# LPC a vektorové kvantování

Jan Černocký, FIT VUT Brno

Při modelování tvorby řeči metodou LPC vycházíme z toho, že buzení prochází lineárním filtrem $H(z) = \frac{G}{A(z)}$, kde $A(z)$ je polynom $P$-tého řádu:

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

a $G$ je **gain** tohoto filtru.

Koeficienty filtru a jeho gain lze vypočítat z autokorelačních koeficientů. V Matlabu to za Vás udělá funkce `lpc`.

## 1 Signál a výpočet autokorelačních koeficientů

Nejprve bychom si měli něco načíst, ustřednit a rozdělit na rámce. Budeme používat rámce bez překrytí. Je připraven oblíbený testovací signál `test.wav`, ale můžete použít i cokoliv vlastního. Pro začátek si také vybereme jeden rámec na hraní.

```
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);
```

Prvním krokem pro LPC je výpočet autokorelačních koeficientů. Pro řád $P$ potřebujeme koeficienty $R[0] \ldots R[P]$. Následující ukazuje jejich výpočet názorně - spust'te jej, sledujte originální a posunutý signál a mačkejte `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 má na autokorelační koeficienty samozřejmě funkci, pojd'me srovnat náš výstup a výstup `xcorr`:

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

### Úkoly

1. Proč je nutné vybrat z výstupu `xcorr` pouze vzorky `N:(N+P)` ?
2. co znamená `'none'` ve volání `xcorr` ?

## 2 Výpočet LPC koeficientů

Pojďme nejprve "ručně" podle soustavy rovnic 19 ve slajdech. Soustavu vyřešíme pomocí Matlabovské inverze matice:

```
% 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
```

Na přednáškách byl ovšem presentován algoritmus pánů Levinsona a Durbina, který nikdo nechápal.... Tady je. Počítá postupně prediktory prvního až $P$tého řádu a ukládá je do sloupců matice `A`. Pro každý řád bude ve vektoru `E` k disposici energie chyby predikce. Pozor na Matlabovské indexování, pro vektory `E` a `R`, kde by se mělo indexovat od nuly, je nutné zvýšit pro Matlab index o jedničku.

```
% 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
```

Výsledné koeficienty jsou pak v posledním sloupci matice `A`. Matlab má také funkci `lpc`, která udělá vše za nás. Pojďme srovnat výsledek ručního výpočtu, "našeho" Levinsona Durbina a funkce `lpc` (která mimochodem také volá Levinsona Durbina):

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

### Úkoly

1. co dělá funkce `toeplitz` ?
2. zkuste si plotnout průběh energie chyby `plot(E)` a komentujte, zda jsme dobře zvolili řád prediktoru jako 10.

## 3 Hraní s chybovým signálem a jeho energií

Pro výsledný prediktor bychom rádi zjistili, jaká je energie chybového signálu $e[n]$. To můžeme udělat 4mi způsoby:

1. skutečně tento signál vygenerovat a vypočítat jeho energii:
2. podívat se na konec vektoru `E` a naší implementace LD.

3. spočítat ji podle rovnice 20 z přednášky.

4. použít Matlabovskou energii z funkce `lpc` - Matlab ji počítá jako normalizovanou, takže budeme muset násobit délkou rámce.

```
% 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 Spektra

Zkusíme se podívat na rozdíl mezi FFT a LPC spektrem (resp. spektrální hustotou výkonu) tohoto rámce. FFT výpočet už jsme dělali minule (natáhneme rámec na délku 1024 pomocí nul):

```
% 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);
```

Do stejného obrázku teď pojd'me plotnout odhad spektrální hustoty výkonu pomocí LPC, který pro nás zařídí oblíbená funkce `freqz`:

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

kde $f$ je normalizovaná frekvence $f = F/F_s$.

```
% 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;
```

Nakonec bude zajímavé zobrazit si FFT spektrum chyby ("residual") 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);
```

### Úkoly

1. Co representuje LPC-spektrum: buzení, artikulační trakt nebo obojí ?

2. Zkuste omezit řád prediktoru jen na 2 (`lpc(x,2)` a plotnout si LPC spektrum. Co vidíte ?

3. Proč se filtru $A(z)$ říká "bělicí" (whitening) ?

4. je skutečně signál $e[n]$ šumem, kde mají být vzroky nekorelované ? Které závisosti jsme oproti $x[n]$ odstranili - krátkodobé nebo dlouhodobé ?

