i = 0:length(frame)-1
    amdf_result(i+1) = sum(abs(frame(i+1:end) - frame(1:end-i)));
end
% Normalize the AMDF result
amdf_result = amdf_result/max(amdf_result);
end

```

این تابع amdf را برای یک فریم داده شده محاسبه می‌کند. ابتدا یک آرایه به نام amdf\_result با صفرها برای ذخیره مقادیر AMDF ایجاد می‌شود. سپس بر روی اندیس‌های فریم تکرار می‌کند و مقدار AMDF را در هر اندیس با جمع اختلاف‌های مطلق بین نمونه‌های فریم محاسبه می‌کند. در نهایت، نتیجه AMDF را با تقسیم آن بر بیشینه مقدار normalize می‌کند.

```

function pitch_candidates = five_pitch(frame)
% Extracts five pitch candidates from the AMDF result.
pitch_candidate = zeros(1,5);
% Smooth using moving median filter
window_size = 15;
frame = medfilt1(frame, window_size);

```

```
% Find negative peaks as pitch candidates
[peaks, peaks_locs] = findpeaks(-frame);
for i = 1:length(peaks)
    max_peaks = max(peaks);
    index = -1;
    for j = 1:length(peaks)
        if peaks(j) >= max_peaks
            index = j;
        end
    end
    peaks(index) = -1;
    pitch_candidate(i) = peaks_locs(index);
end

% Sort the pitch candidates in ascending order
pitch_candidate = sort(pitch_candidate);
pitch_cadidates = zeros(1,5);

% Convert pitch candidates to frequencies
for i = 1:5
    pitch_cadidates(i) = 640 / (pitch_candidate(i) * 0.04);
end

end
```

این تابع five\_pitch پنج کاندید گام را از نتیجه AMDF استخراج می‌کند. ابتدا یک آرایه به نام pitch\_candidate برای ذخیره مقادیر کاندید ایجاد می‌کند. با استفاده از فیلتر میانه، AMDF را با یک پنجره به طول ۱۵ هموار می‌کند. سپس نقاط عملکرد منفی را در نتیجه هموار پیدا می‌کند و آنها را به عنوان کاندیدهای گام در نظر می‌گیرد. کاندیدهای گام را به ترتیب صعودی مرتب می‌کند و آنها را به فرکانس‌ها تبدیل می‌کند. در نهایت، آرایه pitch\_cadidates حاوی پنج کاندید گام را برمی‌گرداند.

#### **autocorrelation.m**

```
function pitch = autocorrelation(frames, frame_size, voiced_frames)
% Perform autocorrelation-based pitch detection with 3-level center-clipping
figure(1);
clf; % Clear the figure before plotting
```

این تابع با استفاده از روش autocorrelation با 3-level center-clipping تشخیص فرکانس پیج را انجام می‌دهد. یک شکل جدید برای نمایش نتایج ایجاد می‌کند.

```
% Compute autocorrelation pitch frequencies
```

```
pitch_freqs = autocorrelation_pitch(frames, frame_size, voiced_frames);
```

در این خط، فرکانس‌های پیچیدگی مبتنی بر autocorrelation با استفاده از تابع autocorrelation\_pitch محاسبه می‌شوند.

```
% Plot original pitch frequencies
```

```
subplot(2,1,1);
```

```
hold on
```

```
% Plot scatter plot for each candidate
```

```
for candidate_index = 1:5
```

```
candidate = pitch_freqs(candidate_index, :);
```

```
scatter(0:length(candidate)-1, candidate, 20, 'Marker', '.');
```

```
end
```

```
title('Autocorrelation with 3-level center-clipping');
```

```
ylabel('Frequency (Hz)')
```

```
set(gca,'xtick',0:50:1000)
```

```
hold off
```

این بخش، فرکانس‌های گام اصلی را در یک زیرنمودار نمایش می‌دهد. از نمودار scatter برای نمایش هر کاندید پیج استفاده می‌کند. محور x شماره فریم را و محور y فرکانس را نشان می‌دهد.

```
% Smooth the pitch frequencies
```

```
subplot(2,1,2);
```

```
pitch_freqs = autocorrelation_pitch(frames, frame_size, voiced_frames);
```

```
smoothed = nan(5, length(pitch_freqs));
```

```
% Apply 3-level center-clipping and smoothing
```

```
for frame_index = 1:length(pitch_freqs)
```

```
for candidate_index = 1:5
```

```
if pitch_freqs(candidate_index, frame_index) <= 160
```

```
smoothed(candidate_index, frame_index) = pitch_freqs(candidate_index, frame_index);
```

```
end
```

```
end
```

```
end
```

```
smoothed = smooth2a(smoothed, 5, 5);
```

این بخش، فرکانس‌های پیج به دست آمده از autocorrelation را صاف می‌کند. یک زیرنحوه جدید برای فرکانس‌های پیج صاف شده ایجاد می‌کند. 3-level center-clipping اعمال می‌شود، به طوری که فرکانس‌های بالاتر از 160 هرتز به NaN (عدد ناشناخته) تبدیل می‌شوند. سپس فرکانس‌های پیج نتیجه را با استفاده از یکتابع smoothing دو بعدی صاف می‌کند.

