1 LPC modeling of vocal tract

LPC (Linear Predictor Coding) is a method to represent and analyze human speech. The idea of coding human speech is to change the representation of the speech. Representation when using LPC is defined with LPC coefficients and an errorsignal, instead of the original speech signal. The LPC coefficients are found by LPC estimation which describes the inverse transfer function of the human vocal tract.

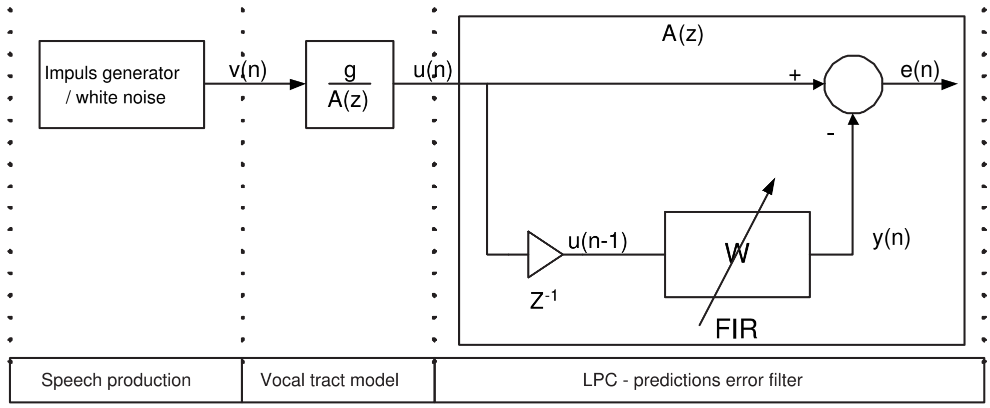


Figure 1.1: Relationship between vocal tract model and LPC model.

The above figure 1.1 shows the relationship between vocal tract transfer function and the LPC transfer function. Left part of the figure shows speech production model, while right-hand side of figure shows LPC prediction error filter (LPC analysis filter) applied to output of the vocal tract model. Vocal transfer function and LPC transfer function are defined as follow:

$$H(z) = \frac{g}{A(z)} = \frac{g}{1 + \sum_{k=1}^{n} a_k z^{-k}} \tag{1.1}$$

$$A(z) = 1 + \sum_{k=1}^{n} a_k z^{-k} \quad a_k = \begin{cases} 1 & k = 0 \\ -w_k & k = 1, 2, .., M \end{cases} \tag{1.2}$$

The method to obtain the LPC coefficients included in the equation 1.2 is calculated using LPC estimation. This method is described in next section. LPC analysis and LPC synthesis is also describe in later sections, which has application in bandwidth expansion.

1.1 LPC-estimation

LPC estimation is used to constructing the LPC coefficients for the inverse transfer function of the vocal tract. The standard methods for LPC coefficients estimation have the assumption that the input signal is stationery. Quasi stationery signal is obtain by framing the input signal which is often done in frames in length of 20 ms. A more stationery signal result in a better LPC estimation because the signal is better described by the LPC coefficients and therefore minimize the residual signal. The residual signal also called the error signal which is described in next section.

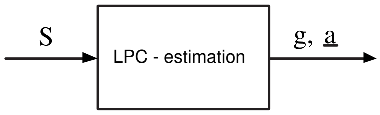


Figure 1.2: LPC-estimation blockdiagram

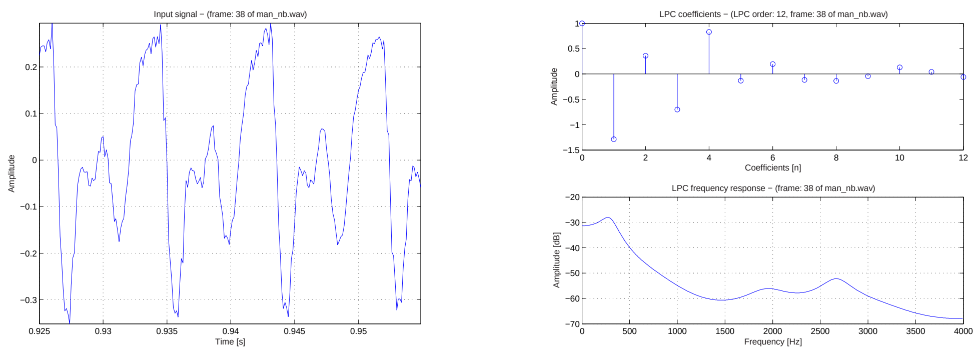
Figure 1.2 show a block diagram of a LPC estimation, where S is the input signal, g is the gain of the residual signal (prediction error signal) and a is a vector containing the LPC coefficients to a specific order. The size of vector depends on the order of the LPC estimation. Bigger order means more LPC coefficients and therefore better estimation of the vocal tract.