# 5 LPC všech rámců a co s nimi

Jeden rámec byl pouze na hraní, běžně (kódování, rozpoznávání, ...) musíme provést LPC analýzu všech rámců. LPC koeficienty (bez $a_0 = 1$) uložíme do jedné velké matice (pro každý rámec do jednoho sloupce), všechny gainy filtrů do řádkového vektoru.

```
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))';         % discrading a0 = 1
  Aall(:,n) = a;
  Gall(n) = sqrt(e);
end
```

První aplikací, kterou si zkusíme, je výpočet chybového signálu (residual) pro celý signál. V jednotlivých rámcích filtrujeme filtrem $A(z)$ s příslušnými koeficienty, dáme si pozor na to, aby se používaly počáteční podmínky filtru z minulého rámce (viz `help filter`). Signál si zobrazíme (zkuste i zoom) a poslechneme.

```
% 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);
```

Zkusíme dále syntézu pomocí filtrů $\frac{1}{A(z)}$, v každém rámci budeme budit bílým Gaussovským šumem.

```
% 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);
```

a konečně zkusme legrácku: v souboru `violinák.wav`[1] je zvuk houslí, pojd'me jej zkusit použít jako buzení pro syntézu:

[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);
```

### Úkoly

1. Rozumíte residuálu celého signálu ?
2. Při syntéze (buzení bílým šumem i houlsemi) je nutné, aby budicí signál měl jednotkovou energii (neboli výkon). Ověřte, zda to platí - odkomentujte `sum(excit .^ 2) / 160`.
3. při buzení houslemi může být u hlásky 's' problém s vyššími frekvencemi, které houslový zvuk neobsahuje. Zkuste k housličkám přimíchat trochu bílého šumu (stačí odkomentovat kousek na lice `excit = vr (:,n);`).

## 6 Parametrizace a syntéza ve funkcích

pro další hraní uzavřeme výpočet koeficientů a gainů pro celý signál: `param`. Existuje také funkce pro syntézu: `syntnoise`. Obě funkce vyzkoušíme na souboru `train.wav` (více řeči než v "létajícím praseti"), který budeme dále využívat pro vektorové kvantování.

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

# Vektorové kvantování

## 7 Úvod k VQ

Vektorové kvantování (Vector Quantization – VQ) se ve zpracování řeči používá např. při redukci bitového toku v kódování či při práci s diskrétními HMM. Spočívá v tom, že $P$-rozměrné vektory parametrů $\mathbf{x}$ převedeme na **symboly** pomocí **kódové knihy** o $L$ kódových vektorech:

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

Při VQ-**kvantování** přiřadíme vektoru $\mathbf{x}$ kódový vektor $\mathbf{y}_i$ s minimální vzdáleností:

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

Vzdálenost $d(\cdot, \cdot)$ může být libovolná, budeme používat kvadratickou míru, což není nic jiného než Eukleidova vzdálenost bez odmocniny. Dá se zapsat i jako skalární součin vektoru $(\mathbf{a} - \mathbf{b})$ a jeho transponované varianty (řádkový vektor krát sloupcový vektor rovná se skalár):

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

**Globální vzdálenost** je dána součtem všech minimálních vzdáleností, normujeme počtem kvantovaných vektorů. Vektory, které jsou kvantovány jedním kódovým vektorem, tvoří tzv. **cluster**.
Kódovou knihu a-priori neznáme a musíme ji natrénovat na souboru trénovacích vektorů. Používáme následující postup:

1. Inicialisace $L$ kódových vektorů.
2. První kvantování trénovacích vektorů podle rovnice 4.
3. "Přeučení" kódových vektorů – výpočet **centroidů** jednotlivých clusterů. Centroidy definovány jako vektorová střední hodnota všech trénovacích vektorů v clusteru. Centroidy nahradí původní kódové vektory.
4. Kvantování trénovacích vektorů novou kódovou knihou.
5. Návrat ke kroku 3. pokud
   - jsme nedosáhli maximálního povoleného počtu iterací.
   - nebo se globální vzdálenost ještě podstatně mění.

**Inicialisace** je důležitým krokem v učení kódové knihy. Používají se tyto postupy:

1. Náhodná inicialisace: ze souboru trénovacích vektorů se náhodně vybere $L$ vektorů.
2. Inicialisace z menší kódové knihy "štěpením" kódových vektorů. Každý kódový vektor knihy o velikosti $L$ se rozdělí na dva nové tak, že se původní kódový vektor "posune" ve dvou opačných směrech. Vznikne tak nová kódová kniha o velikosti $2L$. Tímto způsobem se dají vytvářet pouze knihy o $L = 2^g$ vektorech, kde $g$ je celé číslo. První generace ($L = 1$) se dá jednoduše spočítat jako vektorová střední hodnota **všech** trénovacích vektorů. Tato metoda se nazývá podle svých tvůrců Linde Buzo Gray (LBG).

## 8 VQ v této labině

Pro práci s VQ máme na k disposici tyto funkce:

- `vq_code.m` – kóduje vektory pomocí zadané kódové knihy.
- `vq_clust.m` – z trénovacích vektorů a příslušnosti ke clusterům spočítá nové centroidy.
- `vq_split.m` – provádí štěpení kódové knihy.

Okopírujte si je a prostudujte jejich helpy.

# 9 Příprava dat

Jako trénovací a tstovací data budeme používat soubory `train.wav` a `testspdat.wav`, kde je více řeči než létajícím praseti. Na `train.wav` budeme trénovat kódovou knihu a na `testspdat.wav` budeme testovat.