```
% Find the maximum value in each frame
```

```
max_values = nan(1, length(smoothed));
```

```
for frame_index = 1:length(smoothed)
```

```
    for candidate_index = 1:5
```

```
        if ~isnan(smoothed(candidate_index, frame_index))
```

```
            max_values(frame_index) = max(smoothed(candidate_index, frame_index));
```

```
        end
```

```
    end
```

```
end
```

```
pitch = nanmedian(max_values);
```

این بخش، بیشینه‌ی هر فریم از فرکانس‌های پیج صاف شده را پیدا می‌کند. یک آرایه برای ذخیره بیشینه‌ها ایجاد می‌شود. در هر فریم و کاندید پیج، مقادیری که NaN نیستند برسی می‌شوند و مقدار بیشینه در صورت لزوم به روزرسانی می‌شود. در نهایت، میانه‌ی بیشینه‌ها به عنوان مقدار نهایی پیج محاسبه می‌شود.

```
% Plot the smoothed contour
```

```
plot(0:length(pitch_freqs)-1, max_values, 'linewidth', 2);
```

این خط، منحنی پیج صاف شده را با استفاده از مقادیر بیشینه‌ی به دست آمده در مرحله قبل نمایش می‌دهد. محور x شماره فریم را و محور y فرکانس را نشان می‌دهد.

```
title('Smoothed Autocorrelation Pitch Contour');
```

```
xlabel('Frame number')
```

```
set(gca,'xtick',0:50:1000)
```

```
ylabel('Frequency (Hz)')
```

```
ylim([0 400])
```

```
% Adjust subplot positions and sizes
```

```
subplot(2,1,1);
```

```
pos = get(gca, 'position');
```

```

set(gca, 'position', [pos(1) pos(2)-0.03 pos(3) pos(4)+0.03]);
subplot(2,1,2);
pos = get(gca, 'position');
set(gca, 'position', [pos(1) pos(2)-0.06 pos(3) pos(4)+0.06]);
end

```

این بخش، برچسبها و ظاهر زیرنومودارها را تنظیم می‌کند. محدوده‌ی محور y به [۰, ۴۰۰] تنظیم می‌شود تا نمایش صحیح انجام شود. موقعیت‌ها و اندازه‌های زیرنومودارها برای ترازبندی بهتر تنظیم می‌شوند.

### **autocorrelation\_pitch.m**

```

function pitch_freqs = autocorrelation_pitch(frames, frame_size, voiced_frames)
    % Calculate pitch frequencies using autocorrelation method with center-clipping
    % Initialize pitch frequencies matrix
    pitch_freqs = zeros(5, length(voiced_frames));
    % Iterate over frames
    for i = 1:length(voiced_frames)
        if voiced_frames(i) == 1
            clipped_frame = three_level_center_clipping(frames(i, 1:frame_size));
            % Compute autocorrelation
            autocorrelation_result = autocorr(clipped_frame, frame_size - 1);
            % Extract pitch frequencies
            pitch_freqs(:, i) = extract_pitch(autocorrelation_result);
        end
    end
    % Remove pitch frequencies above 500Hz and below 75Hz
    for i = 1:length(pitch_freqs)
        for j = 1:5
            if pitch_freqs(j, i) >= 600 || pitch_freqs(j, i) <= 75
                pitch_freqs(j, i) = nan;
            end
        end
    end
end

```

این تابع فرکانس‌های پیچ را با استفاده از روش autocorrelation به همراه برش مرکزی محاسبه می‌کند. سه ورودی را دریافت می‌کند: frames (سیگنال ورودی که به قسمت‌هایی تقسیم شده است)، frame\_size (اندازه هر قسمت) و voiced\_frames (یک بردار دودویی که قسمت‌های دارای صدا/بی‌صدا را نشان می‌دهد). یک ماتریس با نام pitch\_freqs برای ذخیره فرکانس‌های پیچ برای هر قسمت ایجاد می‌کند. با تابع three\_level\_center\_clipping تا برش مرکزی را اعمال می‌کند. autocorrelation قسمت برش خورده را با استفاده از تابع autocorr محاسبه می‌کند.

با تابع extract\_pitch پنج تا از قوی‌ترین فرکانس‌های پیچی را از نتیجه autocorrelation استخراج کند. فرکانس‌های پیچ استخراج شده را در ماتریس pitch\_freqs ذخیره می‌کند. در نهایت، هر فرکانس پیچیدگی که بیشتر از 500 هرتز یا کمتر از 75 هرتز است، با تنظیم آنها به nan حذف می‌شود.

```
function pitch_freqs = extract_pitch(autocorr_result)
    % Extract the five strongest pitch frequencies from the autocorrelation result
    % Initialize pitch frequencies array
    pitch_freqs = zeros(1, 5);
    % Find peaks in the autocorrelation result
    [peaks, locations] = findpeaks(autocorr_result);
    % Extract the five strongest pitch frequencies
    for i = 1:5
        [~, index] = max(peaks);
        pitch_freqs(i) = locations(index);
        peaks(index) = 0; % Set the maximum peak to 0 to find the next maximum
    end
    % Sort the pitch frequencies in ascending order
    pitch_freqs = sort(pitch_freqs);
    % Convert pitch frequencies to Hz
    pitch_freqs = 640 ./ (pitch_freqs * 0.04);
end
```

این تابع پنج تا از قوی‌ترین فرکانس‌های پیچ را از نتیجه اتوکورولیشن استخراج می‌کند. یک ورودی به نام autocorr\_result (نتیجه اتوکورولیشن یک قسمت) دریافت می‌کند. یک آرایه به نام pitch\_freqs برای ذخیره فرکانس‌های پیچیدگی ایجاد می‌کند. با استفاده از تابع findpeaks، نقاط بالایی در نتیجه اتوکورولیشن پیدا می‌شود. پنج فرکانس با قوی‌ترین پیچ با استفاده از پیدا کردن قله

بیشترین، ذخیره کردن مکان آن و تنظیم آن به ۰ برای پیدا کردن بیشترین بعدی به صورت تکراری استخراج می‌شوند. فرکانس‌های پیچیدگی استخراج شده به ترتیب صعودی مرتب می‌شوند. فرکانس‌های پیچیدگی به هر تر تبدیل می‌شوند با استفاده از فرمول: فرکانس (هرتز) =  $640 / \text{دوره پیچیدگی} * 0.04$ ، که دوره pitch\_period در نمونه‌ها است.

