Linear Prediction


From the previous lecture: Speech production (vowels)

Voiced glottal excitation
Vocal tract response
Produced speech sound

From the previous lecture: Kelly-Lochbaum equations

Model the vocal tract using simple uniform tube sections:

The lattice-structured filter resulting from the Kelly-Lochbaum equations is an all-pole filter, meaning that its transfer function

\[ H(z) = \frac{G}{A(z)} \]

includes only poles (≠ zeros of the denominator \(A(z)\))
Linear system models

Linear models can be categorized as one of the following (in speech processing, input $x(n)$ refers to glottis excitation and output $y(n)$ to measured speech):

- **Autoregressive moving average model (ARMA)**
  \[ \hat{y}(n) = -\sum_{k} a(k) y(n-k) + \sum_{k} b(k) x(n-k) \]
  \[ H(z)_{ARMA} = \frac{B(z)}{A(z)} \]
  Corresponds to a **generic linear recursive filter**

- **Moving average model (MA)**
  \[ \hat{y}(n) = \sum_{k} b(k) x(n-k) \]
  \[ H(z)_{MA} = B(z) \]
  Corresponds to a **FIR filter**

- **Autoregressive model (AR)**
  \[ \hat{y}(n) = g x(n) - \sum_{k} a(k) y(n-k) \]
  \[ H(z)_{AR} = \frac{g}{A(z)} \]
  Corresponds to an **all-pole filter** (gain $g$ is constant)

AR model is a good choice for modeling the vocal tract

Typically the AR model is used in speech processing, because

- The above-described lattice-structured model for the vocal tract corresponds to an **all-pole filter** (AR model). In other words, the vocal tract is (with certain assumptions) theoretically an all-pole filter

- The input signal $x(n)$ is not known (glottal excitation).

- AR model parameters $a(k)$ can be computed efficiently. (Linear predictive analysis using the Levinson-Durbin algorithm)

- A higher-order AR model can (to some extent) represent also the more generic ARMA model

From AR model to linear prediction

- Linear prediction analysis allows us to estimate the AR model parameters from an input signal
  
  - **Linear prediction is a good method for estimating the parameters of the vocal tract**
  
  - **Linear prediction is one of the most important tools in speech processing**

- Acronyms: LP (**linear prediction**), LP-analysis, LPC (**linear predictive coding**)

- From the speech processing viewpoint, the most important property of LP is its ability to **model the vocal tract**

Linear prediction analysis

- LP analysis finds the filter coefficients ($a_0, a_1, a_2, \ldots, a_p$) that best predict the signal samples according to the AR model (discarding $x(n)$ term):
  \[ y(n) = g x(n) - \sum_{k=1}^{p} a_k y(n-k) \quad \Rightarrow \quad \hat{y}(n) = -\sum_{k=1}^{p} a_k y(n-k) \]

- The coefficients of the predictive filter are chosen so that the squared prediction error is minimized in the analysis window:
  \[ \hat{\sigma} = \arg \min_{\sigma} \sum (y(n) - \hat{y}(n))^2 \]

- Keep in mind that **speech is processed in short frames** and LP analysis is done approx. every 10-30 ms in partly overlapping frames
Example: Magnitude spectrum and LP spectrum of a vowel

Window length in seconds: 480/16kHz = 30ms

Estimating parameters $a(1), a(2), \ldots, a(p)$

A necessary condition for the optimality of coefficient $a(i)$ is that the partial derivative of function $E_p$ with respect to $a(i)$ is zero.

Partial derivative of $E_p$ with respect to variable $a(i)$ ($i = 1, 2, \ldots, p$) is

$$\frac{\partial E_p}{\partial a(i)} = \frac{\partial}{\partial a(i)} \left( \sum_{k=0}^{p} a(k)s(n-k) \right)^2$$

$$= \sum_{n} 2 \left( \sum_{k=0}^{p} a(k)s(n-k) \right) \frac{\partial}{\partial a(i)} \left( \sum_{k=0}^{p} a(k)s(n-k) \right)$$

$$= 2 \sum_{n} \left( \sum_{k=0}^{p} a(k)s(n-k) \right) s(n-i) = 0$$

Estimating parameters $a(1), a(2), \ldots, a(p)$  \hspace{1cm} (1)

Since we model speech signals here, let’s use $s(n)$ to denote the measured speech signal, instead of $\gamma(n)$ above.

