T8 Analiza și sinteza prin predicție liniară

T8.1. Coeficienții de predicție liniară

Analiza prin predicție liniară (en. *Linear Prediction Analysis*) este o altă metodă, alături de cepstrum, de separare a sursei de filtru din modelul sursă-filtru de producere a vorbirii. Principiul fundamental al acestei analize este bazat pe gradul înalt de corelație al eșantioanelor semnalului vocal. Această corelație este dată de inerția organelor fonatoare, astfel încât sunetul emis nu poate fi modificat într-un interval de timp foarte mic. Tot această inerție stă și la baza cvasi-staționarității semnalului vocal.

Datorită acestei corelații eșantioanele de semnal pot fi estimate ca o sumă ponderată a eșantioanele anterioare:

$$\hat{y[n]} = \sum_{k=1}^{p} a_k y[n-k] \tag{T8.1.1}$$

unde p este ordinul de predicție. Eroarea de predicție este dată de:

$$e[n] = y[n] - \hat{y[n]} = \sum_{k=1}^{p} y[n-k] \tag{T8.1.2}$$

Dacă trecem în domeniul z obținem:

$$E(z) = Y(z) - \sum_{k=1}^{p} a_k z^{-k} Y(z) \tag{T8.1.3}$$

Împărțim ambii termeni cu $Y(z)$ și inversăm ecuația:

$$\frac{Y(z)}{E(z)} = \frac{1}{1 - \sum_{k=1}^{p} a_k z^{-k}} \tag{T8.1.4}$$

Această ecuație seamănă cu o funcție de transfer ce conține doar poli. Acest lucru este în conformitate și cu un alt model de producere a vorbirii derivat

din principii de acustică teoretică. Acesta spune că tractul vocal poate fi modelat cu un set finit de tuburi de diferite lungimi și raze. Aceste tuburi introduc fiecare o pereche de poli complex conjugați în funcția de transfer a tractului vocal. Drept urmare, putem scrie $H(z)$ ca un filtru ce are doar poli:

$$H(z) = \frac{1}{1 - \sum_{k=1}^{p} a_k z^{-k}} \tag{T8.1.5}$$

Pornind de la aceste două observații putem conclude faptul că eroare de predicție din ecuațiile inițiale nu este altceva decât sursa semnalului vocal și anume oscilația corzilor vocale sau fluxul de aer nemodulat expirat din plămâni.

Deci, pentru a determina sursa și filtrul din modelul de producere a vorbirii este suficient să determinăm coeficienții a_k ai funcției de transfer anterioare. Acești coeficienți sunt denumiți **coeficienți de predicție liniară** (en. *linear prediction coefficients* (LPC)). Calculul lor implică rezolvarea unui sistem de ecuații de ordin p, iar pentru aceasta există o serie de metode matematice de rezolvare rapidă, dintre care cea mai des folosită este recursivitatea Levinson-Durbin https://en.wikipedia.org/wiki/Levinson_recursion . Detalierea algoritmilor de calcul ai coeficienților LPC nu face parte din scopul acestei cărți și lăsăm la latitudinea cititorului aprofundarea acestora.

Să vedem acum ce informații ne oferă coeficienții LPC despre semnalul vocal. Să citim mai întâi două semnale: sonor și nesonor:

[1]:
```
import wave
import numpy as np
######################
# Citim vocala
input_wav_vowel = 'speech_files/a.wav'
wav_struct_vowel = wave.open(input_wav_vowel, 'r')
sampling_frequency = wav_struct_vowel.getframerate()
wav_bytes_vowel = wav_struct_vowel.readframes(-1)
wav_data_vowel = np.frombuffer(wav_bytes_vowel, \
        dtype='int16')
wav_data_vowel = wav_data_vowel \
        /float(max(abs(wav_data_vowel)))
wav_struct_vowel.close()
######################
# Citim consoana
input_wav_consonant = 'speech_files/s.wav'
wav_struct_consonant = wave.open(input_wav_consonant, 'r')
```