```
function clipped_frame = three_level_center_clipping(frame)
    % Apply three-level center-clipping to the input frame
    % Initialize clipped frame
    clipped_frame = zeros(1, length(frame));
    % Find the maximum amplitude in the frame
    maximum = max(abs(frame));
    clipping_factor = 0.1;
    % Apply three-level center-clipping
    for i = 1:length(clipped_frame)
        if frame(i) >= clipping_factor * maximum
            clipped_frame(i) = frame(i) - clipping_factor * maximum;
        elseif frame(i) <= -clipping_factor * maximum
            clipped_frame(i) = frame(i) + clipping_factor * maximum;
        end
    end
end
```

این تابع 3-level center-clipping را روی قسمت ورودی اعمال می‌کند. یک ورودی به نام frame (قسمت ورودی) دریافت می‌کند. یک متغیر به نام clipped\_frame برای ذخیره قسمت برش خورده ایجاد می‌کند. بزرگترین amplitude را با استفاده از تابع max پیدا می‌کند. یک ضریب برش را (clipping\_factor) با مقدار 0.1 تعریف می‌کند. برش مرکزی سه سطحی را با تکرار بر روی هر نمونه در قسمت اعمال می‌کند. اگر نمونه بزرگتر یا مساوی clipping\_factor \* maximum باشد، مقدار clipping\_factor \* maximum را از نمونه کم می‌کند. اگر نمونه کوچکتر یا مساوی clipping\_factor \* maximum باشد، مقدار clipping\_factor \* maximum را به نمونه اضافه می‌کند و در نهایت قسمت برش خورده حاصل را برمی‌گرداند.

### **center\_clipping.m**

```
function frame = center_clipping(input_frame)
    frame = zeros(1, length(input_frame));
    max_amplitude = max(abs(input_frame));
```

```

clipping_factor = 0.3;
for i = 1:length(frame)
    if input_frame(i) >= clipping_factor * max_amplitude
        % Perform positive clipping
        frame(i) = input_frame(i) - clipping_factor * max_amplitude;
    elseif input_frame(i) <= -(clipping_factor * max_amplitude)
        % Perform negative clipping
        frame(i) = input_frame(i) + clipping_factor * max_amplitude;
    end
end
end

```

این تابع برش مرکزی را روی قسمت ورودی اعمال می‌کند. یک ورودی به نام `input_frame` (قسمت ورودی) دریافت می‌کند. یک متغیر به نام `frame` برای ذخیره قسمت برش خورده ایجاد می‌کند. بزرگترین `amplitude` را در قسمت ورودی با استفاده از تابع `max` و `abs` پیدا می‌کند. یک ضریب برش را (`clipping_factor`) با مقدار 0.3 تعریف می‌کند. برش مرکزی را با تکرار بر روی هر نمونه در قسمت ورودی اعمال می‌کند: اگر نمونه بزرگتر یا مساوی `clipping_factor * max_amplitude` باشد، مقدار `clipping_factor * max_amplitude` را از نمونه کم می‌کند (برش مرکزی مثبت). اگر نمونه کوچکتر یا مساوی `clipping_factor * max_amplitude - clipping_factor * max_amplitude` باشد، مقدار `clipping_factor * max_amplitude - max_amplitude` را به نمونه اضافه می‌کند (برش مرکزی منفی) و در نهایت قسمت برش خورده حاصل را برمی‌گرداند.

### **cepstrum.m**

```

function pitch_freqs = cepstrum_pitch(frames, frame_size, voiced_frames)
    % Calculate pitch frequencies using cepstrum method with center-clipping
    % Initialize pitch frequencies matrix
    pitch_freqs = zeros(5, length(voiced_frames));
    % Iterate over frames
    for i = 1:length(voiced_frames)
        if voiced_frames(i) == 1
            clipped_frame = three_level_center_clipping(frames(i, 1:frame_size));
            % Compute cepstrum
            cepstrum_result = real(ifft(log(abs(fft(clipped_frame)))));
            % Extract pitch frequencies

```

```

pitch_freqs(:, i) = extract_pitch(cepstrum_result);
end
end
% Remove pitch frequencies above 500Hz and below 75Hz
for i = 1:length(pitch_freqs)
    for j = 1:5
        if pitch_freqs(j, i) >= 600 || pitch_freqs(j, i) <= 75
            pitch_freqs(j, i) = nan;
        end
    end
end
end

```

این تابع با استفاده از روش سپسٹر و با اعمال برش مرکزی، فرکانس‌های نت را محاسبه می‌کند. ماتریسی به نام pitch\_freqs برای ذخیره فرکانس‌های پیچ ایجاد می‌شود. بر روی فریم‌ها تکرار می‌کند و بررسی می‌کند که آیا یک فریم دارای صدای گفتار است ( $\text{voiced\_frames}(i) == 1$ ). برش مرکزی را با استفاده از تابع three\_level\_center\_clipping به فریم اعمال می‌کند. فریم را با استفاده از تبدیل فوریه معکوس از لگاریتم قدر مطلق تبدیل فوریه محاسبه می‌کند. فرکانس‌های پیچ را با استفاده از تابع extract\_pitch استخراج می‌کند. فرکانس‌های پیچ را در ماتریس pitch\_freqs ذخیره می‌کند. فرکانس‌های نتی که بالای 500 هرتز یا پایین‌تر از 75 هرتز هستند را با تعیین مقدار nan حذف می‌کند.

