Fundamental Frequency, Encoding and Decoding of Speech

Jan Černocký, Valentina Hubeika, FIT BUT Brno

1 Detection of Fundamental Frequency

Initially, remise the principles of the methods based on ACF (autocorrelation function) and NCCF (Normalized Cross-Correlation Function). In both cases:

- Delay the signal by a potential value of the lag (period of pitch in samples).
- Multiply and sum the corresponding values.
- Identify the maximum in the resulting vector. If the maximum is significant, that is greater than the (threshold × zero autocorrelation coefficient), denote the frame as voiced. The position of the maximum is the lag. If the maximum is not significant, the frame is unvoiced.

The difference between the ACF and the NCCF is that the ACF is applied on the samples from just one frame and "does not see" outside the frame. The NCCF uses the history of the signal (from the previous frame) when shifting the signal.

First, load the signal, apply mean normalization and segment it to frames (with no overlap). Use either the signal `test.wav`, or any other signal you prefer. For the beginning, play with just one frame

```
% signal
s = wavread ('test.wav');
sm = s - mean(s);
plot (sm); sound (sm);
sr = frame (sm, 160, 0); % no overlap !
% playing with one frame - 7th one is the nice one ...
x = sr (:,7);
plot (x);
```

1.1 ACF

Calculate the autocorrelation coefficients on the selected frame:

```
% pitch detection from pure ACF
R = xcorr (x);
R = R(160:end);
n = 0:159; plot (n, R);
```

the minimum and the maximum values of lag have to be defined. The maximum lag has to be less than 160 (introduced in the lecture), because the frame length is 160 (the lag length has to be smaller). The threshold is estimated experimentally. Be careful with the Matlab indexing (starts from 1)...

```
% some constants
Lmin = 20; Lmax = 146;
thr = 0.3; % maximum needs to be at least thr * R[0]
% detect lag
[Rmax,ii] = max(R((Lmin+1):(Lmax+1)));  % needs to add 1 because of Matlab indexing
if Rmax >= thr * R(1)  % R[0] in Matlab indexing
  L=ii+Lmin-1;         % and here needs to remove it again...
else
  L=0;
end
hold on; plot (L,Rmax,'or'); hold off;
```

Look at the calculated lag value, the fundamental period in seconds and pitch in Hz:

```
% show lag in samples
L
% in seconds
T0 = L / 8000
% and fundamental frequency in Hz
F0 = 1/T0
```

1.2 NCCF

Use pointers to the boundaries of the selected frame instead of the frame matrix `sr`. Calculate the energy of the frame. The energy has to be calculated only once and will be used for normalization:

```
% NCCF - we choose the same portion of signal as 7th frame.
from = (7-1) * 160 + 1; to = from + 160 -1;
selected = sm(from:to);  % nonshifted frame
x - selected             % check that it is really the same one ...
E1 = sum(selected .^ 2); % energy of non-shifted frame.
```

The following cycle defines frame delays `shifted` and calculates NCCF for each shift. Note, that $NCCF[0]$ is calculated (does not have to as it is always 1), then NCCF is calculated within the range of the potential lag values. Run the cycle with displaying current values: press `Enter`, look at the relation of `selected` to `shifted` and at the calculated NCCF coefficient. After running a few iterations, comment the displaying line and run the complete cycle at once.

```
% in the following cycle, look at a few values, then comment out the plot and pause
Rnccf = zeros(1,Lmax + 1);
for n = [0 Lmin:Lmax]
  froms = from-n; tos = to-n; % indexes of the shifted frame
  shifted = sm(froms:tos);
  plot(1:160,selected,1:160,shifted); pause;
  E2 = sum(shifted .^ 2); % energy of the shifted one
  numerator = selected' * shifted;
  nccf = numerator / sqrt(E1 * E2)
  Rnccf(n+1) = nccf;  % Matlab indexing.
end
n = 0:Lmax; plot (n, Rnccf);
```