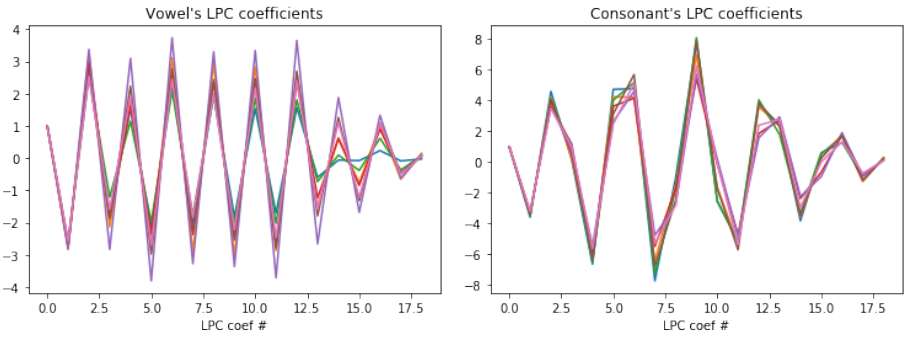
```
wav_bytes_consonant = wav_struct_consonant.readframes(-1)
wav_data_consonant = np.frombuffer(wav_bytes_consonant,\
         dtype='int16')
sampling_frequency_c = wav_struct_consonant.getframerate()
wav_data_consonant = wav_data_consonant  \
        /float(max(abs(wav_data_consonant)))
wav_struct_consonant.close()
```

Pentru a extrage coeficienții LPC din fiecare cadru de semnal, vom folosi din nou modulul `librosa`, submodulul `core` ce conține funcția `lpc()`. Această funcție ia ca intrare un semnal și un ordin al predictorului și returnează coeficienții LPC, inclusiv termenul liber din numitorul funcției de transfer.

> **OBS T8.1** Ordinul predictorului pentru semnalele vocale a fost stabilit empiric de către Fant ca fiind egal cu frecvența de eșantionare exprimată în kHz plus 2. Această formulă se bazează pe observația că în medie într-un semnal vocal nu poate să existe mai mult un pol la fiecare kilohertz. Astfel că alegerea unui ordin egal cu $[F_s] + 2$ funcționează și în practică.

[2]:
```
import librosa.core as lb

from scipy.signal import hamming
import matplotlib.pylab as pl
%matplotlib inline

# Fereastra de analiză cu lungime egală cu putere
# a lui 2 fără suprapunere
window_length = int(20*1e-3*sampling_frequency)
window_fft = int(2**np.ceil(np.log2(window_length)))
p = 0
# Fereastră Hamming
hamming_window = hamming(window_fft)
# Numărul de cadre
number_of_frames = int(len(wav_data_vowel)/window_fft)
# Stabilim ordinul LPC la Fs + 2
lpc_order = sampling_frequency//1000 + 2
# Inițializăm o matrice nulă în care vom stoca valorile
# coeficienților LPC din fiecare cadru. Numărul
# de coeficienți LPC returnat de funcție este egal
# cu ordinul LPC+1 datorită termenului liber
lpcs = np.zeros ([number_of_frames, lpc_order+1])
```

```
for k in range(number_of_frames):
    # Extragem doar un cadru din semnal
    current_frame = wav_data_vowel[k*window_fft:  \
            (k+1)*window_fft]
    hamming_frame = np.multiply(hamming_window,  \
            current_frame)
    lpcs[k,:] = lb.lpc(hamming_frame, lpc_order)
# Plot
pl.plot(np.transpose(lpcs))
pl.title("Vowel's LPC coefficients")
pl.xlabel('LPC coef #');

# Numarul de cadre din consoana
number_of_frames = int(len(wav_data_consonant)/window_fft)
lpcs = np.zeros ([number_of_frames, lpc_order+1])
for k in range(number_of_frames):
    # Extragem doar un cadru din semnal
    current_frame = wav_data_consonant[k*window_fft:  \
            (k+1)*window_fft]
    hamming_frame = np.multiply(hamming_window,  \
            current_frame)
    lpcs[k,:] = lb.lpc(hamming_frame, lpc_order)
# Plot
pl.figure()
pl.plot(np.transpose(lpcs))
pl.title("Consonant's LPC coefficients")
pl.xlabel('LPC coef #');
```



Se poate observa faptul că valorile acestor coeficienți sunt constante de-a lungul celor două segmente vocale și că valoarea primului coeficient returnat de funcția `lpc()` este întotdeauna 1. Valorile constante ale coeficienților

LPC ne indică faptul că tractul vocal și filtrul determinat de acesta nu se modifică. Ceea ce este adevărat atât timp cât segmentul vocal conține o singură fonemă cu caracteristici statice, cum sunt vocalele sau anumite consoane.