```

function pitch_freqs = extract_pitch(cepstrum_result)
    % Extract the five strongest pitch frequencies from the cepstrum result
    % Initialize pitch frequencies array
    pitch_freqs = zeros(1, 5);
    % Find peaks in the cepstrum result
    [peaks, locations] = findpeaks(cepstrum_result);
    % Extract the five strongest pitch frequencies
    for i = 1:5
        [~, index] = max(peaks);
        pitch_freqs(i) = locations(index);
        peaks(index) = 0; % Set the maximum peak to 0 to find the next maximum
    end

```

```
% Sort the pitch frequencies in ascending order
pitch_freqs = sort(pitch_freqs);

% Convert pitch frequencies to Hz
pitch_freqs = 640 ./ (pitch_freqs * 0.04);

end
```

این تابع پنج فرکانس قویتر را از نتیجه استخراج می‌کند. آرایه‌ای به نام `pitch_freqs` برای ذخیره فرکانس‌های پیچ ایجاد می‌شود. نقاط برجسته را در نتیجه `cepstrum` با استفاده از تابع `findpeaks` پیدا می‌کند. پنج فرکانس پیچ قویتر را با پیدا کردن قله‌های بیشینه به صورت مشابه استخراج می‌کند. فرکانس‌های پیچ را در آرایه `pitch_freqs` ذخیره می‌کند. فرکانس‌های پیچ را به ترتیب صعودی مرتب می‌کند. فرکانس‌های پیچ را با استفاده از فرمول  $640 / (pitch\_freqs * 0.04)$  به هرتز تبدیل می‌کند.

```
function clipped_frame = three_level_center_clipping(frame)

% Apply three-level center-clipping to the frame

% Calculate the clipping thresholds

max_val = max(frame);
min_val = min(frame);
upper_threshold = max_val - (max_val - min_val) / 3;
lower_threshold = min_val + (max_val - min_val) / 3;

% Apply center-clipping

clipped_frame = frame;
clipped_frame(clipped_frame > upper_threshold) = upper_threshold;
clipped_frame(clipped_frame < lower_threshold) = lower_threshold;

end
```

این تابع برش مرکزی سه سطحی را به فریم ورودی اعمال می‌کند. آستانه‌های برش را بر اساس حداقل و حداقل مقادیر فریم محاسبه می‌کند. آستانه بالا را به عنوان دو سوم فاصله بین حداقل و حداقل مقادیر کمتر از حداقل را تعریف می‌کند. آستانه پایین را به عنوان یک سوم فاصله بین حداقل و حداقل مقادیر اضافه به حداقل تعریف می‌کند. برش مرکزی را با جایگزین کردن مقادیر در فریم که بیشتر از آستانه بالا هستند با مقدار آستانه بالا و مقادیری که کمتر از آستانه پایین هستند با مقدار آستانه پایین اعمال می‌کند. فریم برش خورده به عنوان خروجی بازگردانده می‌شود.

### **cepstrum\_pitch.m**

```
function pitch_freqs = cepstrum_pitch(frames, window_size, voiced_frames)

% Compute cepstrum pitch frequencies for voiced frames
```

```
% Initialize pitch frequencies matrix
pitch_freqs = zeros(5, length(voiced_frames));

% Process each voiced frame
for i = 1:length(voiced_frames)

    if voiced_frames(i) == 1

        % Apply center-clipping to the frame
        clipped_frame = center_clipping(frames(i, 1:window_size));

        % Compute real cepstrum of the clipped frame
        cepstrum = rceps(clipped_frame);

        % Extract the five pitch candidates from the cepstrum
        pitch_freqs(:, i) = extract_pitch_candidates(cepstrum);

    end

end

% Remove pitches above 500Hz and below 75Hz
pitch_freqs(pitch_freqs >= 600 | pitch_freqs <= 75) = NaN;

end
```

این تابع فرکانس‌های پیج cepstrum را برای فریم‌های صوتی محاسبه می‌کند. یک ماتریس به نام pitch\_freqs برای ذخیره فرکانس‌های پیج ایجاد می‌شود که ابعاد آن ۵ ردیف (نماینده پنج کاندید پیج) و تعداد ستون‌های طول voiced\_frames است. در هر مرحله به ازای هر فریم، بررسی می‌شود که آیا آن فریم یک فریم صوتی است یا خیر (با استفاده از آرایه voiced\_frames). برای هر فریم صوتی، برش مرکزی به فریم اعمال می‌شود با استفاده از تابع center\_clipping. سپس cepstrum واقعی از فریم برش‌خورده با استفاده از تابع rceps محاسبه می‌شود. پس از آن، پنج کاندید پیج را از cepstrum استخراج می‌کند با استفاده از تابع extract\_pitch\_candidates. فرکانس‌های پیج استخراج شده در ماتریس pitch\_freqs ذخیره می‌شوند. پیج‌هایی که بیشتر از ۵۰۰ هرتز یا کمتر از ۷۵ هرتز هستند، از ماتریس pitch\_freqs حذف می‌شوند و به جای آن به NaN تغییر می‌دهد.

