PITCH ESTIMATION AND VOICING DETECTION

Daniel Moreno Manzano

Address - Line 1
Address - Line 2
Address - Line 3

Contents

ABSTRACT

In this paper, I am going to explain the development of two different approaches in order to find the pitch in a voice audio, differentiating first the voiced and the unvoiced parts. The algorithms will be developed in MATLAB©. First, the data will be analyzed in order to find significant parameter in the audio files database to work with and in the interest of understanding how to work with the pitch of the human voice. In each of the approaches a brief explanation of the idea will be proposed. Moreover, the significant code parts will be explained and some results exposed.

1. INTRODUCTION

The way to proceed with the data will be the same with both algorithms and also in the analysis. The audio files will be evaluated in windows of 32 ms, as soon as it is the best way to detect the pitch just in the voiced parts and not in the unvoiced ones. The window will be moved in periods of 15 ms when working with the `fda_ue` database (the **test database**) and in periods of 10 ms when working with the `ptdb_tug` database (the **train database**).

The signal processing methods that I have chosen are:

- Autocorrelation
- Zero Crossing + Cepstrum

Claim: It is important to think in this project as an experiment in order to learn about pitch processing and estimation.

1.0.1. Data

The data used can be achieved from the next Google Drive link:

Download Data

The predefined directory scheme, in order to work without changing the code, is

```
.
data
  fda_ue
  ptdb_tug
      FEMALE
        MIC
            F01
            ...
            F10
      MALE
          MIC
              M01
              ...
              M10
```

1.0.2. Github

And the whole project can be found in Github:

Pitch-Estimation-and-Voicing-Detection

2. DATA ANALYSIS

This section's deep purpose is to understand the difference between the voiced and the unvoiced parts, but working with the proposed algorithms.

The database that we are using also stores `.f0ref` homonym files that have the correct pitch solved, window per window. So, in order to classify the data, first this files from the training database will be loaded and compared with the signal processing methods results. For now on, in this section, the color red in histograms will represent the **unvoiced class** while the blue color will represent the **voiced class**.

It is easy to think that the amplitude of the signal will give us a clue of where the voiced parts are, but this approach does not take into account a lot of talking sounds that are extensions of words, almost mute but important. An example of this could be the 's' sound in the final part of a word. So, this way of thinking is not enough. In any case, an extension of this will be proved.

2.1. Autocorrelation

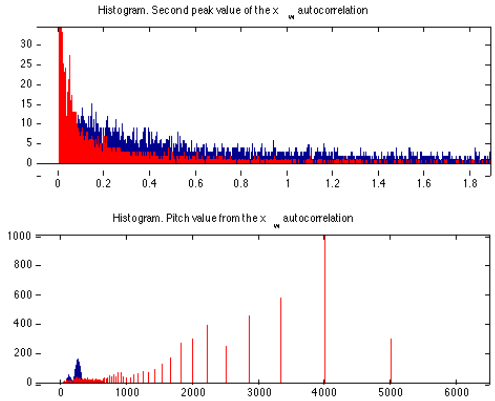
Despite the autocorrelation method and the way of obtaining the pitch frequency is described in the next section, important ideas and results will be explained on how the autocorrelation gives evidences to distinguish voiced of unvoiced parts in the audio.

The first intuition will give us a demonstration of the theory that states that the pitch frequency of the humans are in bands of:

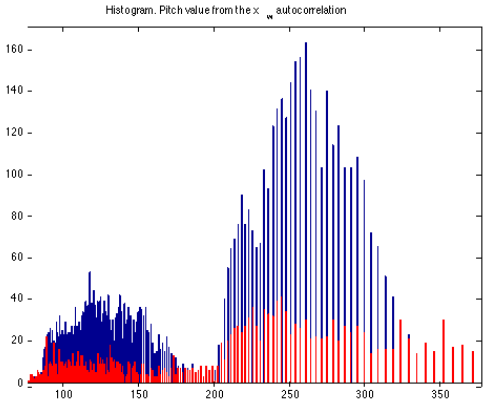
- Male voice: 85 - 180 Hz
- Female voice: 165 - 255 Hz

This intuition begins with the question of, knowing already the pitch value from the `.f0ref` files, *"how are the pitch and the unvoiced-voiced classes related?"*. And, also, *"how are the second peak amplitude and the unvoiced-voiced classes related?"*.