Finally, detect the lag:

```
% detect lag
[Rmax,ii] = max(Rnccf((Lmin+1):(Lmax+1)));  % needs to add 1 because of Matlab indexing
if Rmax >= thr * Rnccf(1)  % R[0] in Matlab indexing
  L=ii+Lmin-1;             % and here needs to remove it again...
else
  L=0;
end
% show lag:
L
```

1.3 Pitch Detection for the Complete Signal, Estimation Correction

Pitch has to be estimated for **all** frames. Use the functions `lag_acf` and `lag_nccf`. Loot at the help of the functions. The function `lag_acf` implements what you have seen above. The function `lag_nccf` calculates the coefficients also at the beginning of the signal.

```
Lacf = lag_acf (sm,160,0,20,146,0.3);
Lnccf = lag_nccf (sm,160,0,20,146,0.7);
subplot(411); plot(sm); axis tight
subplot(412); plot(Lacf); axis tight
subplot(413); plot(Lnccf); axis tight
```

We can see that the detecting using NCCF is better than using ACF, but still there are mistakes present (double lags). Do the corrections by means of a median filter:

```
Lnccf_med = medfilt1 (Lnccf, 5);
subplot(414); plot(Lnccf_med); axis tight
```

Try detection for a long signal `train.wav` used previously:

```
% testing on some longer signal ... train.wav
s = wavread ('train.wav');
sm = s - mean(s);
Lacf = lag_acf (sm,160,0,20,146,0.3);
Lnccf = lag_nccf (sm,160,0,20,146,0.7);
subplot(411); plot(sm); axis tight
subplot(412); plot(Lacf); axis tight
subplot(413); plot(Lnccf); axis tight
Lnccf_med = medfilt1 (Lnccf, 5);
subplot(414); plot(Lnccf_med); axis tight
```

The result shows that pitch detection is not a trivial task and that, for human speech, algorithms can fail.

Tasks

1. Check if $NCCF[0]$ is always 1.
2. Check how the function `lag_nccf` solves the calculation in the beginning of the signal.
3. How would you implement the NCCF for on-line processing when the whole signal is not available?
4. The thresholds 0.3 for ACF and 0.7 for NCCF are just rough estimates. Can you get better results by adjusting the thresholds?
5. Try different lengths of median filter. What can you say about the result?

2 Coding

The goal is to implement a ZRE-super :-) speech encoder with the bit-rate of 1050 bit/s. The output of the encoder is a text-file containing integer numbers that can be encoded by a fixed number of bits.

2.1 Parameter Estimation and Encoding without Quantization

Based on the function `param.m` from the previous lab, a new function `param2.m` implements LPC analysis and pitch detection for all frames. Loot at the function help. Function `synthesize.m` implements synthesis of speech. Look at the function help. First, analyse and synthesize speech with **no** quantization of the parameters:

```
s = wavread ('train.wav');
sm = s - mean(s);
% first parameterization function:
[A,G,L,Nram] = param2 (sm,160,0,10,20,146,0.7);
% and synthesis
ss = synthesize (A,G,L,10,160);
plot(ss); soundsc(ss);
```

Tasks

1. Find out how the function `synthesize.m` ensures the excitation is of the unit energy (power) per sample before it is multiplied by gain.
2. Find out how the function generates voiced excitation. What does the variable `nextvoiced` contain?
3. Will the function still work properly in case the maximum lag exceeds the frame length?

4. Extend the function by the plots of excitation and the resulting frame — add `plot(...); pause` and look at the plots.

5. The excitation in voiced frames is realized in the simplest possible way – a sequence of impulses. Is it possible to improve the quality. (So the resulting speech sounds more 'human'?)

6. How do you think the excitation is implemented in GSM encoders ?

2.2 Encoding

The lag values are 0,20. . . 126, which means the lag is quantified on 7 bits. The values of the coefficients and the gain are real in the format double. That is one frame carries:
`7 + 64 * 10 + 64 * 1` bits of information (this is too much).