Matlab LPC estimation:

```
[autos,lags] = xcorr(s);
autos = autos(find(lags==0):end);
[a,g] = levinson(autos,N);

%autos: Autocorrelation of input signal [vector]
%s: Input signal [vector]
%a: LPC coefficients
%g: Prediction error variance
```

The above Matlab code calculate a and g from a given input signal S.



(a) Input signal (b) LPC coefficients and its frequency response

Figure 1.3: LPC estimation

Figure 1.3(a) show the input signal and figure 1.3(b) show the LPC coefficients and the frequency response of the LPC coefficients, which is found using above Matlab code.

1.2 LPC-analysis

LPC analysis calculates an error signal from the LPC coefficients from LPC estimation. This error signal is called the residual signal which could not be modeled by the LPC estimator. It is possible to calculate this residual signal by filtering the original signal with the inverse transfer function from LPC estimation. If the inverse transfer function from LPC estimation is equal to the vocal tract transfer function then is the residual signal from the LPC analysis equal to the residual signal which is put in to the vocal tract. In that case is the residual signal equal to the impulses or noise from the human speech production (Se illustration 1.1).

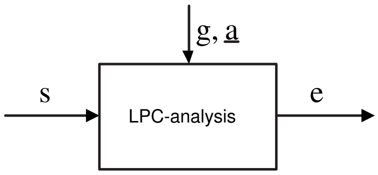


Figure 1.4: LPC-analysis.

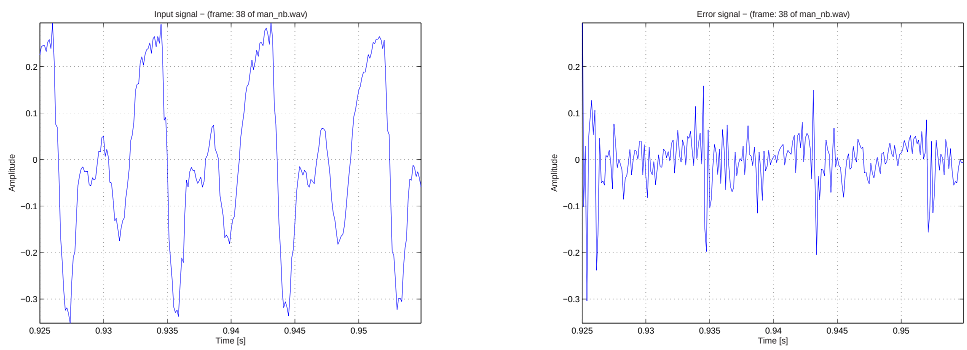
Figure 1.4 show a block diagram of LPC analysis where S is the input signal, g and a is calculated from LPC estimation and e is the residual signal for LPC analysis.

Matlab LPC analysis:

```
e = filter(a,sqrt(g),s);

%e: Error signal from LPC analysis [vector]
%a: LPC coefficients from LPC estimation [vector]
%g: Prediction error variance
%s: Input signal [vector]
```

The above Matlab code calculate e by filtering the input signal S with the inverse transfer function which is found from LPC estimation.



(a) Input signal

(b) Error signal

Figure 1.5: LPC analysis

Figure 1.5(a) show the input signal and figure 1.5(b) show the error signal fra LPC analysis using the above Matlab code.

1.3 LPC-synthesis

LPC synthesis is used to reconstruct a signal from the residual signal and the transfer function of the vocal tract. Because the vocal tract transfer function is estimated from LPC estimation can this be used combined with the residual / error signal from LPC analysis to construct the original signal.

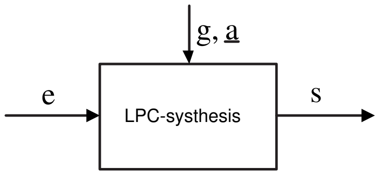


Figure 1.6: LPC-synthesis.