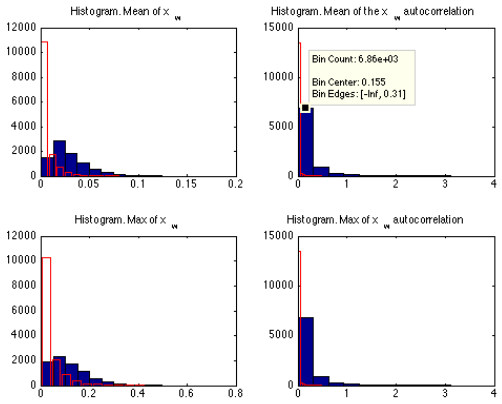
It will be seen that, for the second question, the behaviour of the second-peak highness is more or less the same and it cannot be use as a prior in a classifier model. But, in the case of the pitch analysis, it can be seen in the next histogram picture that the behaviour of the classes is really different. In fact, this information can give us the possibility of doing a filter instead of a classifier in order to post-process errors. There are zones where (depending on the pitch value) is more probable be in the voiced class that in the unvoiced.



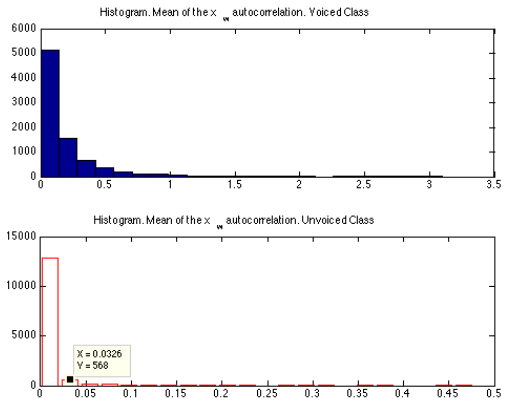
Zooming in, two pitch bands are characterized. And they are the bands from the theory. So, our database behaves like expected and we can filter the possible errors.



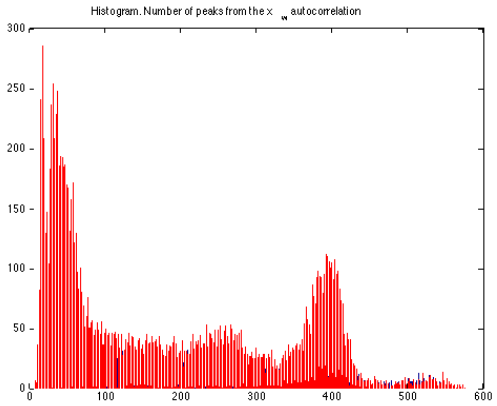
The second intuition lead us to a mean-max study, looking for the answer to the *"how are related the mean and maximum values of the function to the voiced-unvoiced classes related?"* question. Then, we repeat the previous histogram analysis and we compare the values for the mean of the signal and the autocorrelated signal and, also, the maximum values of them, differentiating between the voiced and the unvoiced classes. The four histograms are shown below.



It can be seen a marked difference for the autocorrelated signal analysis. Even more for the mean one. The zero mean value is more common for the unvoiced values than for the voiced ones. In fact, a threshold can be easily set in order to classify voiced and unvoiced parts, because the value of the mean for the unvoiced class decays in a very hard manner. The previous histogram will be shown below in a bigger picture.



Using this threshold is useful, but the error probability is not trivial. This happens because there are also a high amount of voiced signals that have a mean value lower than the threshold. Therefore, seeking a classification that do not depend on the signal's amplitude an analysis of a new intuition is done. This time the analogous question is *"How are the number of peaks and the voiced/unvoiced classes related?"*.

