LPC and Vector Quantization

Jan Černocký, Valentina Hubeika FIT BUT Brno

When modeling speech production based on LPC, we assume that the excitation is passed through the linear filter: $H(z) = \frac{G}{A(z)}$, where $A(z)$ is a P-th order polynome:

$$A = 1 + a_1 z^{-1} + \ldots + a_P z^{-P}, \tag{1}$$

and G is **gain** of the filter.

The coefficients of the filter and its gain can be calculated using autocorrelation coefficients. In Matlab, using the function `lpc`.

1 Signal and Autocorrelation Coefficients Estimation

First, we need to load a signal, apply mean normalization on it and segment it to frames (with no overlap). There is a test signal included in the archive `test.wav`, but you can use any other signal. Extract one frame to play with:

```
s = wavread ('test.wav');
sm = s - mean(s);
plot (sm); sound (sm);
sr = frame (sm, 160, 0); % no overlap !
% one frame to play with
x = sr (:,7);
plot (x);
```

The first step in LPC estimation is calculation of the autocorrelation coefficients. For the order P, the coefficients $R[0] \ldots R[P]$ are required. The demonstration of the calculation is presented by the following example. Run the code, and follow the calculation for each shift by pressing `enter`.

```
% autocorrelation (unnormalized!) - hand job
P = 10;
R = zeros(P+1,1);
N = 160;
for k = 0:P,
  % to be able to multiply the signals, they must have same length
  xaux = [x; zeros(k,1)];
  xauxshifted = [zeros(k,1); x];
  plot (1:(N+k), xaux, 1:(N+k), xauxshifted); pause;
  r = sum (xaux .* xauxshifted) % this would also work with scalar product
  R(k+1) = r;  % attention, Matlab indexes from 1 :-(
end
```

Matlab has an embedded function for the autocorrelation coefficients calculation, compare the output from the previous example and the output of the function `xcorr`:

```
Rmatlab = xcorr(x,'none');  % none means no normalization
Rmatlab = Rmatlab(N:(N+P)); % select only 0 to P
[R Rmatlab]
```

Tasks

1. Why only the samples `N:(N+P)` have to be selected from the output `xcorr` ?
2. What the parameter `'none'` stands for in the function call `xcorr` ?

2 LPC Coefficients Estimation

First, do all calculations "by hand" according to the equation system 19 in the lecture slides. The equation system is calculated by the Matlab matrix inversion:

```
% put together sides of equation 19 from slides
RLEFT = toeplitz(R(1:P))  % should be R[0] to R[P-1]
RRIGHT = -R(2:(P+1))      % should be R[1] to R[P]
% now just solve  RLEFT * a = RRIGHT for a:
adirect = inv(RLEFT) * RRIGHT
```

During lectures we introduced the Levinson and Durbin algorithm. The algorithm gradually calculates the predictors of the 1-st to P-th order and stores them into the columns of the matrix `A`. For each order predictor, the vector `E` contains the energy of the prediction error. Note, indexing in Matlab starts from 1, therefor, for the vectors `E` and `R` increase the indices by 1 (as they are defined in equations starting from 0).

```
% this terrible Levinson-Durbin (Eqs 22-26) ...
% attention to messy Matlab indexing - will define 'im' for Matlab 'i'
A = zeros (P,P); % will have generations of predictor in columns of A:
E = zeros (1,P+1); % will have residual energies in this vector

E(0+1) = R(0+1); % +1 because of Matlab
for i = 1:P,
  suma = 0;
  for j = 1:(i-1), % very slow implementation, just for teaching !
    suma = suma + A(j,i-1) * R(i - j +1); % +1 because of Matlab
  end
  ki = - ( R(i+1) + suma ) / E(i - 1 + 1); % +1 because of Matlab
  A(i,i) = ki;
  for j = 1:(i-1)
    A(j,i) = A(j,i-1) + ki * A(i-j,i-1);
  end
  E(i + 1) = (1 - ki ^ 2) * E(i - 1 + 1); % +1 because of Matlab
end
A
E
```

The resulting coefficients are then stores in the highest-index column of the matrix `A`. In Matlab, there is also an embedded function `lpc`, which calculated the parameters. Compare the results calculated "by hand", then by our implementation of the Levinson and Durbin and results by the function `lpc` (which also implements the Levinson and Durbin algorithm):

```
% getting the final predictor
amylevinson = A(:,P)
% and the easy way by Matlab ...
[a,elpc] = lpc (x,10)
```

Tasks

1. What the function `toeplitz` does?
2. Try to plot the graph of the error energy `plot(E)` and decide whether the order of the predictor (10) was a good choice.

