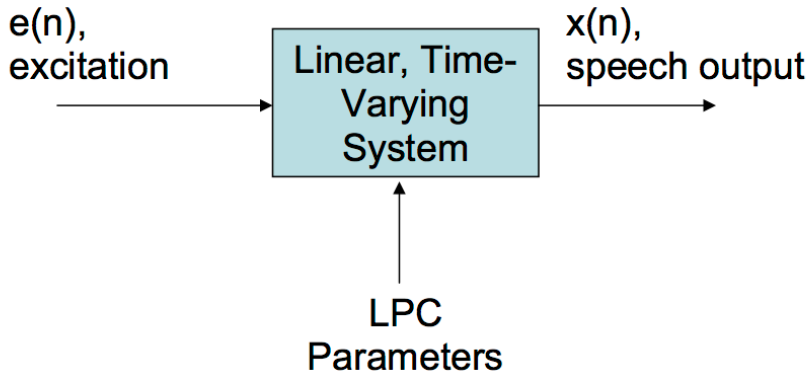
LP Synthesis with a Binary source

1.0 Introduction

The goal of this project is to build an LPC Vocoder based on the model of speech production shown below:



The excitation or input, *e(n)*, is assumed to be either white noise or a quasi-periodic train of impulses (for voiced sounds). The linear system is assumed to be slowly time-varying, such that over short time intervals it can be described by the all-pole system function:

$$H(z) = \frac{G}{1 - \sum_{k=1}^{p} a_k z^{-k}}$$

The input and output signals are related by a difference equation of the form:

$$x(n) = \sum_{k=1}^{p} a_k x(n-k) + Ge(n)$$

Using standard methods of linear predictive (LP) analysis, we can find the set of prediction coefficients $\{\alpha_k\}$ that minimize the mean-squared prediction error betweeh the signal *x(n)* and a predicted signal based on a linear combination of past samples; i.e.,

$$< (d(n))^2 >=< (x(n) - \sum_{k=1}^{p} \alpha_k x[n-k])^2 >$$

where < > represents averaging over a finite range of values of *n*. It can be shown that using one method of averaging, called the *autocorrelation method*, the optimum predictor coefficients $\{\alpha_k\}$ satisfy a set of linear equations of the form:

$$\mathbf{R\,a = r}$$

where **R** is a *p x p* Toeplitz matrix made up of values of the autocorrelation sequence for *x(n)*, **a** is a *p x 1* vector of prediction coefficients, and **r** is a *p x 1* vector of autocorrelation values.

In using LP techniques for speech analysis and synthesis, we make the assumption that the predictor coefficients $\{\alpha_k\}$ are identical to the parameters $\{a_k\}$ of the speech model. Then, by definition of the model, we see that the output of the prediction error filter with system function:

$$A(z) = 1 - \sum_{k=1}^{p} \alpha_k z^{-k}$$

is

$$d(n) = x(n) - \sum_{k=1}^{p} a_k x[n-k] \equiv G \cdot e(n)$$

i.e., the excitation of the model is defined to be the input that produces the given output, *x(n),* for the prediction coefficients estimated from *x(n)*. The gain constant, *G,* is therefore simply the constant that is required so that *e(n)* has unit mean-squared value, and is readily found from the autocorrelation values used in the computation of the prediction coefficients.

It can be shown that because of the special properties of the LP equations, an efficient methods, called the Levinson Recursion, exists for solving the equations for the predictor parameters. MATLAB's Signal Processing Toolbox has a function called **lpc()**, but to avoid requiring that toolbox, it is also possible to use the general MATLAB matrix functions. Specifically, the following code from an M-file **autolpc()** implements the autocorrelation method of LP analysis:

```
%AUTOLPC
%      [A,G,a,r]=autolpc(x,p)
```

```
%               x=vector of input samples
%               p=lpc model order
%               a=vector of lpc coefficients
%               r=vector of autocorrelation coefficients
%               G=rms prediction error
%               A=prediction error filter
%               see also ATOK, KTOA, RTOA

L=length(x);
r=[];
for i=0:p
        r=[r; sum(x(1:L-i).*x(1+i:L))];
end
R=toeplitz(r(1:p));
a=inv(R)*r(2:p+1);
A=[1; -a];
G=sqrt(sum(A.*r));
```