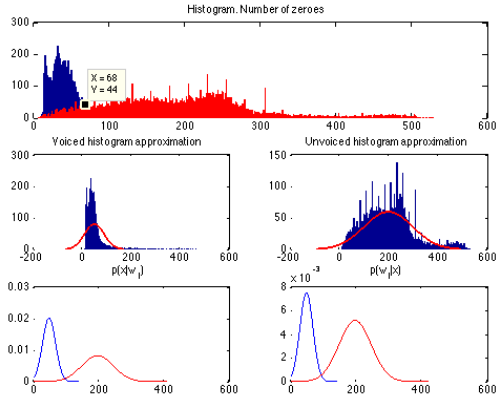


In the previous picture can be seen that the number of peaks is not giving us information in order to classify the data. So, finally, the decisor that is going to be used will depend on the autocorrelation mean value.

2.2. Zero crossing

As in the section ahead, the data analysis using the algorithm method will be done before the method explanation. In this case the zero crossing bring us the question of *"how are the number of zeroes in a signal related with the voiced/unvoiced classes?"*.

So, a very clear pair of gaussians can be interpreted in the resultant histogram and it will be used in order to perform an optimal bayesian decisor. In the next picture, the described results and the gaussian transformation are shown. Moreover, it will be seen the posterior probability $p(w_i/x)$ with an error probability of 0.16.



For this reason, a bayesian decisor will be implemented in the second algorithm. With the correct weights looking for the optimal loss function in order to reduce the risk to the maximum value. As soon as the limit between classes in the previous histogram is around 67, it could say that is more risky decide voiced in this zone. So, maybe, a good weight could be the one that makes the gaussian limit of around 100 more similar to the 67 one.

3. FIRST ALGORITHM. AUTOCORRELATION METHOD

3.1. Theory

The autocorrelation of a signal is often used in time domain signal processing. It always result a function with the biggest peak in the middle. Behaving like the sinc() function, it also has more peaks surrounding the principal one. Formerly speaking, the autocorrelation is the correlation of a signal with itself.

$$r(\tau) = x(t) \otimes x(t)$$

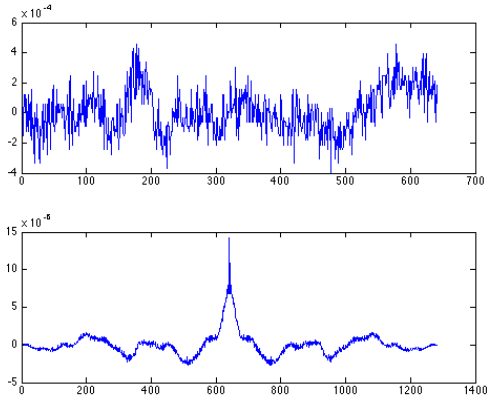
In this case, as soon as we are working with digital signals (sampled audio), the before statement is equivalent to:

$$r[m] = x \otimes x = \sum_i x[i]x[i+m]$$

The autocorrelation has several characteristics:

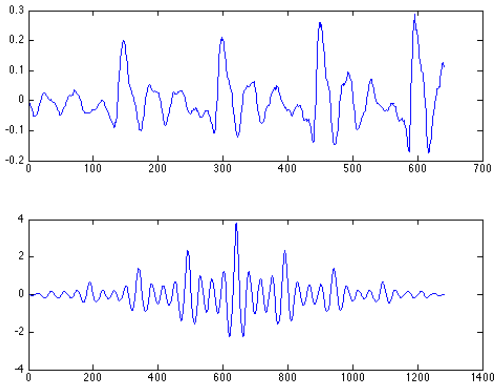
- It is always symmetric
- The clearer the autocorrelation is, the less random the original signal is
- Each of its peaks (and the peak position) has a lot of information about the behaviour of the signal

An illustrative figure of a noisy signal from the database and its autocorrelation is shown below.



As it can be seen, the autocorrelation function from a noisy signal is also noisy and has some irregularities. For example, the big peak next to the main one is not the second highest peak, as it should be when working with a periodic signal.

In the next figure, a voiced sample from the database an its autocorrelation is shown instead. Notice that in this case the main peaks are behaving as expected. The big peaks highness decay in a exponential manner from the main center peak.



