پروژه گفتار

پروژه درس مبانی پردازش زبان و گفتار دوره کارشناسی

لعیا فاخر

1-جمله انتخابی فارسی از کتاب رابینر:

"یافتن دوستان خوب سخت است"

جمله انگلیسی : "Good friends are hard to find"

منبع : فصل 3 کتاب رابینر (Fundamentals of Human Speech Production) صفحه 112

3.6. Segment the waveform in Figure P3.6 into regions of voiced speech (V), regions of unvoiced speech (U), and regions of silence (or background signal) (S). The waveform corresponds to the sentence "Good friends are hard to find."

ابتدا مراحل اولیه برای تخمین فرکانس گام را انجام میدهیم:

1-فریم بندی و پنجره گذاری سیگنال گفتار

2-مقدار DC فریم باید محاسبه و حذف شود تا آفست کل فریم صفر شود

3-تشخیص فریم سکوت (و نویز زمینه) از فریم حاوی گفتار با اسفاده از انرژی فریم های اول و آخر کل سیگنال

4-گفتار در دست تحلیل، مشخص میشود.(محاسبه انرژی هر فریم و مقایسه با حد آستانه(به شرطی که نسبت سیگنال به نویز گفتار کم نباشد.

5-تشخیص واکدار بودن یا نبودن فریم (استفاده از محاسبه نرخ عبور از صفر)

سپس برای **تخمین گام:**

روش AMDF :

با توجه به این خاصیت که تابع AMDF با نزدیک شدن به مقدار پریود سیگنال ، دارای مینیمم می شود از آن برای به دست آوردن فرکانس گام استفاده می کنیم.

$$AMDF(\eta) = \frac{1}{N}\sum_{n=0}^{N-1} |S(n)\text{-}S(n\text{-}\eta)| \qquad 0 \le \eta \le N-1$$

فاصله اولین دره برابر است با lpos:

Fpitch = FS/lpos

روش اتوکورولیشن:

نمودار تغییرات مقدار تابع اتوکورولیشن را رسم میکنیم، فاصله اولین قله یعنی lpos مقدار پریود گام خواهد بود.

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

Fpitch = FS/lpos

روش کپستروم:

ابتدا ضرایب کپستروم را محاسبه میکنیم

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

سپس از یک لیفتر زمان بالا استفاده می کنیم و فاصله اولین نقطه اوج قابل توجه یعنی lpos را پیدا کرده و فرکانس گام را مطابق فرمول زیر بدست می آوریم:

Fpitch = FS/lpos

2-توضیح کد ها:

تعیین مقادیر اولیه برای طول فریم و فاصله فریم ها و فرکانس نمونه برداری و تعداد کل فریم ها در فایل صوتی.

```
[waveform,FS] = audioread('C:\Users\Lenovo\Desktop\واجها\audio.wav','native');
waveform = double(waveform);
frame_length=40; %40ms
N = frame_length*FS/1000; %sample per frame = frame length
waveform_size =  length(waveform);
frame_overlap = 0.50;  % 50% of frame length
M = frame_overlap * N; % frame shift
FramesCount = (length(waveform)-N)/M + 1;
FramesCount = round(FramesCount)-1; %number of frames in audioclip
high_freq = 400;
low_freq = 75;
global candidate_number; %number of candidates in each frame
candidate_number = 6;
```

توابعی برای محاسبه DC و ZCR و energy یک فریم مینویسیم.

```
function dc = DC(frame_i)
    dc = sum(frame_i)/length(frame_i);
end

function zcr = ZCR(frame_i)
    FrameZCR = 0;
    for n=2:length(frame_i)      %claculate ZCR
        FrameZCR=FrameZCR + abs(sign(frame_i(n))-sign(frame_i(n-1)));
    end
    zcr = FrameZCR/(2*length(frame_i));
end
function en = energy(frame_i)
    en = sum(frame_i.*frame_i)/length(frame_i);
end
```

مراحل زیر را برای تخمین فرکانس گام در هر 3 روش amdf و autocorrelation و cepstrum انجام میدهیم:

1-فریم بندی و پنجره گذاری سیگنال گفتار

در یک حلقه به ازای هر فریم :

2-مقدار DC فریم باید محاسبه و حذف شود تا آفست کل فریم صفر شود

3-تشخیص فریم سکوت (و نویز زمینه) از فریم حاوی گفتار با اسفاده از انرژی فریم های اول و آخر کل سیگنال

4-گفتار در دست تحلیل، مشخص میشود.(محاسبه انرژی هر فریم و مقایسه با حد آستانه(به شرطی که نسبت سیگنال به نویز گفتار کم نباشد.

5-تشخیص واکدار بودن یا نبودن فریم (استفاده از محاسبه نرخ عبور از صفر)

```
global candidate_number;
threshold_for_silence = 0.001;
threshold_for_voiced = 2000;
window = hamming(N);
amdfs = [];
for i=1:FramesCount
    frame_i = waveform((i-1)*M+1:(i-1)*M + N);  %splitting ith frame
    frame_i = frame_i .* window; %windowing
    dc = DC(frame_i);
    frame_i = frame_i - dc; %removing DC from each frame
    e=energy(frame_i);
    z=ZCR(frame_i);
    pf = amdf(frame_i);%check is frame is not noise or unvoiced
```

الف) روش amdf:

ابتدا یک تابع برای محاسبه مقدار amdf یک فریم می نویسیم:

```
%method to calculate amdf of a frame
function frame_amdf = amdf(frame_i)
    frame_amdf = zeros(1,length(frame_i));
    for eta=0:length(frame_i)-1
        sum=0;
        for n=eta:length(frame_i)-1
            sum =  sum + abs((frame_i(n+1)-frame_i(n+1-eta)));
        end
        frame_amdf(eta+1) = sum;
    end
end
```

سپس توسط تابع نوشته شده، برای هر فریم مقدار amdf را محاسبه می کنیم و سپس دره ها(مینیمم قابل توجه) را برای هر فریم بدست می آوریم تا در نهایت فرکانس گام با رابطه مربوطه محاسبه شود.به ازای هر فریم تعدادی کاندید(6 عدد) را برای محاسبه فرکانس گام در نظر میگیریم که در متغیر candidate_number ذخیره شده است.