2.0 Project Details

For this project you will use the file **s5.wav**, which contains the utterance "Oak is strong and also gives shade". You will also use the pitch period file **pp5.mat** which contains estimates of the pitch period of the speech, every 10 msec. (Note that unvoiced speech or silence is denoted by a pitch period value of 0).

The goal of this project is to develop a program for doing linear predictive speech analysis and synthesis according to the model above. Specifically you are asked to write a program to estimate the prediction parameters and the gain, on a frame-by-frame basis for an input speech signal vector. Store this date in a two dimensional arrayor MATLAB data structure. The file **pp5.mat** contains measurements (estimates) of voiced/unvoiced decision and pitch period at a frame rate of 100 frames/sec. Since the sampling rate of the speech signal is 8 kHz, the frame spacing is 80 samples. A zero value indicates that the speech frame is unvoiced (or silence), and a non-zero value is the pitch period estimate (in samples at an 8 kHz sampling rate) when the frame is voiced. Using the predictor data, together with the pitch period data provided, the task is to synthesize a synthetic version of sentence **s5.wav** (save your resulting synthesis in a file called **s5_synthetic.wav**).

In the course of creating this "LPC Vocoder", you should think about the following issues and how they affect the quality of your synthesis:

1. Note that what goes on in a given frame is ***not*** independent of what happened in previous frames. For example, as the pitch period changes, you will need to know where the last pitch impulse occurred in the previous frame so as to determine the location of the next impulse in the current frame.

2. You can change the vocal tract filter only once per frame, or you can interpolate between frames and change it more often (e.g., at each new pitch period). What parameters can you interpolate and still maintain stability of the resulting synthesis?
3. You don't have to quantize the parameters.
4. Listen to your synthetic speech and see if you can isolate what are the main sources of distortion.
5. Implement a "debug mode" where you can show the "true excitation signal" (i.e., the residual LPC analysis error, *e(n)*) and the synthetic excitation signal that you are using, as well as the resulting synthetic speech signal and the original speech signal, for any frame or group of frames of speech. Using this debug mode, see if you can refine your estimates of the key sources of distortion in the LPC Vocoder.

Sanity check: using e(n) as excitation => perfect reconstruction

6. Computing G: match energy at the output.
7. Overlap & Add:
   - Current frame + "ringing" of previous frame.
8. Looking to Final Project – quantized LPC system:
   - Use non-overlapping frames
   - Gain: a constant per frame
   - Phase continuity when stitching two V frames
   - Rosenberg global-pulse shape

Notes (summary of last page)

- what goes on in a given frame is **not** independent of what happened in previous frame
  - e.g., position of pitch pulses near stitching boundary (next)
- vocal-tract parameters
  - interpolation of overlapping frames
- no quantization
- location of distortions by listening
- sanity check
  - using residual signal e(n) should result in perfect reconstruction
- computing G: match entry at output
- ~~overlap & add~~
  - ~~current frame + "ringing" of previous frame~~
- an eye on Final Project:
  - use non-overlapping frames
  - Gain = constant per frame
  - Phase continuity when stitching two V frames (next)
  - Rosenberg global-pulse shape

What to Deliver?
(an example)