An example of the information given by the principal peaks is the pitch. The location of the next peak to the principal one gives an estimate of the period, and the height gives an indication of the periodicity of the signal [3]. So, the pith frequency could be estimated as:

$$f_{pitch} = \frac{1}{d_{p0-p1}}$$

3.2. Code

In this section, the implemented code will be described and the main function parts will be written. Moreover, a section of the solved problems will extend the code description. The chosen programming language for the development is Matlab©.

3.2.1. `Getpitch`

This function is the main function responsible of the pitch calculation for the first algorithm, following the autocorrelation method. It needs the audio database `.wav` files, the database directory location, the window size (32 ms) and the window shift (15 or 10 ms on the database) as parameters. It will return four cells:

- `signals`. It stores all the signals from each database file.
- `pitch_array`. This variable stores all the pitch arrays (a pitch value per window, forming the array) for each file.
- `mean_pitch`. It stores the mean pitch values arrays (working as the pitch array) for each file.
- `fs_array`. Storing the sampled frequency of each file.

```
function [signals, fs_array, pitch_array, mean_pitch] =
  ↪  getPitch(wav_files, database_dir, w_L, shift)
```

As soon as we will work with a window per window analysis that moves in a number of milliseconds no proportional to total window size, the file should be expanded twice the shift movement (15 or 10 ms in the beginning and end of the file) to fully analyze the audio.

```
...

for f=1:n_files

    m_pitch = [];

    [x, f_s] =
  ↪  audioread(strcat(database_dir,wav_files(f).name));
    signals{f} = x;
    fs_array{f} = f_s;

    x = expandVector(x,shift,f_s);

    ...
```

where `m_pitch` is the array where all the pitch values from a file will be stored, `x` is the signal variable and `f_s` is the sampled frequency of the audio. In order to make the window shift correctly

```
...
```

```
sampl_numb = length(x);
audio_t = (sampl_numb/f_s)*1e3; %in ms

for counter = 0:shift:(audio_t-w_L) % The 32ms window
 ↪  moves every 15ms
       start = f_s*counter/1e3+1;
       finish = f_s*(counter+w_L)/1e3+1;

       x_w = x(start:finish);

       ...
```

where `x_w` is the signal portion of 32ms window size, `counter` is the auxiliary variable that count the audio fractions and `audio_t` is the time of the whole audios in milliseconds.

The autocorrelation method is implemented in code as

```
rx_wx_w = xcorr(x_w);
```

and the classifier of the voiced/unvoiced classes is

```
if (mean(abs(rx_wx_w))>0.02) % If this sample is voice,
 ↪  then
```

It can be seen that, after the previous data analysis the threshold set is 0.02 in order to differentiate between voiced and unvoiced. Then, the pitch calculation is done by

```
[peaks, places] = findpeaks(rx_wx_w);
[max_peak, max_peak_place] = max(peaks);
max_peak_place = places(max_peak_place); % First max place
[max_peak2, max_peak2_place] =
 ↪  max(peaks(ceil(length(peaks)/2)+1:end)); % Finding
 ↪  the secon peak
places = places(ceil(length(peaks)/2)+1:end);
max_peak2_place = places(max_peak2_place); % Second max
 ↪  place

pitch = [pitch; f_s/(max_peak2_place-max_peak_place)];
m_pitch = [m_pitch; pitch(end)];
```

Finally, this function will write the `.f0` files with the pitch information (audio per audio) as in the `.f0ref` format in order to compare them later looking for some results.

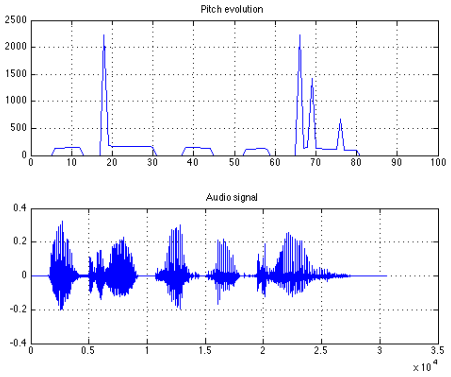
```
audio_name = strsplit(wav_files(f).name, '.');
fileID =
 ↪  fopen(strcat(database_dir,audio_name{1},'.f0'),'w');

fprintf(fileID, '%g\n', m_pitch);
fclose(fileID);
```

3.2.2. Problems

1. Impossible pitch peaks

   After calculating the pitch value with the `getPitch()` function, some unexpected pitch peaks appear. In the next picture, an example is shown. But, this is a problem easy to solved, thanks to the previous pitch analysis.