Fpitch = FS/Ipos

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

فاصله اولین دره برابر است با Ipos.

```
%method to calculate amdf of a waveform
%6 pitch candidate for each frame
function pitchFreq = amdf_pitch(waveform,FramesCount,N,M,frame_length)
    global candidate_number;
    threshold_for_silence = 0.001;
    threshold_for_voiced = 2000;
    window = hamming(N);
    amdfs = [];
    for i=1:FramesCount
        frame_i = waveform((i-1)*M+1:(i-1)*M + N);  %splitting ith frame
        frame_i = frame_i .* window; %windowing
        dc = DC(frame_i);
        frame_i = frame_i - dc; %removing DC from each frame
        e=energy(frame_i);
        z=ZCR(frame_i);
        pf = amdf(frame_i);%check is frame is not noise or unvoiced
         if e > threshold_for_silence || z < threshold_for_voiced
             amdfs = [amdfs ; pf ];
         else
             amdfs = [amdfs ;  zeros(1,N) ];
         end
    end
    for i=1:FramesCount
        p = smooth(smooth(smooth(amdfs(i,:),10),10),50) ;
        p = -p;
        [peaks,Ipos] = findpeaks(p);
        Ipos_new = [];
        minimum = min(candidate_number,length(Ipos));
        for j=1:minimum
            max = min(peaks);
            z = find(peaks==max);
            if length(z)==1
                Ipos_new = [ Ipos_new , Ipos(z) ];
                peaks = peaks(peaks~=max);
                Ipos = Ipos(Ipos~=Ipos(z));
            elseif length(z)==2
                n = Ipos(z);
                Ipos_new = [ Ipos_new , n(1) ,n(2)  ];
                peaks = peaks(peaks~=max);
                Ipos = Ipos(Ipos~=n(1));
                Ipos = Ipos(Ipos~=n(2));
            end
        end
        Ipos = Ipos_new;
        for j=1:length(Ipos)
            pitchFreq(i,j) = 1/((Ipos(j))*frame_length/(N*1000));
        end
        if length(Ipos)<=1
           for j=1:candidate_number
               pitchFreq(i,j)=0;
           end
        end
    end
end
```

برای رسم نمودار فرکانس گام (در هر 3 روش) نیاز است که مقادیر فرکانس گام را ابتدا فیلتر(فرکانس های کمتر از 75 یا بیشتر از 400 هرتز را حذف کنیم.(این مقادیر در متغیر های high_freq و low_freq ذخیره شده اند.))

در نهایت نمودار فرکانس گام را با کمک تابع plot رسم می کنیم(یکبار در حالتی که مقادیر فرکانس گام فیلتر و smooth شده باشد (توسط تابع Smooth)و یکبار هم وقتی که فرکانس گام فقط فیلتر شده باشد و smooth نشده باشد(یعنی مقادیر فرکانس گام کاندیدا های هر فریم مشخص باشند.))

```
pitch_frequency = amdf_pitch(waveform,FramesCount,N,M,frame_length);
%pitch frequencies and smoothing  - amdf
% remove frequencies that are more than 400hz or less than 75
pitch_frequency2 = pitch_frequency;
X = NaN(1,FramesCount);
Y = NaN(1,FramesCount);
for i=1:FramesCount
   for j=1:candidate_number
      if  pitch_frequency2(i,j) < low_freq ||  pitch_frequency2(i,j) >high_freq
          pitch_frequency2(i,j)  = nan;
      end
   end
end
pitch_frequency2 = Smooth(pitch_frequency2,20,10);
for i=1:FramesCount
    X(i) =  i ;
end
for i=1:FramesCount
    for j=1:candidate_number
        if pitch_frequency2(i,j) > low_freq && pitch_frequency2(i,j) <high_freq
            Y(i) = pitch_frequency2(i,j);
            break;
        end
    end
end
%filtering pitch frequencies without smoothing to show pitch candidates - amdf
%(6 candidate for each frame)
candidate_X = NaN(1,candidate_number*length(pitch_frequency));
candidate_Y = NaN(1,candidate_number*length(pitch_frequency));
for i=1:length(pitch_frequency)
   for j=1:candidate_number
       if pitch_frequency(i,j)>low_freq && pitch_frequency(i,j)<high_freq
           candidate_X((i-1)*candidate_number+j) = i;
           candidate_Y((i-1)*candidate_number+j) = pitch_frequency(i,j);
       end

   end
end
%plot pitch frequencies - amdf
subplot(4,1,2)
plot(candidate_X,candidate_Y,'.','MarkerEdgeColor','r')
ylabel("Pitch Frequency (Hz)"+newline+"candidates",'Fontsize',8);
xlim([0 FramesCount])
ylim([50 400])
nanmean(Y)
%plot pitch frequencies (smooth) - amdf
subplot(4,1,1)
```

```
plot(X,Y,'m','LineWidth',1.5)
ylabel("Pitch Frequency (Hz)"+newline+"smooth contour",'Fontsize',8);
xlim([0 FramesCount])
ylim([50 300])
title("AMDF");
```