Using prediction coefficients $a(1), a(2), \ldots, a(p)$ (length of the prediction filter is $p$) the energy of the prediction error is

$$E_p = \sum_{n} (s(n) - \hat{s}(n))^2 = \sum_{n} \left( s(n) - \sum_{k=0}^{p} a_k s(n-k) \right)^2$$

By defining $a(0) = 1$, the error energy can be written conveniently as

$$E_p = \sum_{n=0}^{\infty} \left( \sum_{k=0}^{p} a_k s(n-k) \right)^2$$

Autocorrelation function

Let’s represent the above derivative of the error function using the autocorrelation:

$$\sum_{n=0}^{\infty} 2 \left( \sum_{k=0}^{p} a(k)s(n-k) \right) s(n-i)$$

$$= 2 \sum_{k=0}^{p} a(k) \sum_{n=0}^{\infty} s(n-k)s(n-i)$$

$$= 2 \sum_{k=0}^{p} a(k) r(k,i)$$

where

$$r(k,i) = \sum_{n=0}^{\infty} s(n-k)s(n-i) = r(k-i)$$

Histogram data correlation function

$r(-k) = r(k)$
Autocorrelation equations

Let's get back to the result of the derivation (Derivative of the error with respect to coefficients \(a(1), a(2), ..., a(p)\)):

\[
\frac{\partial E_p}{\partial a(i)} = 2 \sum_{k=1}^{p} a_k r(k - i) = 0
\]

The zeros of the derivatives of the prediction error energy are obtained as:

\[
\begin{align*}
\sum_{i=0}^{p} a_i r(k - 1) &= 0 \\
\sum_{i=0}^{p} a_i r(k - 2) &= 0 \\
&\vdots \\
\sum_{i=0}^{p} a_i r(k - p) &= 0
\end{align*}
\]

\[
\begin{bmatrix}
    a(0) = 1 \\
    r(a) = r(n)
\end{bmatrix}
\]

\[
\begin{bmatrix}
    r(1) & r(0) & r(1) & \cdots & r(p - 1) \\
    r(2) & r(1) & r(0) & \cdots & r(p - 2) \\
    r(3) & r(2) & r(1) & \cdots & r(p - 3) \\
    \vdots & \vdots & \vdots & \ddots & \vdots \\
    r(p) & r(p - 1) & r(p - 2) & \cdots & r(0)
\end{bmatrix}
\]

\[
\begin{bmatrix}
    a_1 \\
    a_2 \\
    a_3 \\
    \vdots \\
    a_p
\end{bmatrix}
\]

where the coefficient matrix \(R\) is:

\* symmetric, because \(r(k, i) = r(k - i)\)
\* Toeplitz (= diagonal-constant matrix), because \(r(k, i) = r(k - i)\)

That is essential when looking for an efficient solving algorithm.

\[E_p \quad ???\]

Let's prove that, with optimal coefficients \(a\),

\[
E_p = \sum_{i=0}^{p} a_i r(i) = \sum\left(s(n) - \hat{s}(n)\right)^2
\]

The proof is obtained by expanding the square and rearranging terms:

\[
\begin{align*}
&= \sum\left(s(n)^2 - 2s(n)\hat{s}(n) + \hat{s}(n)^2\right) \\
&= \sum s(n)^2 - 2\sum a_i s(n)i(n) + \sum a_i \hat{s}(n)i(n) \\
&= r(0) + 2\sum a_i r(i) + \sum a_i s(n - i) \sum a_i \hat{s}(n - k) \\
&= r(0) + 2\sum a_i \sum a_i (-a_i) r(k - i) + \sum a_i \sum a_i a_i \sum s(n - i) s(n - k)
\end{align*}
\]

\[E_p \quad ??? \quad (2)\]

and further:

\[
\begin{align*}
&= r(0) - 2\sum\sum a_i a_i r(k - k) + \sum a_i \sum a_i a_i r(k - k) \\
&= r(0) + \sum a_i a_i r(k - i) \\
&= r(0) + \sum a_i a_i r(i) \\
&= \sum a_i a_i r(i)
\end{align*}
\]

So we got that:

\[
\sum_{i=0}^{p} a_i r(i) = \sum_n \left(s(n) - \hat{s}(n)\right)^2 = E_p
\]
Levinson-Durbin recursive algorithm

Prediction coefficients $a(1), a(2), \ldots, a(p)$ could be solved directly from the above-derived equations using for example the matrix inverse $R^{-1}$. However that is computationally expensive. An efficient algorithm for solving a Toeplitz-type matrix equation is Levinson-Durbin recursive algorithm

The idea is to solve the matrix equation $R \mathbf{x} = \mathbf{y}$ in blocks by increasing the size of matrix $R$ and the length of vector $x$ and by computing a new solution based on the previous one.