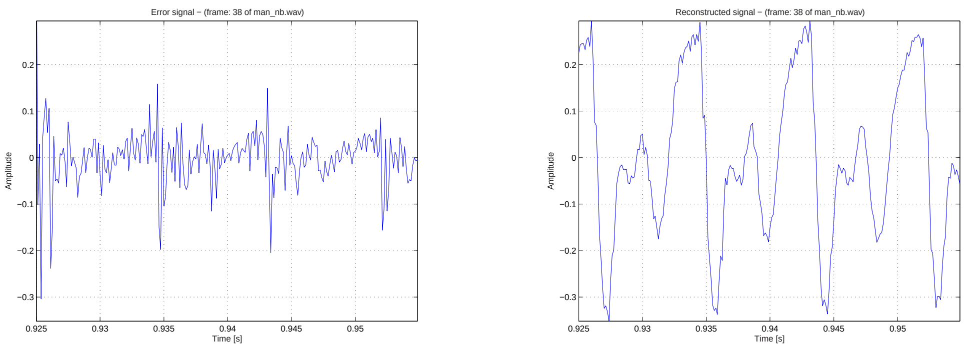
Figure 1.6 show a block diagram of LPC synthesis where e is the error signal found from LPC analysis and g and a from LPC estimation. Reconstruction of the original signal s is done by filtering the error signal with the vocal tract transfer function.

Matlab LPC synthesis:

```
s = filter(sqrt(g),a,e);

%s: Input signal [vector]
%g: Prediction error variance
%a: LPC coefficients from LPC estimation [vector]
%e: Error signal from LPC analysis [vector]
```

The above Matlab code calculate the original signal S from a error signal e and vocal tract transfer function represented with a and g.



(a) Error signal (b) Reconstructed signal

Figure 1.7: LPC synthesis

Figure 1.7(a) show the error signal and figure 1.7(b) show the original signal which is found from LPC synthesis using the above Matlab code.

1.4 Application of LPC

Bandwidth expansion is a method to increase the frequency range of a signal. The increase in frequency is done by adding information about the higher frequency components. The original frequency components (LPC coefficients) is found by using LPC estimation. Then by adding the higher frequency components using code book for envelope extension and excitation extension is it possible to increase the bandwidth of the signal.

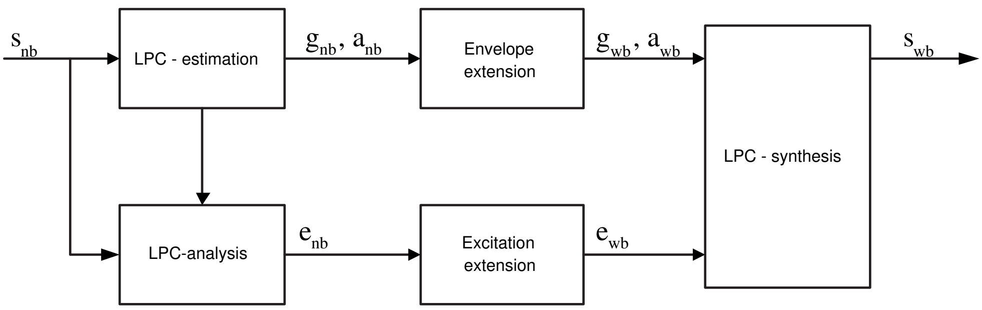


Figure 1.8: Bandwidth expansion.

Figure 1.8 show the block diagram of bandwidth expansion using LPC and codebook (envelope and excitation extension) with additional frequency information.

The matlab code in appendics implements all on the above blockdiagram besides excitation and envelope extension.

1.5 Appendix

1.5.1 Wiener filter theory

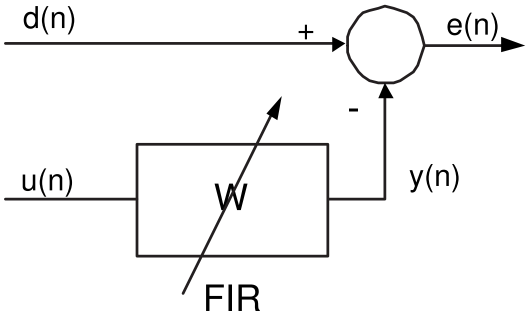


Figure 1.9: Linear discrete-time filter

Orthogonality

$$y(n) = \hat{u}(n|U_n) = \sum_{k=0}^{\infty} w_k^* u(n-k) \quad n = 0, 1, 2, \dots \tag{1.3}$$

$$e(n) = d(n) - y(n) \tag{1.4}$$

$$J = E\left[e(n)e^*(n)\right] = E\left[|e(n)|^2\right] \tag{1.5}$$

$$\nabla_k J = -2E\left[u(n-k)e^*(n)\right] \tag{1.6}$$

$$\nabla_k J = 0 \quad \Rightarrow \quad E\left[u(n-k)e_o^*(n)\right] = 0 \quad k = 0, 1, 2, \dots \tag{1.7}$$