تابع Smooth:

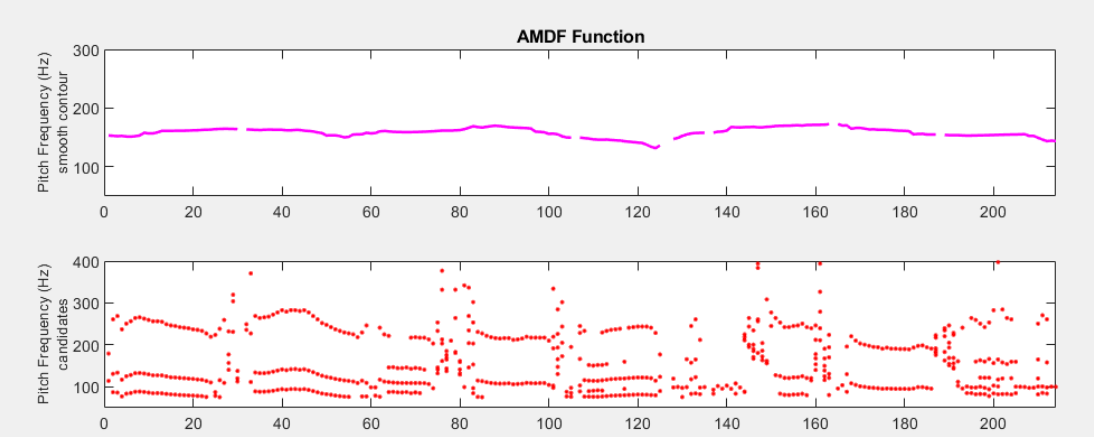
این تابع مقادیر فرکانس گام کاندیداها را در متغیر pitchFreq در یافت کرده و با استفاده از فیلتر میانگین روی یک مستطیل به اندازه

(2*تعداد ردیف + 1) - در - (2 * تعداد ستون + 1)

صاف (smooth) می کند.از آنجایی که در هر فریم، 6 کاندید را برای محاسبه فرکانس گام در نظر گرفتیم، این تابع به نمایش بهتر نمودار فرکانس گام کمک میکند.

```
% This function smooths the data in pitchFreq using a mean filter over a
% rectangle of size (2*rows+1)-by-(2*columns+1).
function outputData =  Smooth(pitchFreq,rows,columns)
    frame_length(1) = rows;
    if nargin < 3
        frame_length(2) = frame_length(1);
    else
        frame_length(2) = columns;
    end
    [row,col] = size(pitchFreq);
    % Building matrices that will compute running sums.  The left-matrix, RS,
    % smooths along the rows.  The right-matrix, CS, smooths along the
    % columns.
    RS = spdiags(ones(row,2*frame_length(1)+1),(-frame_length(1):frame_length(1)),row,row);
    CS = spdiags(ones(col,2*frame_length(2)+1),(-frame_length(2):frame_length(2)),col,col);
   % Setting all "NaN" elements of "pitchFreq" to zero so that these will not
   % affect the summation.
    A = isnan(pitchFreq);
    pitchFreq(A) = 0;
    nrmlize = RS*(~A)*CS;
    nrmlize(A) = NaN;
    outputData = RS*pitchFreq*CS;
    outputData = outputData./nrmlize;
end
```

.خروجی برنامه در روش amdf:



ب) روش اتوکورولیشن:

ابتدا یک تابع برای محاسبه اتوکورلیشن یک فریم و یک تابع three_level_center_clipping(برای بهبود اتوکورلیشن) می نویسیم.

```
%method to calculate autocorelation of a frame
function frame_autocorelation = autocorelation(frame_i)
    frame_autocorelation = zeros(1,length(frame_i));
    for eta=0:length(frame_i)-1
        sum=0;
        for n=eta:length(frame_i)-1
            sum =  sum + frame_i(n+1)*frame_i(n+1-eta);
        end
        frame_autocorelation(eta+1) = sum;
    end
end
%method to apply 3-level-center-clipping on a frame
function frame = three_level_center_clipping(frame_i)
    frame = zeros(1, length(frame_i));
    maximum = max(abs(frame_i));
    c = 0.3;
    for i = 1:length(frame)
        if frame_i(i) >= c * maximum
            frame(i) = 1;
        elseif frame_i(i) > -(c * maximum) && frame_i(i) < c * maximum
            frame(i) = 0;
        elseif frame_i(i) <= -(c * maximum)
            frame(i) = -1;
        end
    end
end
```

به ازای هر فریم اتوکورلیشن را محاسبه کرده و از 3-level-center-clipping استفاده میکنیم تا نقاط اوج نمودار بهتر نشان داده شوند. سپس 6 کاندید را انتخاب میکنیم و فرکانس گام آن ها را بدست می آوریم.
مطابق رابطه:

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

Fpitch = FS/Ipos

فاصله اولین قله یعنی Ipos مقدار پریود گام خواهد بود.