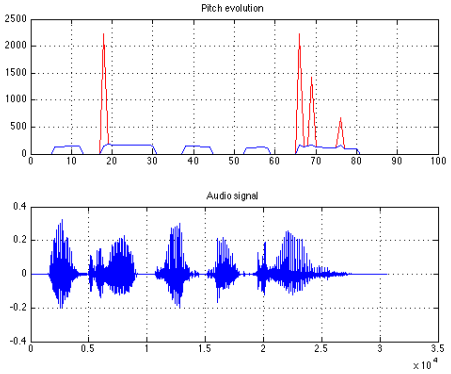


As soon as our audio database corresponds to the human voice theory that states a pair of bands for the human pitch one for the female pitch and other for male one, a filter can be used.

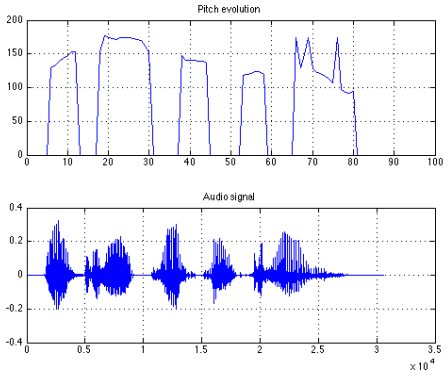
```
% HUMAN VOICE RANGE:
% - Male: 85 to 180 Hz
% - Female: 165 to 255 Hz
human_max_range = 255;
human_min_range = 85;

if (f_s/(max_peak2_place-max_peak_place) >
 ↪  human_max_range) ||
 ↪  (f_s/(max_peak2_place-max_peak_place) <
 ↪  human_min_range) % Filter to detect errors
 ↪  not filtered before
  pitch(end) = mode(pitch);
end
```

After filtering, it can be seen in the next example that the peaks are eliminated. A comparison of the previous pitch evolution (red) and the filtered one (blue) is shown below.



And the final result pitch behaviour is



That behaves in a more regular way.

2. Voiced not well detected

   As it can be seen in the pictures before, despite of the mean classifier, there are some error detecting the voiced and unvoiced parts. Some of the pitch signal were not exactly covering the audio parts where clearly were voiced parts.

   As seen in the data analysis, another way to improve the classification that do not depend on the signal's amplitude is needed.

3.3. Results

The results of the autocorrelation algorithms are very good. This is the best algorithm performed in this experiment, as it will be seen farther on the paper. Good results, simple and fast.

```
   ### Summary
Num. frames:        22130 = 13906 unvoiced + 8224 voiced
Unvoiced frames as voiced:        1181/13906 (8.5%)
Voiced frames as unvoiced:        1125/8224 (14%)
Gross voiced errors (+20%):        630/7099 (8.9%)
MSE of fine errors:        2.2%
--------------------------
```

4. SECOND ALGORITHM. TRAINED BAYESIAN DECISOR + CEPSTRUM

4.1. Theory

4.1.1. Bayesian decisor based in the Number of Zeroes

In order to implement the bayesian classifier, the results of the data analysis will be used. Therefore, following them and the Bayesian decision theory [2] the Bayes Decision Rule is

$$w_1 \underset{if}{\longrightarrow} P(w_1/x) > P(w_2/x)$$

where

- w_1 is the voiced class and w_2 is the unvoiced class.
- x is the number of zeroes

So, everytime the probability of the voiced class having the number of zeroes is higher than the probability of the unvoiced class having the number of zeroes the algorithm will start to calculate the pitch.