Minimum mean-square error

$$e_o(n) = d(n) - y_o(n) \tag{1.8}$$

$$e_o(n) = d(n) - \hat{d}(n|U_n) \tag{1.9}$$

$$J_{\min} = E\left[|e_o(n)|^2\right] \tag{1.10}$$

Wiener hopf

$$E\left[u(n-k)\left(d^*(n)-\sum_{i=0}^{\infty} w_{oi}u^*(m-i)\right)\right]=0 \quad k=0,1,2,... \tag{1.11}$$

$$\sum_{i=0}^{\infty} w_{oi}E\left[u(n-k)u^*(n-i)\right]=E\left[u(n-k)d^*(n)\right] \quad k=0,1,2,... \tag{1.12}$$

$$r(i-k)=E\left[u(n-k)u^*(n-i)\right] \tag{1.13}$$

$$p(-k)=E\left[u(n-k)d^*(n)\right] \tag{1.14}$$

$$\sum_{i=0}^{\infty} w_{oi}r(i-k)=p(-k) \quad k=0,1,2,... \tag{1.15}$$

$$Rw_o=p \tag{1.16}$$

Wiener hopf (Matrix Formulation)

$$R=\left[u(n)u^H(n)\right] \quad R=\begin{bmatrix} r(0) & r(1) & ... & r(M-1) \\ r^*(1) & r(0) & ... & r(M-2) \\ . & . & & . \\ r^*(M-1) & r^*(M-2) & ... & r(0) \end{bmatrix} \tag{1.17}$$

$$p=E\left[u(n)d^*(n)\right] \quad p \ =\left[p(0),p(-1),...,p(1-M)\right]^T \tag{1.18}$$

$$w_o=\left[w_{o1},w_{o2},...w_{oM-1}\right]^T \tag{1.19}$$

$$w_o=R^{-1}p \tag{1.20}$$

1.5.2 Prediction error filter

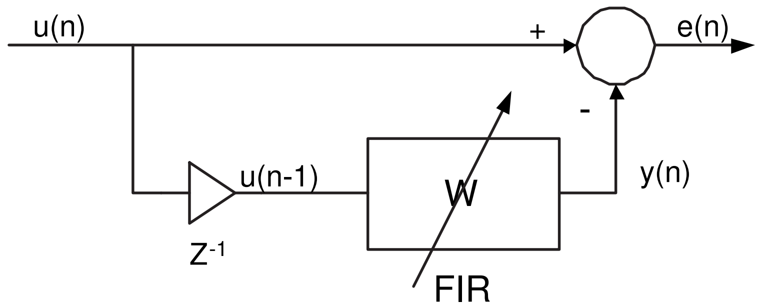


Figure 1.10: Prediction error filter

$$y(n) = \hat{u}(n|U_{n-1}) = \sum_{k=1}^{M} w_k^* u(n-k) \tag{1.21}$$

$$e(n) = u(n) - \hat{u}(n|U_{n-1}) \tag{1.22}$$

$$e(n) = u(n) - \sum_{k=1}^{M} w_k^* u(n-k) \tag{1.23}$$

$$e(n) = \sum_{k=0}^{M} a_k^* u(n-k) \tag{1.24}$$

$$e(n) = \sum_{k=0}^{M} a_k^* u(n-k) \quad a_k = \begin{cases} 1 & k=0 \\ -w_k & k=1,2,..,M \end{cases} \tag{1.25}$$