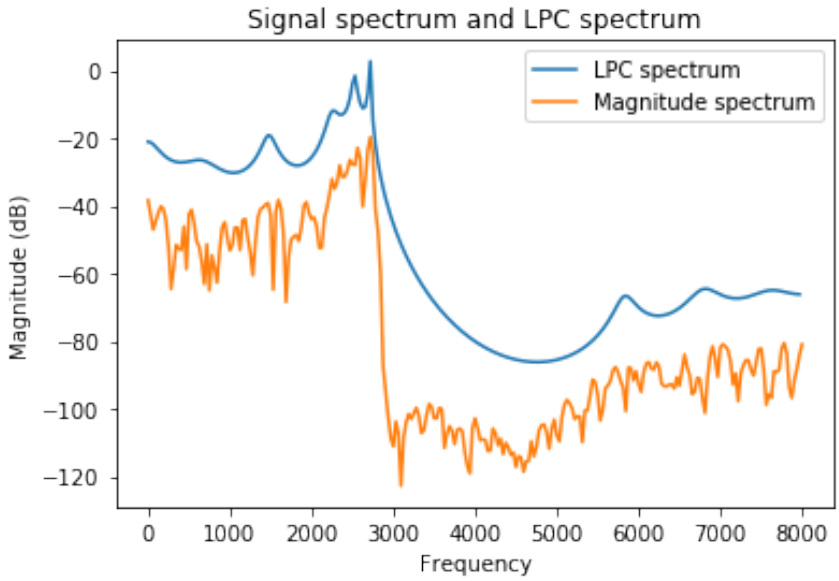
Exercițiu T8.1.1 Afișați valorile coeficienților LPC pentru o altă consoană cu caracteristici mai dinamice, cum ar fi p, c, d, etc. ■

Exercițiu T8.1.2 Afișați valorile coeficienților LPC pentru un segment vocal ce conține mai multe foneme. Sunt valorile coeficienților LPC constante? ■

T8.2. Spectrul LPC

Știind că acești coeficienți LPC sunt de fapt coeficienții unui filtru, putem să vizualizăm răspunsul său în frecvență. Vom afișa acest răspuns alături de spectrul semnalului, pentru a putea identifica eventualele similitudini:

[3]:
```
from scipy.signal import freqz
# Extragem un singur cadru al vocalei
k = 3
vowel_frame = wav_data_vowel[k*window_fft: (k+1)*window_fft]
hamming_frame = np.multiply(hamming_window, current_frame)
a = lb.lpc(hamming_frame, lpc_order)
# Obținem răspunsul în frecvență al filtrului dat de
# coeficienții LPC. Lungimea răspunsului o luăm egală
# cu numărul de puncte FFT al spectrului semnalului
w, h = freqz(1, a , window_fft//2)
# Axa frecvenței
freq_axis = np.arange(window_fft//2)*sampling_frequency \
        /window_fft
# Afișăm pe axă logaritmică spectrul LPC
pl.plot(freq_axis, 20*np.log10(1.0/window_fft*abs(h)))
# Spectrul semnalului
pl.magnitude_spectrum(hamming_frame,  \
        Fs = sampling_frequency, scale='dB')
pl.title("Signal spectrum and LPC spectrum")
pl.legend(["LPC spectrum", "Magnitude spectrum"]);
```



Din figura anterioară putem să observăm faptul că spectrul LPC, asemeni spectrului lui h dat de coeficienții cepstrali este anvelopa spectrală a spectrului semnalului. Astfel că putem să concluzionăm faptul că acești coeficienți sunt o bună aproximare a filtrului dat de tractul vocal.

Exercițiu T8.2.1 Variați ordinul coeficienților LPC și observați modificarea spectrului LPC. ■

Exercițiu T8.2.2 Afișați spectrul LPC pentru consoană. ■

T8.2.1 Calculul formanților pe baza coeficienților LPC

Având mai bine evidențiată anvelopa spectrală a semnalului vocal și zonele de energie maximă locală, putem să determinăm formanții segmentelor sonore.

După cum am menționat anterior, **formanții** sunt frecvențele de rezonanță ale tractului vocal și sunt prezenți doar în cadrul segmentelor sonore. Formanții sunt un element important al analizei semnalului vocal și determină identitatea sunetului emis (vocala). Ca urmare, o primă formă de sinteză de voce, utilizată și în ziua de azi de către foneticieni, este **sinteza formantică**.[1]

[1] `https://ccrma.stanford.edu/~jos/pasp/Formant_Synthesis_Models.html`

Din spectrul LPC afișat anterior putem identifica punctele de energie spectrală maximă locală corespunzătoare formanților prin identificarea punctelor de inflexiune ale funcției matematice. Punctele de inflexiune sunt date de rădăcinile numărătorului funcției.