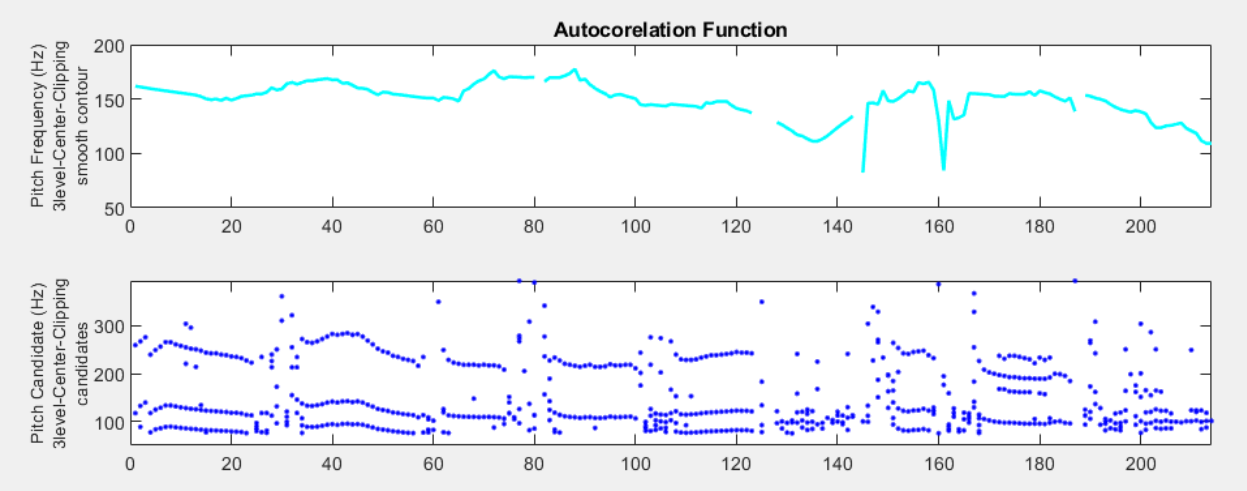
برای رسم نمودار فرکانس گام (در هر 3 روش) نیاز است که مقادیر فرکانس گام را ابتدا فیلتر(فرکانس های کمتر از 75 یا بیشتر از 400 هرتز را حذف کنیم.(این مقادیر در متغیر های high_freq و low_freq ذخیره شده اند.))

```
%method to calculate autocorelation of a waveform
%6 pitch candidate for each frame
function pitchFreq_ac =
autocorelation_pitch(waveform,FramesCount,N,M,frame_length)
    global candidate_number;
    window = hamming(N);
    threshold_for_voiced = 2000;
    threshold_for_silence = 0.001;
    autocorelations = [];
    for i=1:FramesCount
        frame_i = waveform((i-1)*M+1:(i-1)*M + N);%splitting ith frame
        frame_i = frame_i .* window; %windowing
        dc = DC(frame_i);
        frame_i = frame_i - dc; %removing DC from each frame
        Energy=energy(frame_i);
        zcr=ZCR(frame_i);
        frame = three_level_center_clipping(frame_i);
        auto = autocorelation(frame);
        if Energy < threshold_for_silence  || zcr > threshold_for_voiced
            autocorelations = [autocorelations ;  zeros(1,N) ];
        else
            autocorelations = [autocorelations ; auto ];
        end
    end
    for i=1:FramesCount
         p = smooth(smooth(smooth(autocorelations(i,:),10),10)) ;
        [peaks,Ipos] = findpeaks(p);
        Ipos_final = [];
        minimum = min(candidate_number,length(Ipos));
        for j=1:minimum
            m = max(peaks);
```

```
            n = find(peaks==m);
            if length(n)==1
                Ipos_final = [ Ipos_final , Ipos(n) ];
                peaks = peaks(peaks~=m);
                Ipos = Ipos(Ipos~=Ipos(n));
            elseif length(n)==2
                et = Ipos(n);
                Ipos_final = [ Ipos_final , et(1) ,et(2)  ];
                peaks = peaks(peaks~=m);
                Ipos = Ipos(Ipos~=et(1));
                Ipos = Ipos(Ipos~=et(2));
            end
        end
        Ipos = Ipos_final;
        for j=1:length(Ipos)
            pitchFreq_ac(i,j) = 1/((Ipos(j))*frame_length/(N*1000));
        end
        if length(Ipos)<=1
           for j=1:candidate_number
               pitchFreq_ac(i,j)=0;
           end
        end
    end
end
```

در نهایت نمودار فرکانس گام را با کمک تابع plot رسم می کنیم(یکبار در حالتی که مقادیر فرکانس گام فیلتر و smooth شده باشد (توسط تابع Smooth)و یکبار هم وقتی که فرکانس گام فقط فیلتر شده باشد و smooth نشده باشد(یعنی مقادیر فرکانس گام کاندیدا های هر فریم مشخص باشند.))

.خروجی برنامه در روش اتوکورولیشن:



پ) روش کپستروم:

ابتدا یک تابع برای کپستروم یک فریم می نویسیم.

```
%method to calculate cepstrum of a frame
function frame_cep = cepstrum(frame_i)
    frame_cep = real(ifft(log(abs(fft(frame_i)))));
end
```

سپس یک تابع مینویسیم که لیفتر زمان بالا را اعمال کند:

```
%method to apply high time liftering on a frame
function ceps = highTimeLifter(frame)
    NN = 20;
    ceps = zeros(1, length(frame));
    for i=1:length(ceps)
        if i >= NN
            ceps(i) = frame(i);
        else
            ceps(i) = 0;
        end
    end
end
```

به ازای هر فریم کپستروم را محاسبه کرده و از لیفتر زمان بالا استفاده میکنیم تا سیگنال تحریک راداشته باشیم. سپس 6 کاندید را انتخاب میکنیم و با peak picking مقادیر فرکانس گام را بدست می آوریم. مطابق رابطه:

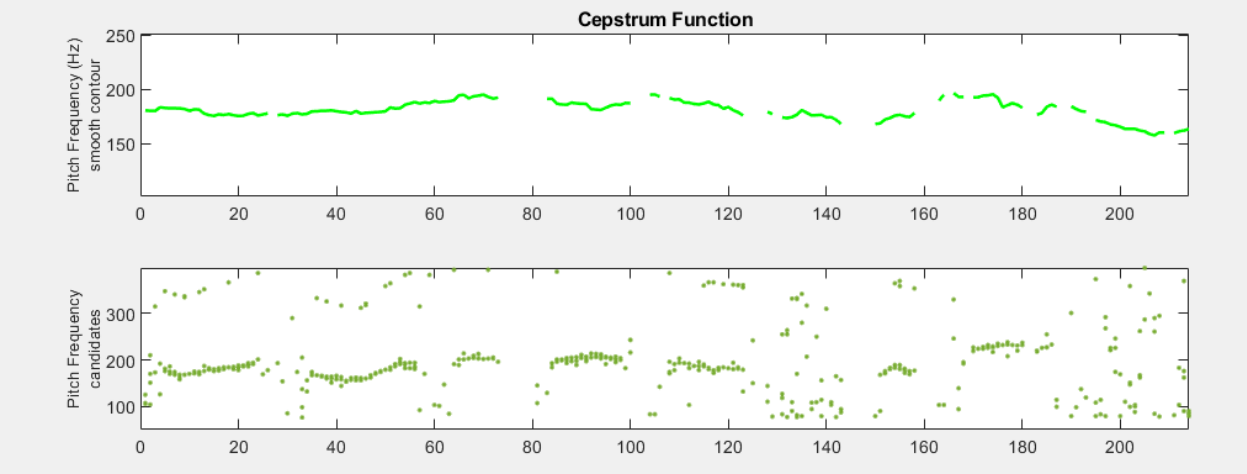
Fpitch = FS/lpos

lpos فاصله اولین قله قابل توجه است.

```
%method to calculate cepstrum of a waveform
%6 pitch candidate for each frame
function pitchFreq_cep = cepstrum_pitch(waveform,N,M,FramesCount)
    global candidate_number;
    pitchFreq_cep = zeros(FramesCount,candidate_number);
    threshold_for_voiced = 2000;
    threshold_for_silence = 0.001;
    window = hamming(N);
```

```
    for i=1:FramesCount
        frame_i = waveform((i-1)*M+1:(i-1)*M + N);
        frame_i = frame_i .* window; %windowing
        e = energy(frame_i);
        zcr=ZCR(frame_i);
        if e < threshold_for_silence || zcr > threshold_for_voiced
            continue
        else
            frame=cepstrum(frame_i);
            ceps = highTimeLifter(frame);
            n_ceps=length(ceps);
            ceps=ceps(1:n_ceps/2);%because of symmetry in cepstrum
            [ peaks , Ipos ] = findpeaks(ceps);
            for j=1:candidate_number
                [ maximum , peak ] = max(peaks);
                peaks = peaks(peaks~=maximum);
                if  length(peak)>1
                    pitchFreq_cep(i,j) = Ipos(peak(1));
                    pitchFreq_cep(i,j+1) = Ipos(peak(2));
                    Ipos = Ipos(Ipos~=Ipos(peak(1)));
                    Ipos = Ipos(Ipos~=Ipos(peak(1)));
                    j = j+1;
                elseif length(peak)==1
                    pitchFreq_cep(i,j) = Ipos(peak);
                    Ipos = Ipos(Ipos~=Ipos(peak));
                end
            end
        end
    end
end
```

مانند روش های قبلی نمودار فرکانس گام را یکبار به ازای مقادیر کاندید و یکبار به ازای مقادیر smooth شده بدست می آوریم.خروجی برنامه در روش cepstrum:



مقایسه روش ها و خروجی های آن ها:

روش‌های مختلف بسته به نوع و کیفیت سیگنال ورودی، پیچیدگی محاسباتی و دقت و استحکام نتایج ممکن است مزایا و معایب متفاوتی داشته باشند .

کپستروم:

یک روش حوزه فرکانس است که از تبدیل فوریه معکوس لگاریتم طیف یک سیگنال برای تخمین فرکانس گام آن استفاده می کند .می تواند برای سیگنال هایی با اجزای هارمونیک موثر باشد، اما ممکن است برای سیگنال های نویزدار یا غیر ثابت مشکل داشته باشد.

اتوکورلیشن:

یک روش حوزه زمانی است که شباهت یک سیگنال با خودش را به عنوان تابعی از تاخیر زمانی اندازه گیری می کند .می تواند تناوب را در یک سیگنال تشخیص دهد و فرکانس گام آن را با پیدا کردن اوج تابع همبستگی خود تخمین بزند .با این حال، ممکن است به دلیل ساب هارمونیک یا نویز، تشخیص نادرست نیز داشته باشد.

مشکلات روش:

فرمنت اول به علت قوی بودن و یا نزدیکی به فرکانس گام می تواند باعث اشتباه در تخمین شود .

سیگنال گفتار عمالً غیر پریودیک است

در محل اتصال بخش های واکدار و بیواک احتمال خطای تشخیص گام وجود دارد.

amdf : روش دیگری در حوزه زمانی است که میانگین اختلاف مطلق بین یک سیگنال و نسخه شیفت یافته خود را به عنوان تابعی از تاخیر زمانی محاسبه می کند .همچنین می تواند فرکانس گام را با یافتن حداقل تابع AMDFتخمین بزند .این روش ساده‌تر و سریع‌تر از اتوکورلیشن است، اما ممکن است دقت کمتری داشته باشد و نسبت به نویز حساس‌تر باشد.