1.5.3 Application matlab code

```
clear;close all

%%%%%%%%%%%%%%%%%%%%
%Initialize - start
numberofLPCcoeff = 12 %order of forward linear predictor (gives N+1 coefficients)
offset = 20; %used frame in input signal
fftpoints = 512; %number of fftpoints (used for fft and freqz analysis)
hammingwindowed = 1; % if true, the input signal is hamming windowed
wave_as_input = 1; %if true, wave file is used as input signal

plotnumber = 1; %used for numbering the figures(increment for each figure)
plot_global = 1;
plot_estimation_analysis_input_signal = 1;
plot_LPC_estimation_frequency_response = 1;
plot_LPC_analysis_error_signal = 1;
plot_LPC_synthesis_signal_reconstruction = 1;
epsfiles = 0;

framelength = 20*10^-3; %length of frame from inputsignal(even number) [unit: second]
framelengthoverlap = 5*10^-3; %length of overlap between to frames [unit: second]
framelengthwindow = framelength + 2*framelengthoverlap; %total length of frames [unit: second]

%Initialize - end
```

```
%%%%%%%%%%%%%%%%%%

%Loading wavfile - start
used_wav_file = 'man_nb.wav';
[y,fs] = wavread(used_wav_file);
y = y(:,1);

if wave_as_input == 0
y = sin(2*pi*1000*(0:40000)*1/fs)';
end
%Loading wavfile - end

%Downsample inputsignal - start
y = decimate(y,2);
fs = fs/2
%Downsample inputsignal - end

%%%%%%%%%%%%%%%%%%%%%%%
%Pre initalize - start
framesamples = framelengthwindow / (1/fs);  %length of frame from inputsignal [unit: samples]
framesamplesoverlap = framelengthoverlap / (1/fs); %length of overlap between to frames [unit: samples]

y = y(1 : length(y) - mod(length(y),framesamples) ); %fix the length of inputsignal for framing
minmaxy = [min(y) max(y)]; %min and max values of inputsignal (used for plotting)
%Pre initalize - end
%%%%%%%%%%%%%%%%%%%%%%

%Frameing - start
dimensiony = size(y); %used for reconstruction (contain the true dimensions of the inputsignal)
dimensionyframe = [framesamples length(y)/framesamples]; %used for frameing [samples in frame, number of frames]

%framing the data
for i = 1:dimensionyframe(2)
    yframe(:,i) = y(1 + (framesamples-framesamplesoverlap)*(i-1):framesamples + (framesamples-framesamplesoverlap)*(i-1));
end
minmaxyframe = [min(yframe(:,offset)) max(yframe(:,offset))];
%Frameing - end

%Window - start
if hammingwindowed
yframewindow = yframe.*[hamming(dimensionyframe(1))*ones(1,dimensionyframe(2))];
else
yframewindow = yframe;
end
%Window - end

%Modelfitting - start
signalNB = yframewindow;
%Modelfitting - end

%LPC estimation - start
for i = 1:dimensionyframe(2)
    [autosignalNB(:,i),lags(i,:)] = xcorr(signalNB(:,i));
end
autosignalNB = autosignalNB(find(lags(i,:)==0):end,:);
[a,g] = levinson(autosignalNB,numberofLPCcoeff);
%LPC estimation - end

%Frequency reponse of LPC transfer function - start
[H,F] = freqz(g(offset)^0.5,a(offset,:),fftpoints,fs);
%Frequency reponse of LPC transfer function - end

%LPC poly to LSF - start
lsf = poly2lsf(a(offset,:))
%LPC poly to LSF - end

%Frequency reponse of LSF - start
[H1,F1] = freqz(1,lsf,fftpoints,fs);
%Frequency reponse of LSF - end

%LPC LSF to poly - start
%a = lsf2poly(lsf)
%LPC LSF to poly - end

%LPC analysis - start
errorNB = filter(a(offset,:),g(offset).^0.5,signalNB);  % Error signal
%LPC analysis - end

%LPC synthesis - start
signalNBreconstructed = filter(g(offset)^0.5,a(offset,:),errorNB);
%LPC synthesis - end

if plot_global
figure(plotnumber)
plotnumber = plotnumber + 1;

subplot(2,1,1)
hold on
```

```
plotyframe = plot([1:framesamples],yframe(:,offset),'r')
if hammingwindowed