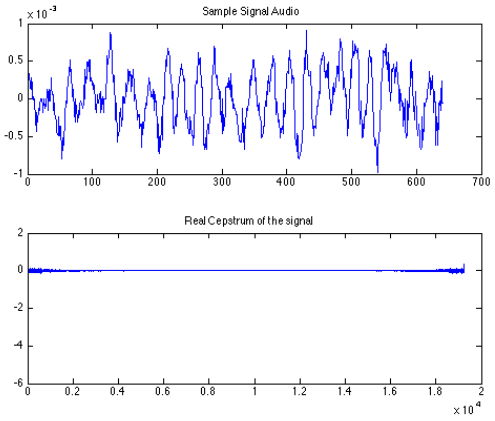
4.1.2. Cepstrum

Cepstrum is a very well known method for voice treatment despite off it flattens the spectrum, giving robustness to formant signals. But, the problems with Cepstrum are that it rises the noise level and that it needs a high resolution signal (with a big amount of samples) in order to work properly. Also, it is not a simple method to perform because the first peak quest is hard, despite of the sensibility to errors, resulting in a wrong pitch approximation.

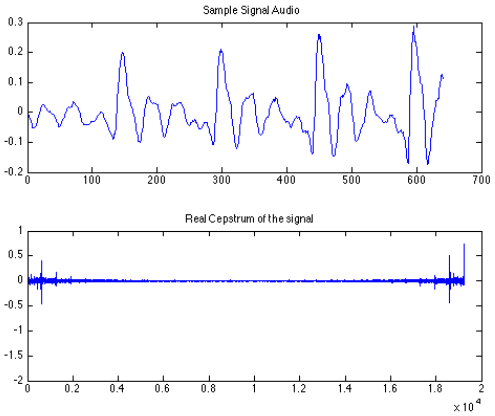
The Cepstrum method is described mathematically as

$$C(\tau) = \mathcal{F}^{-1}\left\{\log(|\mathcal{F}\{f(t)\}|^2)\right\}$$

The next picture draw a noisy signal and its Cepstrum processed signal



and this one depict a voiced signal also compared with its Cepstrum. It should be noticed that the voiced signal has a very clear peak, meaning that the signal has periodicity.



The real Cepstrum is often used than the complex one, despite of for the unvoiced speech zones only the Real Cepstrum has meaning [3]. Finally, the fundamental frequency is estimated in the same way as in the autocorrelation method [1]

$$f_{pitch} = \frac{1}{d_{p0-p1}}$$

4.2. Code

This algorithm is not so fast, because it has to learn first from the known data.

4.2.1. `learnFromData`

The needed inputs will be:

- `data`. The files of the referenced data.
- `database_dir_data`. The directories of the referenced data.
- `audios`. The audios to process.
- `database_dir_audios`. The directories of the audios.
- `w_L`. Window length.
- `shift`. Shift movement.

and the outputs will be the probabilistic voiced and unvoiced models.

```
function [voiced_model, unvoiced_model] =
  ↪   learnFromData(data, database_dir_data, audios,
  ↪   database_dir_audios, w_L, shift)
```

In order to model the classifier, the reference files should be loaded and stores in the `w_prior` variable.

```
...

for f=1:n_files

w_prior = [w_prior;
  ↪   textread(strcat(database_dir_data,data(f).name))];

...
```

As in the previous algorithm, the audios are read and expanded, also the time in milliseconds of the audio and the window movement is calculated. Then, in order to create the Bayesian decisor based on the number of zeroes as evidence, the function begins counting the number of zeroes or, better, the number of times that the audio crosses the x-axis. As soon as non-continued function like this one can cross the axis without giving a zero value, the crossed time should be count instead of just the number of zeroes.

```
...

zeroes_n = 0;

    for k = 2:(w_L*f_s*1e-3)
        if x_w(k) == 0
            zeroes_n = zeroes_n+1;
        elseif (x_w(k) > 0 && x_w(k-1) < 0)
            zeroes_n = zeroes_n+1;
        elseif (x_w(k) < 0 && x_w(k-1) > 0)
            zeroes_n = zeroes_n+1;
        end
    end

    z_array = [z_array; zeroes_n];

    ...
```