```
function pitch_candidates = extract_pitch_candidates(cepstrum)

    % Extract the five pitch candidates from the cepstrum
    pitch_candidates = zeros(1, 5);

    % Discard the first four coefficients to remove the DC offset
    cepstrum = cepstrum(1, 5:end);

    % Find the peaks in the cepstrum with a minimum peak distance of 50 samples
```

```
[peaks, peak_locs] = findpeaks(cepstrum, 'MinPeakDistance', 50);
% Select the five highest peaks as pitch candidates
for i = 1:5
    [~, max_index] = max(peaks);
    pitch_candidates(i) = 640 / (peak_locs(max_index) * 0.04);
    peaks(max_index) = []; % Remove the selected peak
    peak_locs(max_index) = []; % Remove the corresponding peak location
end
end
```

این تابع پنج کاندید پیج را از cepstrum استخراج می‌کند. یک آرایه به نام pitch\_candidates برای ذخیره کاندید پیج ایجاد می‌شود. چهار ضریب اول cepstrum حذف می‌شوند تا تفاوت DC را حذف کنند. با استفاده از تابع findpeaks، بر روی cepstrum، در یافتن نقاط برجسته با حداقل فاصله بین برجستگی‌ها برابر با  $50$  نمونه انجام می‌شود. پنج تای بالاتر به عنوان کاندید پیج انتخاب می‌شوند. مقدار پیج‌ها با تقسیم  $640$  بر حاصل ضرب شاخص نمونه‌های انتخاب شده با  $0.04$  محاسبه می‌شوند. مورد برجسته انتخاب شده و مکان متضاظر آن از آرایه‌ها حذف می‌شوند. کاندید پیج‌ها در آرایه pitch\_candidates ذخیره می‌شوند.

```
function clipped_frame = center_clipping(input_frame)
    % Apply center-clipping to the input frame
    clipped_frame = zeros(1, length(input_frame));
    maximum = max(abs(input_frame));
    clipping_factor = 0.3;
    for i = 1:length(clipped_frame)
        if input_frame(i) >= clipping_factor * maximum
            % Positive clipping
            clipped_frame(i) = input_frame(i) - clipping_factor * maximum;
        elseif input_frame(i) <= -(clipping_factor * maximum)
            % Negative clipping
            clipped_frame(i) = input_frame(i) + clipping_factor * maximum;
        end
    end
    function clipped_frame = center_clipping(input_frame)
```

```
% Apply center-clipping to the input frame
clipped_frame = zeros(1, length(input_frame));
maximum = max(abs(input_frame));
clipping_factor = 0.3;
for i = 1:length(clipped_frame)
    if input_frame(i) >= clipping_factor * maximum
        % Positive clipping
        clipped_frame(i) = input_frame(i) - clipping_factor * maximum;
    elseif input_frame(i) <= -(clipping_factor * maximum)
        % Negative clipping
        clipped_frame(i) = input_frame(i) + clipping_factor * maximum;
    end
end
end
```

این تابع کلیپ برش را به فریم ورودی اعمال می‌کند. یک آرایه به نام clipped\_frame برای ذخیره فریم کلیپ شده ایجاد می‌شود. حداکثر مقدار مطلق فریم ورودی با استفاده از تابع max محاسبه می‌شود. ضریب clipping برابر  $30^\circ$  تعریف می‌شود. در هر نمونه از فریم ورودی، بررسی می‌شود که آیا مقدار آن از آستانه clipping مثبت یا منفی عبور می‌کند. اگر نمونه از آستانه clipping مثبت عبور کند، از مقدار آستانه ضرب شده در حداکثر مقدار کم می‌شود. اگر نمونه از آستانه کلیپ منفی عبور کند، به مقدار آستانه ضرب شده در حداکثر مقدار اضافه می‌شود. فریم کلیپ شده حاصل در آرایه clipped\_frame ذخیره می‌شود.

### **rectangular\_framing.m**

```
function [t, f] = rectangular_framing(s, fs, frame_length)
%RECTANGULAR_FRAMING Perform rectangular framing on a signal.
% [t, f] = RECTANGULAR_FRAMING(s, fs, frame_length) performs framing on the
% input signal 's' with a given frame length 'frame_length' and sampling
% frequency 'fs'. It returns a frame matrix 'f' containing the framed segments
% of the signal and a time vector 't' representing the corresponding time
% instants of each frame.
% 50 percent overlap
hop = frame_length / 2;
```

```
% Calculate the number of frames
num_frame = floor((length(s) - frame_length) / hop) + 1;
% Initialize the frame matrix
frame_matrix = zeros(num_frame, frame_length);
% Perform framing
for m = 0:num_frame - 1
    frame_matrix(m + 1, :) = s(m * hop + 1 : m * hop + frame_length);
end
% Set the output variables
f = frame_matrix;
t = (frame_length/2 : hop : frame_length/2 + (num_frame - 1)*hop) / fs;
end
```

تابع /'rectangular\_framing/' به منظور انجام فریم بندی مستطیلی روی یک سیگنال ورودی ایجاد شده است. این تابع با استفاده از طول فریم مشخص شده ('frame\_length') و فرکانس نمونهبرداری ('fs')، فریم بندی را بر روی سیگنال ورودی 's' انجام می‌دهد. این تابع یک ماتریس فریم ('f') حاوی بخش‌های فریم شده سیگنال و یک بردار زمان ('t') حاوی لحظات زمانی متناظر با هر فریم را برمی‌گرداند. رای فریم بندی با پنجره مستطیل، از ۵۰٪ همپوشانی استفاده می‌شود. بنابراین طول گام ('hop') برابر با نصف طول فریم ('frame\_length / 2') تعیین می‌شود. تعداد فریم‌ها ('num\_frame') با تقسیم کل طول سیگنال ورودی به طول گام و سپس کاهش عدد به طور کلی محاسبه می‌شود. ماتریس فریم ('frame\_matrix') به طول تعداد فریم‌ها و با طول فریم تعریف شده، با صفر مقداردهی اولیه می‌شود. فریم‌بندی با استفاده از حلقه انجام می‌شود. برای هر فریم، بخش متناظر از سیگنال ورودی را از طریق اندیس‌های مناسب در ماتریس فریم قرار می‌دهیم. مقدارهای خروجی 'f' و 't' به ترتیب برابر با ماتریس فریم و بردار زمان می‌باشند. بردار زمان با توجه به طول فریم، گام و تعداد فریم‌ها محاسبه می‌شود و به واحدهای زمانی نرمال شده (ثانیه) تبدیل می‌شود.

### **smooth2a.m**