Filter parameters quantization is done by vector quantization. Vector quantization was introduced in the previous lab. There is a code-book with 210 code vectors prepared for the today's lab: `cb210.txt`[1] After the quantization the number of the bits for coding the coefficients per frames decreases to `8`.
Gain is quantized using scalar quantization on 64 levels. The code-book is available in `gcb64.txt`. However we work with scalars here, we can use the `vq_*` functions implemented for vector quantization [2]. The required number of bits is `6`.

Note, that this is a school example – both code-books were trained on the speaker (male) from the file `train.wav`. The code-books are thus speaker dependent which means they cannot be used for different speakers. . .

```
load cb210.txt
load gcb64.txt

s = wavread ('train.wav');
sm = s - mean(s);
% first parameterization function:
[A,G,L,Nram] = param2 (sm,160,0,10,20,146,0.7);
[asym,gd] = vq_code(A, cb210);
gsym = vq_code(G, gcb64);
% decoding
Adecoded = cb210(:,asym);
Gdecoded = gcb64(:,gsym);

% and synthesis
ss = synthesize (Adecoded,Gdecoded,L,10,160);
plot (ss); soundsc(ss);
```

Tasks

1. Verify, that the bit-rate is 1050 bit/s. Solution: `(8 + 7 + 6) * 50 .`

2. Compare this bit-rate to the GSM coders.

3. Why does the LPC parameters code-book has the size of 210 and not 256? Look at `hrani.m`.

4. How was the gain code-book created?

5. Is the output speech comprehensible?

6. Does the output speech sound natural?

7. How would you value the implemented coder according to MOS (mean opinion score – see the lecture about encoding) ?

[1]the details on the code-book generation can be found in `hrani.m`, the functions `vq_*` are used as in the previous lab.
[2]the scalar code-book generation is implemented in `hrani.m`

2.3 Coding of the other Signals

Now, encoding and decoding are wrapped by two functions. Look at their help!

- `coder` converts a WAV file into a text file, where each raw carries information about 1 frame: the index to the LPC coefficients code-book (8 bits), the index to the gain code-book (6 bits) and the lag (7 bits). As the auxiliary Matlab output, the function produces the mean-normalized signal (corresponding to the read signal).
- `decoder` decodes the signal based on the parameters stored in the text file and saves the resulting signal in a WAV file. As the auxiliary Matlab output, the functions produces the decoded signal.

First, code a signal of the same speaker `testspdat.wav`. Listen to the signal and evaluate the quality of the obtained speech: do you understand it? Now code the original speech signal:

```
s = coder('testspdat.wav', '/tmp/koko');
ss = decoder ('/tmp/koko', '/tmp/testspdatd.wav');
soundsc(ss); pause; soundsc (s);
```

Next, test the implemented coding on a different speaker:

```
s = coder('ses0019.wav', '/tmp/koko');
ss = decoder ('/tmp/koko', '/tmp/testspdatd.wav');
soundsc(ss); pause; soundsc (s);
```

Finally, use a recording from a female speaker:

```
s = coder('ses0099.wav', '/tmp/koko');
ss = decoder ('/tmp/koko', '/tmp/testspdatd.wav');
soundsc(ss); pause; soundsc (s);
```

Tasks

1. Is the obtained speech comprehendible ?
2. Does the obtained speech sound natural ?
3. How would you value the coder according to MOS (mean opinion score) ?
4. Check whether the resulting file does not contain values higher than 1050 bit/s.
5. Look at the functions `coder` and `decoder` – complete display of the original and the decoded parameters and try to find out where is the main problem with the quality.
6. Verify the stability of all the filters (the coefficients are stored in the code-book `cb210.txt`).
7. Try to encode and decode your own signal (record something or download something from the Internet. If it does not correspond to the format of 8 kHz, 1 channel and 16 bits, use `sox` for the convertion).
8. Suggest some improvements of coding !