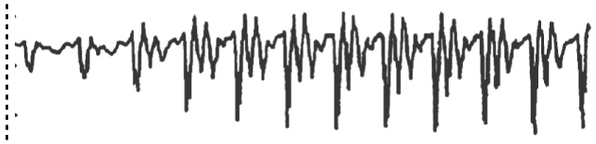
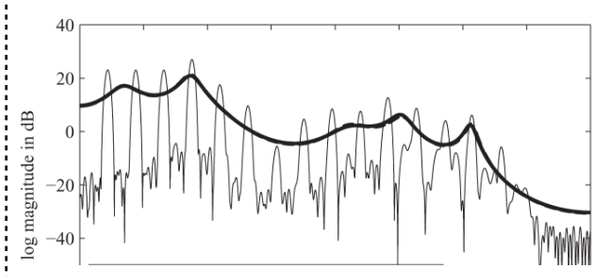
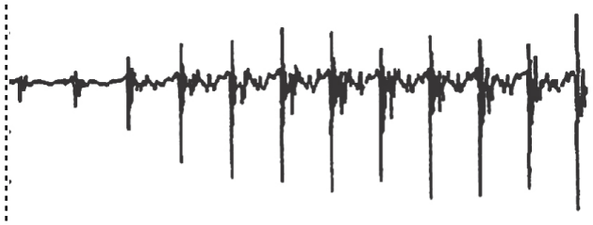
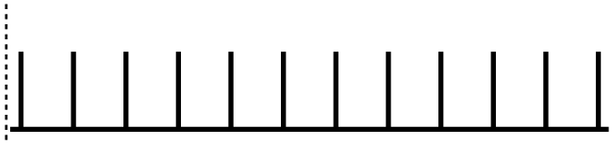
- Report (PDF, 5-page long max)
  1. introduction - half a page
  2. system description
     1. A block diagram
     2. Window size = 30ms; non-overlapping frames
     3. Prediction order, *p = 4, 10, 18*
  3. illustration figures (next page)
  4. performance
     1. MOS
     2. whole sentence SNR (*wSNR*)
     3. segmental SNR (*sSNR*)
- Two synthetic speech files and MATLAB scripts
  - your system using the residual per frame as input
  - your system using a binary source as input
  - (all in a zipped directory, separate from report)
- Due date – Thursday, April 12, morning

figures

waveform	input *s(n)*	input *s(n)*	input *s(n)*
power spectrum & overlays	$\lvert H \rvert$ $p > p_{opt}$	$\lvert H \rvert$ p_{opt}	$\lvert H \rvert$ $p < p_{opt}$
waveform	residual *e(n)*	residual *e(n)*	residual *e(n)*
waveform	binary source *u(n)*	binary source *u(n)*	binary source *u(n)*
waveform	binary source *G·u(n)*	binary source *G·u(n)*	binary source *G·u(n)*
waveform	*s'(n) for e*(n) as input	*s'(n) for e*(n) as input	*s'(n) for e*(n) as input
waveform	*s'(n) for G·u(n)* as input	*s'(n) for G·u(n)* as input	*s'(n) for G·u(n)* as input

Steps to produce the figures:

1. derive the residual e(n) for the entire file
2. replace e(n) with u(n) (for the entire file)
   - u(n) is the binary excitation without G
3. create G·u(n) (for the entire file)
4. create s'(n) with e(n) as excitation
5. create s'(n) with G·u(n) as excitation
6. cut a 60-ms long vowel window of the original s(n) – create plot matrix
   - find the power spectrum and $|H|$ for this window
   - cut all corresponding windows from the waveforms created in steps 1 to 5
7. Repeat step 6 for a 60-ms long UNV window
8. cut a 150-ms Vowel-Fricative-Vowel segment – create plot matrix
   - no need for power spectrum and $|H|$ here
9. Repeat for p = 4, 10, 18

voiced			
same for unvoiced	$\|H\|$ $p > p_{opt}$		$\|H\|$ $p < p_{opt}$
	residual *e(n)*		residual *e(n)*
	binary source *u(n)*		binary source *u(n)*
	binary source *G·u(n)*		binary source *G·u(n)*
	s'(n), for *e(n)* as input	*s'(n)*, for *e(n)* as input 	*s'(n)*, for *e(n)* as input
	s'(n), for *G·u(n)* as input	*s'(n)*, for *G·u(n)* as input 	*s'(n)*, for *G·u(n)* as input