تمام کد های برنامه به صورت یکجا:

```
[waveform,FS] =
audioread('C:\Users\Lenovo\Desktop\?????\audio.wav','native');
```

```
waveform = double(waveform);
frame_length=40; %40ms
N = frame_length*FS/1000; %sample per frame = frame length
waveform_size =  length(waveform);
frame_overlap = 0.50;  % 50% of frame length
M = frame_overlap * N; % frame shift
FramesCount = (length(waveform)-N)/M + 1;
FramesCount = round(FramesCount)-1; %number of frames in audioclip
high_freq = 400;
low_freq = 75;
global candidate_number; %number of candidates in each frame
candidate_number = 6;

pitch_frequency = amdf_pitch(waveform,FramesCount,N,M,frame_length);
%pitch frequencies and smoothing  - amdf
% remove frequencies that are more than 400hz or less than 75
pitch_frequency2 = pitch_frequency;
X = NaN(1,FramesCount);
Y = NaN(1,FramesCount);
for i=1:FramesCount
   for j=1:candidate_number
      if  pitch_frequency2(i,j) < low_freq ||  pitch_frequency2(i,j)
>high_freq
          pitch_frequency2(i,j)  = nan;
      end
   end
end
pitch_frequency2 = Smooth(pitch_frequency2,20,10);
for i=1:FramesCount
    X(i) =  i ;
end
for i=1:FramesCount
    for j=1:candidate_number
        if pitch_frequency2(i,j) > low_freq && pitch_frequency2(i,j)
<high_freq
            Y(i) = pitch_frequency2(i,j);
            break;
        end
    end
end
%filtering pitch frequencies without smoothing to show pitch
candidates - amdf
%(6 candidate for each frame)
candidate_X = NaN(1,candidate_number*length(pitch_frequency));
candidate_Y = NaN(1,candidate_number*length(pitch_frequency));
for i=1:length(pitch_frequency)
   for j=1:candidate_number
       if pitch_frequency(i,j)>low_freq &&
pitch_frequency(i,j)<high_freq
           candidate_X((i-1)*candidate_number+j) = i;
           candidate_Y((i-1)*candidate_number+j) =
pitch_frequency(i,j);
```

```
       end

   end
end
%plot pitch frequencies - amdf
subplot(4,1,2)
plot(candidate_X,candidate_Y,'.','MarkerEdgeColor','r')
ylabel("Pitch Frequency (Hz)"+newline+"candidates",'Fontsize',8);
xlim([0 FramesCount])
ylim([50 400])
nanmean(Y)
%plot pitch frequencies (smooth) - amdf
subplot(4,1,1)
plot(X,Y,'m','LineWidth',1.5)
ylabel("Pitch Frequency (Hz)"+newline+"smooth contour",'Fontsize',8);
xlim([0 FramesCount])
ylim([50 300])
title("AMDF Function");

%pitch frequencies and smoothing  - autocorelation
% remove frequencies that are more than 400hz or less than 75
pitch_frequency_ac =
autocorelation_pitch(waveform,FramesCount,N,M,frame_length);
pitch_frequency_ac2 = pitch_frequency_ac;
X2 = NaN(1,FramesCount);
Y2 = NaN(1,FramesCount);
for i=1:FramesCount
   for j=1:candidate_number
      if  pitch_frequency_ac2(i,j) < low_freq ||
pitch_frequency_ac2(i,j) >high_freq
          pitch_frequency_ac2(i,j)  = nan;
      end
   end
end
pitch_frequency_ac2 = Smooth(pitch_frequency_ac2,11,2);
for i=1:FramesCount
    X2(i) =  i ;
end
for i=1:FramesCount
    for j=1:candidate_number
        if pitch_frequency_ac2(i,j) > low_freq &&
pitch_frequency_ac2(i,j) <high_freq
            Y2(i) = pitch_frequency_ac2(i,j);
            break;
        end
    end
end
%filtering pitch frequencies without smoothing to show pitch
candidates - autocorelation
%(6 candidate for each frame)
candidate_X2 = NaN(1,candidate_number*length(pitch_frequency_ac));
candidate_Y2 = NaN(1,candidate_number*length(pitch_frequency_ac));
```

```
for i=1:length(pitch_frequency_ac)
   for j=1:6
       if pitch_frequency_ac(i,j)>low_freq &&
pitch_frequency_ac(i,j)<high_freq
           candidate_X2((i-1)*candidate_number+j) = i;
           candidate_Y2((i-1)*candidate_number+j) =
pitch_frequency_ac(i,j);
       end
   end
end

%plot pitch frequencies - autocorelation
subplot(4,1,4)
plot(candidate_X2,candidate_Y2,'.','MarkerEdgeColor','b')
ylabel("Pitch Candidate (Hz)"+newline+"3level-Center-
Clipping"+newline+"candidates",'Fontsize',8);
xlim([0 FramesCount])
ylim([50 200])
%plot pitch frequencies (smooth) - autocorelation
subplot(4,1,3)
plot(X2,Y2,'c','LineWidth',1.5)
ylabel("Pitch Frequency (Hz)"+newline+"3level-Center-
Clipping"+newline+"smooth contour",'Fontsize',8);
xlim([0 FramesCount])
ylim([50 200])
title("Autocorelation Function");

pitch_frequency_cep = cepstrum_pitch(waveform,N,M,FramesCount);
pitch_frequency_cep2 = pitch_frequency_cep;
X3 = NaN(1,FramesCount);
Y3 = NaN(1,FramesCount);
for i=1:FramesCount
   for j=1:candidate_number
      if  pitch_frequency_cep2(i,j) < low_freq ||
pitch_frequency_cep2(i,j) >high_freq
          pitch_frequency_cep2(i,j)  = nan;
      end
   end
end
pitch_frequency_cep2 = Smooth(pitch_frequency_cep2-15,20,20);
for i=1:FramesCount
    X3(i) =  i ;
end
for i=1:FramesCount
    for j=1:candidate_number