The classes should be found

```
voiced = w_prior;
voiced(voiced>0) = 1;

labelsVoiced = find(voiced>0);
voicedSamples = z_array(labelsVoiced);
labelsUnvoiced = find(voiced<1);
unvoicedSamples = z_array(labelsUnvoiced);
```

Finally, the gaussians are modeled following the distribution function of

```
x=1:length(z_array); %maximum number of samples required
likelihood_voiced =
  ↪   1/(sqrt(2*pi*var(voicedSamples)))*exp(-1/(2*var(voi
  ↪   cedSamples))*(x-mean(voicedSamples)).^2);
likelihood_unvoiced =
  ↪   1/(sqrt(2*pi*var(unvoicedSamples)))*exp(-1/(2*var(u
  ↪   nvoicedSamples))*(x-mean(unvoicedSamples)).^2);

prior_voiced = length(labelsVoiced)/length(x);
prior_unvoiced = length(labelsUnvoiced)/length(x);

voiced_model = prior_voiced*likelihood_voiced;
unvoiced_model = prior_unvoiced*likelihood_unvoiced;
```

4.2.2. `getPitchCepstrum`

In this case, just the pitch vector will be returned, but as the `getPitch` function, files will be printed. `getPitchCepstrum` will need the same parameters than `getPitch` but, in addition, the voiced and unvoiced bayesians model should be introduced.

```
function [pitch_array] = getPitchCepstrum(audios,
  ↪   audios_database_dir, voiced_model, unvoiced_model,
  ↪   w_L, shift)
```

```
...

if voiced_model(zeroes_n)>unvoiced_model(zeroes_n)
x_cepstrum = rceps(x_w);
...
```

```
pitch = max_peak_place*1e3/w_L
```

4.2.3. *Problems*

1. Windows detected difference between the reference and the calculated pitch

   As soon as there were some errors when comparing the read vector from the reference files, I decided to

limit the number of windows shifts equal the number of shifts done in the algorithm, so

```
 if length(w_prior) - length(z_array) == 1
    w_prior = w_prior(1:end-1);
elseif length(w_prior) - length(z_array) == -1
    z_array = z_array(1:end-1);
end
```

2. Impossible pitch peaks

   As in the previous algorithm a filter to solve this problem is implemented. Moreover, in order to find the real peak in the Cepstrum signal, it will be filtered looking for a smoother version of it. In order to use the best approximation, a previous test of filter was done. In the next picture, some results are offered.

```
% HUMAN VOICE RANGE:
       % - Male: 85 to 180 Hz
       % - Female: 165 to 255 Hz
       human_max_range = 300;
       human_min_range = 80;
       min_cep_distance = ceil(f_s/human_max_range);
       max_cep_distance = ceil(f_s/human_min_range);

       cepstrum_fltrd = smooth(x_cepstrum,
 ↪   'rloess');
```

3. Cepstrum resolution

   Cepstrum analysis requires much more samples than the samples that we have. For that reason an extension is done with 30 times more samples.

```
x_cepstrum = real(ifft(log(fft(x_w,
 ↪   length(x_w)*30)))); % more samples in order
 ↪   to have a good Cepstrum resolution
```

4.3. Results

The final results for the Cepstrum algorithm will be very bad.

```
   ### Summary
Num. frames:         22127 = 13903 unvoiced + 8224 voiced
Unvoiced frames as voiced:         240/13903 (1.7%)
Voiced frames as unvoiced:         7204/8224 (88%)
Gross voiced errors (+20%):         670/1020 (66%)
MSE of fine errors:         9%
--------------------------
```

5. REFERENCES

[1] Pitch detection methods. `http://sound.eti.pg.gda.pl/student/eim/synteza/leszczyna/index_ang.htm`. Accessed: 2017-03-20.

[2] Richard O Duda, Peter E Hart, David G Stork, et al. *Pattern classification*, volume 2. Wiley New York, 1973.

[3] Xuedong Huang, Alex Acero, and Hsiao-W Hon. *Spoken Language Processing: A Guide to Theory, Algorithm, and System Development*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 2001.