Levinson-Durbin recursion

$p=0$: $\begin{bmatrix} r(0) \\ r(l) \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ a(l) & 1 \end{bmatrix} \begin{bmatrix} E_0 \\ 0 \end{bmatrix}$

$p=1$: $\begin{bmatrix} r(0) \\ r(l) \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} + k_1 \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} E_0 \\ r(l) \end{bmatrix}$

Levinson-Durbin recursion

$p=0$: $\begin{bmatrix} r(0) \\ r(l) \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ a(l) & 1 \end{bmatrix} \begin{bmatrix} E_0 \\ 0 \end{bmatrix}$

$p=1$: $\begin{bmatrix} r(0) \\ r(l) \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} + k_1 \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} E_0 \\ r(l) \end{bmatrix}$

$E_0 = r(0) = \sum s(n)^2$
Levinson-Durbin recursion

\[ E_0 = r(0) = \sum s(n)^2 \]

**p=0:**

\[
\begin{bmatrix}
 r(0) \\
 r(1)
\end{bmatrix} =
\begin{bmatrix}
 [E_0] \\
 0
\end{bmatrix},
\]

\[ k_1 = - \frac{r(1)}{E_0} \]

\[ E_i = E_0 + k_1 r(1) \]

\[ a_i(1) = k_1 \]

**p=1:**

\[
\begin{bmatrix}
 r(0) & r(1) \\
 r(1) & r(0)
\end{bmatrix} =
\begin{bmatrix}
 [E_1] \\
 0
\end{bmatrix},
\]

\[ k_1 = \frac{-r(1)}{E_0} \frac{1}{a_1(1)} \]

\[ E_i = E_0 + k_1 r(1) \]

\[ a_i(1) = \frac{k_1}{a_1(1)} \]

\[ q_2 = r(2) + r(1) a_1(1) \]

\[ E_2 = E_1 + k_1 q_2 \]

**p=2:**

\[
\begin{bmatrix}
 r(0) & r(1) & r(2) \\
 r(1) & r(0) & r(1) \\
 r(2) & r(1) & r(0)
\end{bmatrix} =
\begin{bmatrix}
 [E_2] \\
 0 \\
 0
\end{bmatrix},
\]

\[ k_1 = \frac{-q_2}{E_1} \]

\[ E_i = E_0 + k_1 r(1) \]

\[ a_i(1) = \frac{k_1}{a_1(1)} \]

\[ q_3 = r(3) + r(2) a_1(1) + r(1) a_2(1) \]

\[ E_3 = E_2 + k_1 q_3 \]

Notes about the Levinson-Durbin algorithm

- The variables of the Levinson-Durbin algorithm are the reflection coefficients of the corresponding acoustic tube model!

- The energy of the error decreases when more coefficients are added to the prediction filter:

\[ E_{p+1} \leq E_p \]

**proof:**

\[ E_{p+1} = E_p + k_p q_{p+1} = E_p - \frac{q_{p+1}}{E_p} q_{p+1} = E_p - \frac{q_{p+1}^2}{E_p} \]

because both \( E_p \) and \( q^2 \) are always positive, thus \( E_{p+1} \leq E_p \)

Choosing the model order (1)

- There is usually one formant per kHz

- Model order \( p \) can be estimated as the sampling rate in kHz

For example:

- sampling rate 8kHz → model order 8
- sampling rate 16kHz → model order 16

- However to compensate for model inaccuracies, usually a slightly higher model order is selected

For example:

- sampling rate 8kHz → model order 10 or 12
- sampling rate 16kHz → model order 18 or 20
Choosing the model order (2)

- Example:

Let's take a short segment of phone /y/ (sampling rate 16kHz) and window it using a smooth window function.

Next, let's solve the prediction coefficients for models of varying orders and investigate the frequency responses of the obtained LP filters.

Choosing the model order (3)

At sampling rate 16kHz, a good choice for model order would be 18. What happens if too high or low model order is chosen?

Pre-emphasis of high frequencies

- As can be seen in the above figure, speech spectrum has much less energy at high frequencies than at low frequencies. That may have the consequence that LP analysis will not find any of the higher formants.
- To address the issue, usually a pre-emphasis filter is used that flattens the spectral tilt before the LP analysis:

$$H_{\text{pre-exp}}(z) = 1 - b_1 z^{-1}$$

where usually: $0.95 < b_1 < 0.99$

Pre-emphasis of high frequencies: an example

Original spectrum

Pre-emphasized spectrum
Where is LP analysis used?

Speech coding: enables the separate coding of excitation and vocal tract parameters

Speech recognition: provides information about the speech spectrum (and therefore about the phoneme identity)

Speech synthesis: allows separate control of the excitation and vocal tract parameters

In MATLAB, LP analysis can be done with the command `lpc`