3 Error Signal and its Energy

For the resulting predictor, we want to estimate the error signal $e[n]$ energy. It can be done in 4 different ways:

1. Generate the signal and calculate its energy:

2. Look at the high-index items of the vector `E` and our implementation of the LD algorithm.

3. Calculate the energy according to the equation 20 from the lecture.

4. Use the Matlab energy of the function `lpc` - Matlab calculates the normalized energy, so we have to multiply it by the frame length.

```
% playing around error signal and its energy - order P
e = filter (a,1,[x; zeros(P,1)]);
plot (e)
% energies
Esignal = sum (e .^ 2)   % from the signal
EfromLD = E(10 + 1)      % from Levinson Durbin
Eequation = R(1) + sum (a(2:P+1)' .* R(2:(P+1))) % from Eq 20
elpc * 160                  % Matlab computes normalized one !
```

4 Spectra

Look at the difference between the FFT and the LPC spectrum (e.g. spectral power density). We've done the FFT calculation before (apply zero padding to get the frame length to 1024):

```
% axis
Fs = 8000; f = (0:511) / 1024 * Fs;
% original
X = fft([x' zeros(1,1024-160)]); X = X(1:512); Gdft= 1/160 *abs(X) .^ 2;
Gdftlog = 10 * log10 (Gdft); plot(f,Gdftlog);
```

Plot the PSD estimation using LPC (implemented by the function `freqz`) to the same graphic:

$$\hat{G}_{LPC}(f) = \left| \frac{G}{A(z)} \right|^2_{z=e^{j2\pi f}}, \tag{2}$$

where f is the normalized frequency $f = F/F_s$.

```
% from filter parameters
G = sqrt (Eequation / 160);
Glpc = abs(freqz(G,a,512)) .^ 2; Glpclog = 10 * log10 (Glpc);
hold on; plot(f,Glpclog,'r','LineWidth',3); hold off;
```

Finally, plot the FFT spectrum of the error ("residual") of the LPC:

```
% residual
E = fft([e' zeros(1,1024-170)]); E = E(1:512); Gdft= 1/170 *abs(E) .^ 2;
Gdftlog = 10 * log10 (Gdft); plot(f,Gdftlog);
```

Tasks

1. What does LPC-spectrum represent: excitation, articulatory tract or both?

2. Reduce the predictor order to 2 (`lpc(x,2)`) and plot the LPC spectrum. What do you see?

3. Why the filter $A(z)$ is called "whitening"?

4. Is the signal $e[n]$ really noise, where are the samples uncorrelated? Which dependencies (regarding $x[n]$) are removed - the short-time or the long-time dependencies?

5 LPC over all the Frames and their Application

One frame was just a toy example, normally (encoding, recognition, ...) the LPC analysis has to be done over the all frames. The LPC coefficients (except $a_0 = 1$) are stored to a matrix (each column contains the coefficients of a frame), the gains are stored in a row vector.

```
Nram = size(sr,2)    % number of frames
Aall = zeros (P,Nram); % will store them without a0=1 !
Gall = zeros (1,Nram);

for n=1:Nram,
  [a,e] = lpc(sr(:,n), 10);
  a = a(2:(P+1))';          % discarding a0 = 1
  Aall(:,n) = a;
  Gall(n) = sqrt(e);
end
```

The first application, we try, is the error signal (residual) calculation over the whole signal. Each frame is filtered by the filter $A(z)$ with the corresponding coefficients, careful with the initial conditions of the filter (derived from the previous frame, see `help filter`). Plot the signal (try zoom) and listen to it.

```
% do residual of the whole signal - remember the state of the filter !
init=zeros(P,1);        % initial conditions of the filter
ebig = [];              % very inefficient, if long signal, pre-allocate !
for n = 1:Nram,
  x = sr(:,n);  a = [1; Aall(:,n)]; % appending with 1 for filtering
  [e,final] = filter (a,1,x,init);
  init = final;
  ebig = [ebig e'];
end
plot (ebig); sound(ebig);
```

Now, try synthesis using the filter $\frac{1}{A(z)}$, in each frame the excitation is the while Gaussian noise.

```
% do some synthesis with white noise !
init=zeros(P,1);        % initial conditions of the filter
syntbig = [];              % very inefficient, if long signal, pre-allocate !
for n = 1:Nram,
  a = [1; Aall(:,n)]; % appending with 1 for filtering
  G = Gall(n);
  excit = randn (1,160); % this has power one ...
  [synt,final] = filter (G,a,excit,init);
  init = final;
  syntbig = [syntbig synt];
end
plot (syntbig); sound(syntbig);
```

Finally, lets try something funny: the file `violinák.wav`[1] containes the sound of violin, use it as the excitation for the synthesis:

[1] z `http://ccrma.stanford.edu/~jos/waveguide/Sound_Examples.html`

```
v = wavread ('violin8k.wav');
vm = v - mean(v);
vr = frame (vm, 160, 0); % no overlap !
init=zeros(P,1);        % initial conditions of the filter
syntbig = [];              % very inefficient, if long signal, pre-allocate !
for n = 1:Nram,
  a = [1; Aall(:,n)]; % appending with 1 for filtering
  G = Gall(n);
  excit = vr (:,n); %  + 0.1 *  randn (160,1); % for better 's' !
  excit = excit ./ sqrt(sum(excit .^ 2) / 160); % normalizing to power 1.
  % sum(excit .^ 2) / 160
  [synt,final] = filter (G,a,excit,init);
  init = final;
  syntbig = [syntbig synt'];
end
plot (syntbig); soundsc(syntbig);
```

Tasks

1. Do you understand the residual of the signal ?
2. In the synthesis (excitation by the white noise and the violin) is necessary that the excitation signal has the unit energy (power). Verify, it holds - uncomment the line `sum(excit .^ 2) / 160`.
3. When violin sound is used as the excitation, generating the phoneme 's' can be problematic at the higher frequencies that the violin excitation does not contain. Try to mix the violin with the white noise (uncomment the line `excit = vr (:,n);`).

6 Functions for Parameterization and Synthesis

For further experimenting, wrap the coefficient and gain estimation into a function: `param`. There is also function `syntnoise` available. Try both functions on the file `train.wav` (contains a longer speech signal than in the previous example), which will be later used in vector quantization.

```
% param.m:
[A,G,Nram] = param ('train.wav',160,0,10);
% syntnoise.m:
ss = syntnoise(A,G,10,160);
soundsc(ss);
```

Vector Quantization (VQ)

7 Introduction to VQ

In speech processing, vector quantization is used for instance for bit stream reduction in coding or in the tasks based on HMM. In VT, P-dimensional parameter vectors $\mathbf{x}$ are translated into **symbols** using a **code book** of L coding vectors:

$$\mathbf{Y} = \{\mathbf{y}_i; 1 \leq i \leq L\}. \tag{3}$$

In the VQ, the vector $\mathbf{x}$ is assigned the code vector $\mathbf{y}_i$ corresponding to the minimum distance:

$$q(\mathbf{x}) = \mathbf{y}_i \quad \text{pokud} \quad d(\mathbf{x}, \mathbf{y}_i) \leq d(\mathbf{x}, \mathbf{y}_j), \;\; j \neq i, \; 1 \leq j \leq L. \tag{4}$$

The distance metric $d(\cdot, \cdot)$ can be arbitrary, here, quadratic metric (power of Euclidean distance) is used. It can be defined as a scalar product of the vectors $(\mathbf{a} - \mathbf{b})$ and its transpose (a row vector multiply by a column results in a scalar):

$$d(\mathbf{a}, \mathbf{b}) = \sum_{i=1}^{P} (a_i - b_i)^2 = (\mathbf{a} - \mathbf{b})^T (\mathbf{a} - \mathbf{b}) \tag{5}$$

Global distance is given by the sum of all minimum distances, normalized by the number of quantization vectors. The vectors that are quantized by a single coding vector make a so-called **cluster**.
The coding book is not known a priori. It has to be trained on the set of the training vectors. The approach is the following:

1. Initialization of L coding vectors.
2. First quantization of the training vectors is given by the equation 4.
3. "Re-training" of the coding vectors – estimation of the **centroids** of each cluster. A centroid (of a cluster) is defined as the vector mean value of all the training vectors in the cluster. At this point, the original coding vectors are replaced by the values of the corresponding centroids.
4. Encode the training vectors using the new code book.
5. Return to the step 3 if
   - the maximum number of iterations is not reached.
   - or the global distance change is still significant comparing to the previous iteration.

Initialization is an important step in the code book estimation. The following approaches are used:

1. Random initialization: L vectors are randomly chosen from the training vector set.
2. Initialization from a smaller coding book by splitting the chosen vectors. Each coding vector from the coding book of the size L is split into two new vectors in such a way that the original vector is "shifted" in two opposite directions. Thus the new generated code book is of the size $L' = 2L$. Using this approach, only the code books of the size $L = 2^g$ can be created, where g is an integer number. The first generation ($L = 1$) can be easily calculated as the vector mean value over **all** training vectors. This method is called after its founders, the Linde Buzo Gray (LBG) method.

8 VQ in this Lab

For the VQ implementation we dispose of the following functions:

- `vq_code.m` – encodes vectors using the given code book.
- `vq_clust.m` – calculates the new centroids using the training vectors and their alignment to the classes.
- `vq_split.m` – splits the code book.