```
``function matrix_out = smooth2a(matrix_in,window_rows,windows_cols)
% matrix_out = SMOOTH2A(matrix_in, window_rows, window_cols) performs a 2D
% smoothing operation on the input matrix 'matrix_in' using a rectangular
% averaging window of size 'window_rows' by 'window_cols'. The output is the
% smoothed matrix 'matrix_out'.
%
% Check the number of input arguments
```

```

if nargin < 2, error('Not enough input arguments!'), end
N(1) = window_rows;
% Set the default window_cols value if not provided
if nargin < 3, N(2) = N(1); else N(2) = windows_cols; end
% Check if window_rows and window_cols are scalars
if length(N(1)) ~= 1, error('window_rows must be a scalar!'), end
if length(N(2)) ~= 1, error('windows_cols must be a scalar!'), end
% Get the size of the input matrix
[row,col] = size(matrix_in);
% Create the sparse matrices for the averaging window
eL = spdiags(ones(row,2*N(1)+1),(-N(1):N(1)),row,row);
eR = spdiags(ones(col,2*N(2)+1),(-N(2):N(2)),col,col);
% Create the sparse matrices for the averaging window
is_nan = isnan(matrix_in);
matrix_in(is_nan) = 0;
% Compute the normalization factor
normalize = eL*(~is_nan)*eR;
normalize(is_nan) = NaN;
% Perform the smoothing operation
matrix_out = eL*matrix_in*eR;
matrix_out = matrix_out./normalize;

```

این بخش مربوط به تابع `smooth2a` است که یک عملیات `smoothing` دو بعدی روی ماتریس ورودی `matrix_in` با استفاده از یک پنجره میانگین گیری مستطیلی به اندازه `window_rows` در `window_cols` انجام می‌دهد. خروجی این تابع ماتریسی است که مقادیر آن `smooth` شده است و در متغیر `matrix_out` ذخیره می‌شود. تابع در ابتدا تعداد آرگومان‌های ورودی را بررسی می‌کند و در صورت کم بودن آرگومان‌ها، خطأ را نمایش می‌دهد. سپس، مقدار پیش‌فرض برای `window_cols` را تعیین می‌کند در صورتی که ارائه نشده باشد. سپس، بررسی می‌کند که آیا `window_rows` و `window_cols` عدددهای مقدارهای ثابت هستند و در صورت عدم اتفاق، خطأ را نمایش می‌دهد. سپس، اندازه ماتریس ورودی `matrix_in` با استفاده از تابع `size` مشخص می‌شود. سپس، ماتریس‌های `eL` و `eR` برای پنجره میانگین گیری با استفاده از تابع `spdiags` ایجاد می‌شوند. تابع بررسی می‌کند که آیا عناصر `matrix_in` شامل مقادیر `NaN` هستند و آن‌ها را با صفر جایگزین می‌کند. عامل نرمال‌سازی با محاسبه ضرب داخلی بین `eL`، یک ماتریس نشان‌دهنده تعداد

عناصر غیر-NaN، و eR محاسبه می‌شود. عملیات smoothing از طریق محاسبه ضرب داخلی بین eL و matrix\_in و سپس ضرب داخلی با eR انجام می‌شود. نتیجه تقسیم بر عامل نرم‌السازی می‌شود. تابع ماتریس smooth شده matrix\_out را برمی‌گرداند.

```
function frame = three_level_center_clipping(f)
%THREE_LEVEL_CENTER_CLIPPING Perform three-level center clipping on the input
frame.
% frame = THREE_LEVEL_CENTER_CLIPPING(f) applies three-level center clipping
% to the input frame 'f'. The output is the clipped frame 'frame'.
frame = zeros(1, length(f));
maximum = max(abs(f));
clipping_factor = 0.3;
for i = 1:length(frame)
if f(i) >= clipping_factor * maximum
frame(i) = 1;
elseif f(i) <= -(clipping_factor * maximum)
frame(i) = -1;
end
end
end
```

این کد تابع three\_level\_center\_clipping را تعریف می‌کند که عملیات clipping سه سطحی را روی فریم ورودی f انجام می‌دهد. خروجی این تابع، فریم کلیپ شده frame است. تابع یک آرایه به نام frame با همان طول فریم ورودی f را مقداردهی اولیه می‌کند. سپس با استفاده از توابع max و abs، بیشینه‌ی مقدار مطلق در فریم ورودی را پیدا می‌کند. یک عامل clipping با مقدار 0.3 تعریف می‌شود. سپس تابع برای هر عنصر از فریم ورودی تکرار می‌شود و عملیات center-clipping انجام می‌شود. اگر عنصر بزرگتر یا مساوی clipping\_factor ضرب در بیشینه مقدار باشد، آن را برابر 1 قرار می‌دهد. اگر عنصر کوچکتر یا مساوی ضرب منفی clipping\_factor در بیشینه مقدار باشد، آن را برابر -1 قرار می‌دهد. در غیر این صورت، مقدار عنصر بدون تغییر باقی می‌ماند. تابع نتیجه‌ی فریمی clipping روی آن اعمال شده frame را برمی‌گرداند.

### **voiced\_frame.m**

```
function label = voiced_frame(frames)
%VOICED_FRAME Determine the voiced frames based on short-time energy.
% label = VOICED_FRAME(frames) calculates the short-time energy for each frame
```

```
% in the input 'frames' and determines whether the frame is voiced or unvoiced
% based on a fixed threshold of 0.01. The output 'label' is a logical array
% indicating the voiced frames.