```
% 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 Tvorba kódové knihy s jedním vektorem, visualisace

Tento příklad je "toy example", na kterém si ukážeme použití funkcí `vq*.m` Nejprve inicialisujeme kódovou knihu s **jedním** vektorem tak, že jej náhodně vybereme z trénovacích vektorů. Pak tímto vektorem "zakódujeme" data - všechny indexy budou samozřejmě '1'. Existuje pak samozřejmě jen jeden cluster dat (neboli všechna data).

```
% first let us do just one codevector 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 atributed to different code vectors ?
nbs
```

Pro zobrazení je nachystána funkce funkce `show.m`, která zobrazuje ve 3D. Z 10ti koeficientů je potřeba vybrat 3 pro zobrazení, toto je ve funkci `show.m` na lince `i1=1; i2=2; i3=3;`. Pokud máte zájem o jiné koeficienty než $a_1, a_2, a_3$, dáte do proměnných prostě jiné indexy než 1,2,3. Centroid je označen velkým kolečkem, trénovací vektory jsou malé křížky. Pohrejte si s koeficienty, na které se "díváme". Zkuste různé zoomování a rotování obrázku.

# 11 Kódová kniha s 8mi vektory

Natrénujeme kódovou knihu s $L = 8$ a s náhodnou inicialisací. Zobrazíme globální vzdálenost, centroidy a clustery dat během trénovacích iterací.

```
%% 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
```

### Úkoly

1. Po kolika iteracích by bylo vhodné trénování kódové knihy s 8mi vektory zastavit ?
2. Jak byste toto řešili automaticky ?

3. Vyzkoušejte trénování větší kódové knihy ($L = 64$) s náhodnou inicialisací. Jde to ? Jaké jsou počty vektor; v jednotlivých clusterech na konci (`nbs`) ?

# 12 Linde-Buzo-Gray

Natrénujte kódovou knihu s $L = 64$ postupným štěpením: $1 \rightarrow 2 \rightarrow \ldots \rightarrow 64$. Pro štěpení kódové knihy je k disposici funkce `vq_split.m`. Vše je hotovo ve skriptíku `lbg.m`, který očekává data v matici `A` (tu bychom měli mít). Výsledná kódová kniha bude v `CB`. Prostudujte vnitřnosti `lbg.m` a pak jej pust'te.

### Úkoly

1. Jak je řešeno zastavení trénování pro každou velikost kódové knihy ?

# 13 Syntéza signálů s LPC koeficienty kódovanými pomocí VQ

Cílem VQ je snížení bitového toku. Pokud použijeme kódovou knihu o velikosti 64, snížíme bitový tok na LPC koeficienty z
$10x8x8 = 640$ (10 double čísel x 8 byte na double x 8 bitů na byte)
na 6 bitů na jeden rámec.
Koeficienty kódujeme pomocí `vq_code` na posloupnost indexů, pak podle těchto indexů dekódujeme výběrem z kódové knihy. Proces spustíme pro malou (8) a velkou (64) kódovou knihu a poslechneme si výsledek. Bude dobré poslechnout si také signál syntetizovaný s originálními (nekódovanými) LPC koeficienty.

```
% 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);
```

### Úkoly

1. jak přesně probíhá dekódování (tedy převod indexů na kódové vektory) ? Rozumíte lince `Atdecoded = CB(:,symt);` ?
2. Dá se u signálu, který je buzený pouze bílým šumem, posoudit jeho kvalita ?
3. Dostáváte pro obě kódové knihy pokaždé stabilní filtry $\frac{1}{A(z)}$ ? Jak to poznáte ?
4. Proč je globální vzdálenost větší než u trénování ?