        if pitch_frequency_cep2(i,j) > low_freq &&
pitch_frequency_cep2(i,j) <high_freq
            Y3(i) = pitch_frequency_cep2(i,j);
            break;
        end
    end
end
```

```
%filtering pitch frequencies without smoothing to show pitch
candidates - cepstrum
%(6 candidate for each frame)
candidate_X3 = NaN(1,candidate_number*length(pitch_frequency_cep));
candidate_Y3 = NaN(1,candidate_number*length(pitch_frequency_cep));
for i=1:length(pitch_frequency_cep)
   for j=1:candidate_number
       if pitch_frequency_cep(i,j)>low_freq &&
pitch_frequency_cep(i,j)<high_freq
           candidate_X3((i-1)*candidate_number+j) = i;
           candidate_Y3((i-1)*candidate_number+j) =
pitch_frequency_cep(i,j);
       end
   end
end

%plot pitch frequencies - cepstrum
subplot(4,1,4)
plot(candidate_X3,candidate_Y3,'.','MarkerEdgeColor',"#77AC30")
ylabel("Pitch Frequency"+newline+"candidates",'Fontsize',8);
xlim([0 FramesCount])
ylim([50 200])
%plot pitch frequencies (smooth) - cepstrum
subplot(4,1,3)
plot(X3,Y3,'g','LineWidth',1.5)
ylabel("Pitch Frequency (Hz)"+newline+"smooth contour",'Fontsize',8);
xlim([0 FramesCount])
ylim([50 200])
title("Cepstrum Function");

%method to calculate amdf of a waveform
%6 pitch candidate for each frame
function pitchFreq = amdf_pitch(waveform,FramesCount,N,M,frame_length)
    global candidate_number;
    threshold_for_silence = 0.001;
    threshold_for_voiced = 2000;
    window = hamming(N);
    amdfs = [];
    for i=1:FramesCount
        frame_i = waveform((i-1)*M+1:(i-1)*M + N);  %splitting ith
frame
        frame_i = frame_i .* window; %windowing
        dc = DC(frame_i);
        frame_i = frame_i - dc; %removing DC from each frame
        e=energy(frame_i);
        z=ZCR(frame_i);
        pf = amdf(frame_i);%check is frame is not noise or unvoiced
         if e > threshold_for_silence || z < threshold_for_voiced
             amdfs = [amdfs ; pf ];
         else
             amdfs = [amdfs ;  zeros(1,N) ];
         end
```

```
    end
    for i=1:FramesCount
        p = smooth(smooth(smooth(amdfs(i,:),10),10),50) ;
        p = -p;
        [peaks,Ipos] = findpeaks(p);
        Ipos_new = [];
        minimum = min(candidate_number,length(Ipos));
        for j=1:minimum
            max = min(peaks);
            z = find(peaks==max);
            if length(z)==1
                Ipos_new = [ Ipos_new , Ipos(z) ];
                peaks = peaks(peaks~=max);
                Ipos = Ipos(Ipos~=Ipos(z));
            elseif length(z)==2
                n = Ipos(z);
                Ipos_new = [ Ipos_new , n(1) ,n(2)  ];
                peaks = peaks(peaks~=max);
                Ipos = Ipos(Ipos~=n(1));
                Ipos = Ipos(Ipos~=n(2));
            end
        end
        Ipos = Ipos_new;
        for j=1:length(Ipos)
            pitchFreq(i,j) = 1/((Ipos(j))*frame_length/(N*1000));
        end
        if length(Ipos)<=1
           for j=1:candidate_number
               pitchFreq(i,j)=0;
           end
        end
    end
end

%method to calculate autocorelation of a waveform
%6 pitch candidate for each frame
function pitchFreq_ac =
autocorelation_pitch(waveform,FramesCount,N,M,frame_length)
    global candidate_number;
    window = hamming(N);
    threshold_for_voiced = 2000;
    threshold_for_silence = 0.001;
    autocorelations = [];
    for i=1:FramesCount
        frame_i = waveform((i-1)*M+1:(i-1)*M + N);%splitting ith frame
        frame_i = frame_i .* window; %windowing
        dc = DC(frame_i);
        frame_i = frame_i - dc; %removing DC from each frame
        Energy=energy(frame_i);
        zcr=ZCR(frame_i);
        frame = three_level_center_clipping(frame_i);
        auto = autocorelation(frame);
```

```
        if Energy < threshold_for_silence  || zcr >
threshold_for_voiced
            autocorelations = [autocorelations ;  zeros(1,N) ];
        else
            autocorelations = [autocorelations ; auto ];
        end
    end
    for i=1:FramesCount
         p = smooth(smooth(smooth(autocorelations(i,:),10),10)) ;
        [peaks,Ipos] = findpeaks(p);
        Ipos_final = [];
        minimum = min(candidate_number,length(Ipos));
        for j=1:minimum
            m = max(peaks);
            n = find(peaks==m);
            if length(n)==1
                Ipos_final = [ Ipos_final , Ipos(n) ];
                peaks = peaks(peaks~=m);
                Ipos = Ipos(Ipos~=Ipos(n));
            elseif length(n)==2
                et = Ipos(n);
                Ipos_final = [ Ipos_final , et(1) ,et(2)  ];
                peaks = peaks(peaks~=m);
                Ipos = Ipos(Ipos~=et(1));
                Ipos = Ipos(Ipos~=et(2));
            end
        end