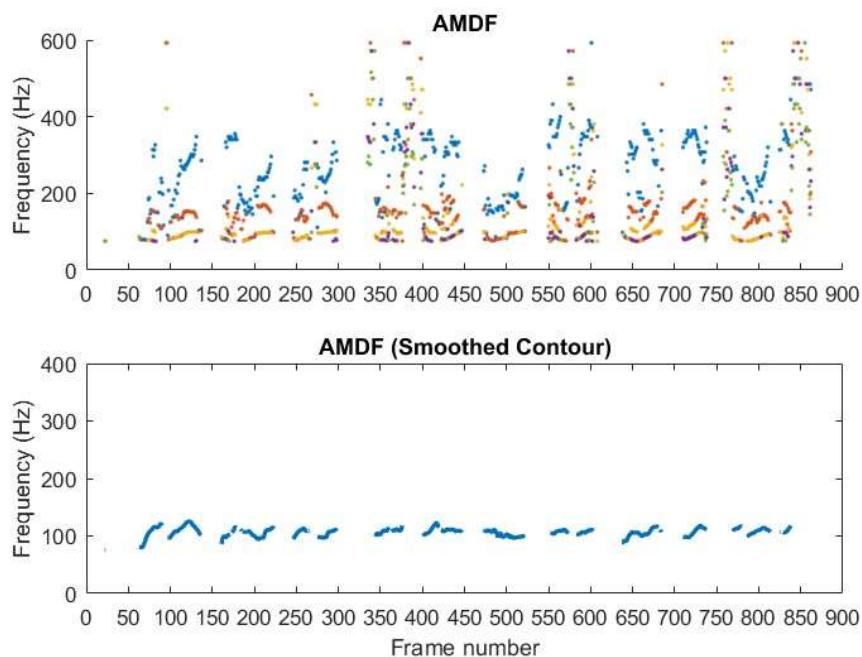
% Calculate the short-time energy for each frame
short_time_energy = sum(frames.^2, 2);

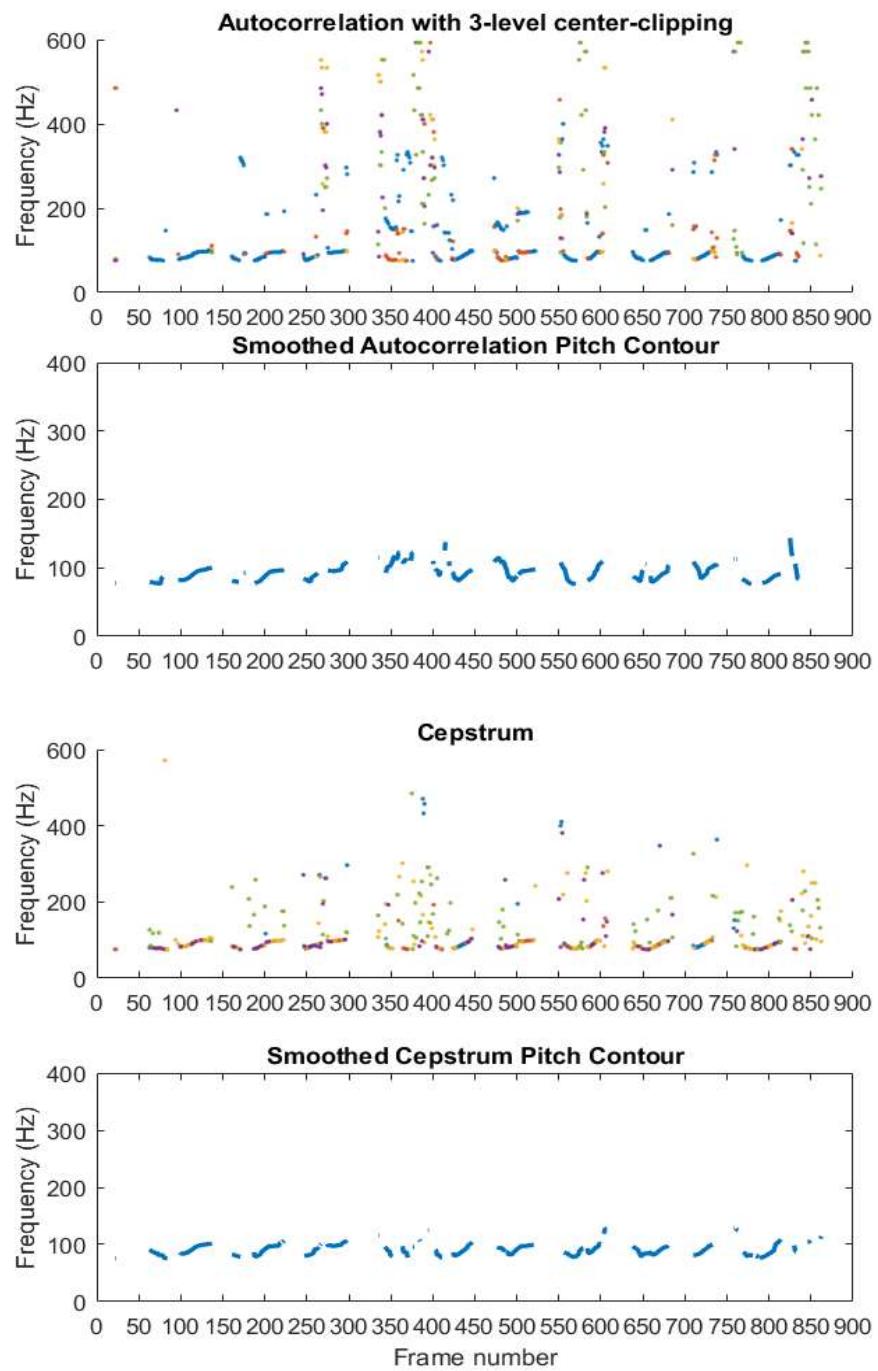
% Determine the voiced frames based on the threshold
voiced_frames = short_time_energy > 0.01;