Download the functions (from the web-page) and look at their help.

9 Data Preparation

As the training and the test data, the files `train.wav` and `testspdat.wav` are used, that contain longer speech segments. The file `train.wav` is used to train a code book and the file `testspdat.wav` is used for testing.

```
% get some training and test data
[A,G,Nram] = param ('train.wav',160,0,10);
[At,Gt,Nramt] = param ('testspdat.wav',160,0,10);
```

10 Generation of a One Vector Code Book, Visualization.

This is a "toy example", which will demonstrate usage of the function `vq*.m`. First, initialize a code book with **one** vector randomly chosen from the training vector set. Using this (one) vector "code" the data - all the indices will be obviously 1. Thus there is only one cluster of data (e.g. containing all the data).

```
% first let us do just one code-vector and visualize it.
% random selection of one vector
aux = randperm (Nram); theone = aux(1);
CB = A(:,theone);
[sym, gd] = vq_code(A,CB);
% look at the data and show the global distance
show (A,CB,sym); gd
```

```
% re-train,  re-code and show again
[CB, nbs]=vq_clust(A, sym, 1);
[sym, gd] = vq_code(A,CB);
show (A,CB,sym); gd
% how many vectors were attributed to different code vectors ?
nbs
```

The function `show.m` is used for displaying in 3D. 3 out of 10 coefficients (dimensions) have to be chosen, which is done in the function `show.m` at the line `i1=1; i2=2; i3=3;`. If you like to choose other coefficients than $1, 2, 3$, set simply different values to the variables. The calculated centroid is indicated by a big circle and the training vectors are indicated by small crosslet. Try different dimensions (different coefficients `i1, i2, i3`) and look carefully at what is done. Try zooming and rotation of the figure.

11 Code Book with 8 Vectors

Train a code book with $L = 8$ using random initialization. Plot the global distance, the centroids and the data clusters over the training iterations.

```
%% now with bigger CB - size 8
% init
aux = randperm (Nram); theones = aux(1:8);
CB8 = A(:,theones);
%% iterations: coding (+ visualization) and re-training
for iter=1:10
  [sym, gd] = vq_code(A,CB8);
  show (A,CB8,sym); gd
  pause
  [CB8, nbs]=vq_clust(A, sym, 8);
  nbs
  pause
end
```

Tasks

1. How many iterations are required (to achieve good estimation) to train a code book with 8 vectors?
2. How would you solve it automatically?
3. Try to train a larger code book ($L = 64$) with a random initialization. Is it feasible? how many vectors the clusters contain in the end of the training procedure (`nbs`) ?

12 Linde-Buzo-Gray

Train a code book with $L = 64$ using a gradual splitting: $1 \rightarrow 2 \rightarrow \ldots \rightarrow 64$. There is the function `vq_split.m` available for splitting. Everything is ready in the script `lbg.m`, that requires data in a matrix `A` (we have it already). The resulting code book is stored in the `CB`. Look at the help of `lbg.m` and then run it.

Tasks

1. What is the stop criterion (implemented in the example above) in training for each size of the code book?

13 Signal Synthesis using LPC Coefficients Encoded by VQ

The aim of VQ is reduction of the bit stream. Using a code book with $L = 64$, the bit streams of the LPC coefficients lowers from
`10 x 8 x 8 = 640` (10 double numbers x 8 byte per double x 8 bits per byte)
to 6 bit per frame.
The coefficients are encoded using `vq_code` to a sequence of indices, then according to these indices decoding is done by picking up the associated values from the code book. Run the process for a small (8) and a large (64) code book and listen to the result. Additionally, listen to the signal generated using the original LPC coefficients.

```
% synt. with original coeffs:
sst = syntnoise(At,Gt,10,160);
plot (sst); soundsc(sst);

% small codebook
[symt, gd] = vq_code(At, CB8);
gd
Atdecoded = CB8(:,symt);
sstdec = syntnoise(Atdecoded,Gt,10,160);
plot (sstdec); soundsc(sstdec);

% big codebook
[symt, gd] = vq_code(At, CB);
gd
Atdecoded = CB(:,symt);
sstdec = syntnoise(Atdecoded,Gt,10,160);
plot (sstdec); soundsc(sstdec);
```

Tasks

1. How exactly is the decoding done (converting the indices to the code vectors)? What the line `Atdecoded = CB(:,symt);` mean?
2. Can we measure the quality of the signal generated with the white noise excitation?
3. Do you obtain each time (for both code books) stable filters $\frac{1}{A(z)}$? How do you find out?
4. Why is the global distance larger than that obtained during training?