        Ipos = Ipos_final;
        for j=1:length(Ipos)
            pitchFreq_ac(i,j) = 1/((Ipos(j))*frame_length/(N*1000));
        end
        if length(Ipos)<=1
           for j=1:candidate_number
               pitchFreq_ac(i,j)=0;
           end
        end
    end
end

%method to calculate cepstrum of a waveform
%6 pitch candidate for each frame
function pitchFreq_cep = cepstrum_pitch(waveform,N,M,FramesCount)
    global candidate_number;
    pitchFreq_cep = zeros(FramesCount,candidate_number);
    threshold_for_voiced = 2000;
    threshold_for_silence = 0.001;
    window = hamming(N);
    for i=1:FramesCount
        frame_i = waveform((i-1)*M+1:(i-1)*M + N);
        frame_i = frame_i .* window; %windowing
        e = energy(frame_i);
        zcr=ZCR(frame_i);
```

```
        if e < threshold_for_silence || zcr > threshold_for_voiced
            continue
        else
            frame=cepstrum(frame_i);
            ceps = highTimeLifter(frame);
            n_ceps=length(ceps);
            ceps=ceps(1:n_ceps/2);%because of symmetry in cepstrum
            [ peaks , Ipos ] = findpeaks(ceps);
            for j=1:candidate_number
                [ maximum , peak ] = max(peaks);
                peaks = peaks(peaks~=maximum);
                if  length(peak)>1
                    pitchFreq_cep(i,j) = Ipos(peak(1));
                    pitchFreq_cep(i,j+1) = Ipos(peak(2));
                    Ipos = Ipos(Ipos~=Ipos(peak(1)));
                    Ipos = Ipos(Ipos~=Ipos(peak(1)));
                    j = j+1;
                elseif length(peak)==1
                    pitchFreq_cep(i,j) = Ipos(peak);
                    Ipos = Ipos(Ipos~=Ipos(peak));
                end
            end
        end
    end
end

function dc = DC(frame_i)
   dc = sum(frame_i)/length(frame_i);
end

function zcr = ZCR(frame_i)
    FrameZCR = 0;
    for n=2:length(frame_i)      %claculate ZCR
        FrameZCR=FrameZCR + abs(sign(frame_i(n))-sign(frame_i(n-1)));
    end
    zcr = FrameZCR/(2*length(frame_i));
end
function en = energy(frame_i)
    en = sum(frame_i.*frame_i)/length(frame_i);
end

%method to calculate amdf of a frame
function frame_amdf = amdf(frame_i)
    frame_amdf = zeros(1,length(frame_i));
    for eta=0:length(frame_i)-1
        sum=0;
        for n=eta:length(frame_i)-1
            sum =  sum + abs((frame_i(n+1)-frame_i(n+1-eta)));
        end
        frame_amdf(eta+1) = sum;
    end
end
```

```
%method to calculate autocorelation of a frame
function frame_autocorelation = autocorelation(frame_i)
    frame_autocorelation = zeros(1,length(frame_i));
    for eta=0:length(frame_i)-1
        sum=0;
        for n=eta:length(frame_i)-1
            sum =  sum + frame_i(n+1)*frame_i(n+1-eta);
        end
        frame_autocorelation(eta+1) = sum;
    end
end
%method to apply 3-level-center-clipping on a frame
function frame = three_level_center_clipping(frame_i)
    frame = zeros(1, length(frame_i));
    maximum = max(abs(frame_i));
    c = 0.3;
    for i = 1:length(frame)
        if frame_i(i) >= c * maximum
            frame(i) = 1;
        elseif frame_i(i) > -(c * maximum) && frame_i(i) < c * maximum
            frame(i) = 0;
        elseif frame_i(i) <= -(c * maximum)
            frame(i) = -1;
        end
    end
end

%method to calculate cepstrum of a frame
function frame_cep = cepstrum(frame_i)
    frame_cep = real(ifft(log(abs(fft(frame_i)))));
end
%method to apply high time liftering on a frame
function ceps = highTimeLifter(frame)
    NN = 20;
    ceps = zeros(1, length(frame));
    for i=1:length(ceps)
        if i >= NN
            ceps(i) = frame(i);
        else
            ceps(i) = 0;
        end
    end
end
% This function smooths the data in pitchFreq using a mean filter over
a
% rectangle of size (2*rows+1)-by-(2*columns+1).
function outputData =  Smooth(pitchFreq,rows,columns)
    frame_length(1) = rows;
    if nargin < 3
        frame_length(2) = frame_length(1);
    else
```

```
        frame_length(2) = columns;
    end
    [row,col] = size(pitchFreq);
    % Building matrices that will compute running sums.  The left-
matrix, RS,
    % smooths along the rows.  The right-matrix, CS, smooths along the
    % columns.
    RS = spdiags(ones(row,2*frame_length(1)+1),(-
frame_length(1):frame_length(1)),row,row);
    CS = spdiags(ones(col,2*frame_length(2)+1),(-
frame_length(2):frame_length(2)),col,col);
   % Setting all "NaN" elements of "pitchFreq" to zero so that these
will not
   % affect the summation.
    A = isnan(pitchFreq);
    pitchFreq(A) = 0;
    nrmlize = RS*(~A)*CS;
    nrmlize(A) = NaN;
    outputData = RS*pitchFreq*CS;
    outputData = outputData./nrmlize;
end
```