% Set the label as the voiced frames
label = voiced_frames;
end
```

تابع ارائه شده با نام `voiced_frame` بر اساس انرژی، فریم‌های صوتی را مشخص می‌کند. ابتدا، تابع `short-time energy` را برای هر فریم ورودی محاسبه می‌کند. برای این منظور، مجذور هر عنصر فریم را محاسبه کرده و در طول ستون‌ها جمع می‌گیرد. سپس با استفاده از یک آستانه ثابت به ارزش 0.01، فریم‌های صوتی و بی‌صدا را تشخیص می‌دهد. اگر مقدار انرژی برای یک فریم بزرگتر از این آستانه باشد، آن فریم به عنوان فریم صوتی تعیین می‌شود و در غیر این صورت به عنوان فریم بی‌صدا شناخته می‌شود. خروجی این تابع یک آرایه منطقی به نام `label` است که نشان می‌دهد کدام فریم‌ها صوتی هستند. اگر یک عنصر `label` برابر با 1 باشد، فریم متناظر آن صوتی است و اگر برابر با 0 باشد، فریم بی‌صدا است.

**خروجی:**





```
>> main
Method: AMDF
Estimated Pitch: 107.8277 Hz
Method: Autocorrelation
Estimated Pitch: 92.5041 Hz
Method: Cepstrum
Estimated Pitch: 90.1191 Hz
```

روش‌های کپستروم، اتوکورلیشن و AMDF به عنوان روش‌های تخمین فرکانس گام در تشخیص فریم‌های صدا را می‌توان در نظر گرفته شده اند.(در خروجی ۳ figure subplot ۲ هستند که خروجی فرکانس پیج و روش‌ها اختصاص می‌یابد و هر یک شامل smooth شده آن نمایش می‌دهد). این روش‌ها بر اساس خواص سیگنال ورودی، میزان پیچیدگی محاسباتی و دقت و استحکام نتایج، مزایا و معایب متفاوتی دارند.

کپستروم یک روش در حوزه فرکانس است که با استفاده از تبدیل فوریه معکوس لگاریتم طیف یک سیگنال، به تخمین فرکانس گام آن می‌پردازد. این روش برای سیگنال‌هایی با اجزای هارمونیک مناسب است، اما ممکن است در مواجهه با سیگنال‌های نویزدار یا غیرثابت دچار مشکل شود. اتوکورلیشن یک روش در حوزه زمان است که شباهت یک سیگنال با خودش را به عنوان تابعی از تاخیر زمانی اندازه‌گیری می‌کند. این روش قادر است تناوب را در یک سیگنال تشخیص داده و فرکانس گام آن را با پیدا کردن اوج تابع اتوکورلیشن خود تخمین بزند. با این حال، ممکن است به دلیل وجود سایر هارمونیک‌ها یا نویز، تشخیص نادرستی نیز داشته باشد. همچنین فرمنت اول به علت قوی بودن و یا نزدیکی به فرکانس گام می‌تواند باعث اشتباه در تخمین شود.

AMDF نیز یک روش دیگر در حوزه زمان است که میانگین اختلاف مطلق بین یک سیگنال و نسخه شیفت‌یافته خود را به عنوان تابعی از تاخیر زمانی محاسبه می‌کند. این روش می‌تواند فرکانس گام را با یافتن حداقل تابع AMDF تخمین بزند. روش ساده‌تر و سریع‌تر از اتوکورلیشن است، اما ممکن است دقت کمتری داشته باشد و به نویز حساس‌تر باشد.

در هر روش، برخی مشکلات ممکن است وجود داشته باشد. به عنوان مثال، در روش کپستروم، به دلیل قدرت و یا نزدیکی فرکانس گام، تخمین اشتباہی ممکن است ارائه شود. همچنین، در سیگنال‌های گفتاری غیرپریودیک، خطاهای تشخیص گام ممکن است به دلیل اتصال بخش‌های واکدار و بیواک وجود داشته باشد. در روش AMDF نیز، علاوه بر دقت کمتر، حساسیت بیشتری نسبت به نویز وجود دارد.

فایل‌های مربوط به کد کامل برنامه به صورت یکجا، در لینک زیر قابل مشاهده است:

[https://drive.google.com/drive/folders/1ozWtYS2stdleKDgNi\\_l7qaqlNXJV3VMI?usp=sharing](https://drive.google.com/drive/folders/1ozWtYS2stdleKDgNi_l7qaqlNXJV3VMI?usp=sharing)