plot([1:framesamples],yframewindow(:,offset),'g'),plotyframewindow = plot([1:5:framesamples],yframewindow(1:5:end,offset),'go')
end
plot([1:framesamples],real(errorNB(:,offset)),'b'),ploterror = plot([1:5:framesamples],real(errorNB(1:5:end,offset)),'bx')
title(sprintf('Input signal and error signal (frame: %d)',offset))
xlabel('Samples [n]'),ylabel('Amplitude'),grid,xlim([1 framesamples])
if hammingwindowed
legend([plotyframe plotyframewindow ploterror],'Inputsignal','Inputsignal*hamming','Errorsignal')
else
legend([plotyframe ploterror],'Inputsignal','Errorsignal')
end

subplot(2,1,2)
minmaxdBscale = [min(20*log10(2*abs(H) / fftpoints))-6 max(20*log10(2*abs(H)/fftpoints))+6];
hold on
plotfft = plot([0:fftpoints-1]*fs/(fftpoints-1),20*log10( 2* abs( fft( yframewindow(:,offset),fftpoints)) / fftpoints ) )
plot(F,20*log10(2 * abs(H) / fftpoints ),'r'), plotlpc = plot(F(1:10:end),20*log10(2 * abs(H(1:10:end)) / fftpoints ),'rx')
stem((lsf/pi)*fs/2,-200+20*log10(ones(1,length(lsf))),'m')
title(sprintf('FFT of input signal and frequency reponse of LPC (frame: %d)',offset))
xlabel('Frequency [Hz]'),ylabel('Amplitude [dB]')
legend([plotfft plotlpc],sprintf('FFT (fftpoints: %d)',fftpoints),sprintf('LPC (order: %d)',numberofLPCcoeff)),grid,ylim([minmaxdBscale]),xlim([0 fs/2])

if epsfiles
    print -depsc -tiff -r300 eps/lpc_estimation_global_plot_fft_lpc_time_BJ
end

end

if plot_estimation_analysis_input_signal
figure(plotnumber)
plotnumber = plotnumber + 1;

plot([0:framesamples-1]*1/fs + (framelengthwindow - framelengthoverlap)*(offset-1),yframewindow(:,offset))
title(texlabel(sprintf('Input signal - (frame: %d of %s)',offset,used_wav_file),'literal'))
xlim([0 framesamples-1]*1/fs + (framelengthwindow - framelengthoverlap)*(offset-1))
xlabel('Time [s]'),ylabel('Amplitude'),ylim([minmaxyframe]),grid

if epsfiles
    print -depsc -tiff -r300 eps/lpc_estimation_input_signal_BJ
end

end

if plot_LPC_estimation_frequency_response
figure(plotnumber)
plotnumber = plotnumber + 1;

subplot(2,1,1)
stem([0:numberofLPCcoeff],a(offset,:))
title(texlabel(sprintf('LPC coefficients - (LPC order: %d, frame: %d of %s)',numberofLPCcoeff,offset,used_wav_file),'literal'))
xlabel('Coefficients [n]'),ylabel('Amplitude')

subplot(2,1,2)
plot(F,20*log10(2*abs(H) / fftpoints ))
title(texlabel(sprintf('LPC frequency response - (frame: %d of %s)',offset,used_wav_file),'literal'))
xlim([0 fs/2]),xlabel('Frequency [Hz]'),ylabel('Amplitude [dB]'),grid

if epsfiles
    print -depsc -tiff -r300 eps/lpc_estimation_frequency_response_of_lpc_BJ
end

end

if plot_LPC_analysis_error_signal
figure(plotnumber)
plotnumber = plotnumber + 1;

plot([0:framesamples-1]*1/fs + (framelengthwindow - framelengthoverlap)*(offset-1),errorNB(:,offset))
title(texlabel(sprintf('Error signal - (frame: %d of %s)',offset,used_wav_file),'literal'))
xlim([0 framesamples-1]*1/fs + (framelengthwindow - framelengthoverlap)*(offset-1))
xlabel('Time [s]'),ylabel('Amplitude'),ylim([minmaxyframe]),grid

if epsfiles
    print -depsc -tiff -r300 eps/lpc_analysis_error_signal_BJ
end

end

if plot_LPC_synthesis_signal_reconstruction
figure(plotnumber)
plotnumber = plotnumber + 1;

plot([0:framesamples-1]*1/fs + (framelengthwindow - framelengthoverlap)*(offset-1),signalNBreconstructed(:,offset))
```

```
title(texlabel(sprintf('Reconstructed signal - (frame: %d of %s)',offset,used_wav_file),'literal'))
xlim([0 framesamples-1]*1/fs + (framelengthwindow - framelengthoverlap)*(offset-1))
xlabel('Time [s]'),ylabel('Amplitude'),ylim([minmaxyframe]),grid

if epsfiles
    print -depsc -tiff -r300 eps/lpc_synthesis_signal_reconstruction_BJ
end

end
```