Secvența de cod următoare calculează aceste rădăcini complexe și le ordonează crescător în funcție de faza lor. Valorile în Hz a formanților fiind date de formula:

$$F = \frac{faza}{2\pi} * F_s \ [Hz]$$

În aplicații practice, se folosesc maxim primii 3-4 formanți. Vom limita și noi calculul lor la 4 valori:

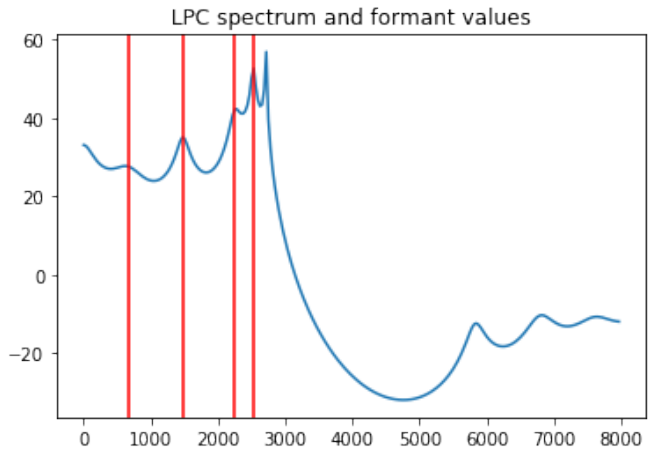
[4]:
```
def extract_formants(input_sample, lpc_order, fs):
    a = lb.lpc(input_sample,lpc_order)

    # Extragem rădăcinile polinomului dat de
    # coeficienții LPC
    roots = np.roots(a)
    # Rădăcinile sunt complex conjugate, reținem doar
    # o valoare din pereche
    roots = roots[np.where(np.imag(roots) > 0)]
    # Calculăm fazele rădăcinilor
    angles = np.arctan2(np.imag(roots), np.real(roots))
    # Calculăm frecvențele
    freqs = angles * (fs / (2 * np.pi))
    # Reordonăm în ordinea crescătoare a fazelor
    frequency_indices = np.argsort(freqs)
    formants = [int(x) for x in freqs[frequency_indices]]
    # Benzile de frecvență ale formanților
    # sunt date de distanța polilor față de
    # cercul unitate
    bw = -1 / 2 * (fs / (2 * np.pi))  \
            * np.log(np.abs(roots[frequency_indices]))
    # Frecvențele formanților trebuie să
    # fie mai mari decât 90Hz cu o bandă
    # de frecvențe mai mică de 400Hz
    formants = [f for i,f in enumerate(formants) \
                if f>90 and bw[i]<400]
    return formants[:4]
```

[5]:
```
# Extragem valorile formanților dintr-un cadru al vocalei
formants = extract_formants(hamming_frame, lpc_order, \
        sampling_frequency)
print ("Formant values: "+' '.join([str(x)+'Hz' \
        for x in formants]) )
```

[5]:
```
Formant values: 661Hz 1476Hz 2247Hz 2524Hz
```

Să afișăm aceste valori peste spectrul LPC:

[6]:
```
# Plot spectrul LPC
pl.plot(freq_axis, 20*np.log10(abs(h)))
# Plot valori formanți
for f in formants:
    pl.axvline(f, color = 'r')
pl.title("LPC spectrum and formant values");
```



Valorile formanților determinate anterior se suprapun perfect cu punctele de inflexiune ale spectrului LPC.

Exercițiu T8.2.3 Încercați să determinați formanții și pentru consoană. Ce obțineți? ■

T8.2.2 Eroarea de predicție

La începutul acestui tutorial am menționat faptul că eroarea de predicție este egală cu sursa de semnal. Să vedem cum arată această eroare de

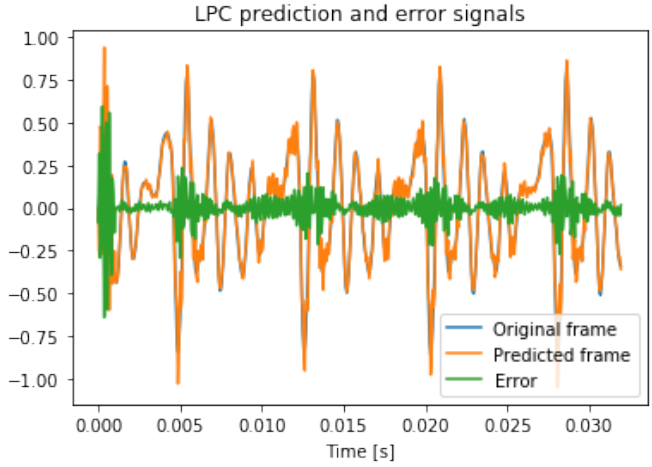
predicție. Vom filtra semnalul vocal cu filtrul invers dat de:

$$H(z) = \sum_{k=1}^{p} a_k z^{-k} \tag{T8.2.1}$$

pentru a obține semnalul prezis, iar mai apoi vom scădea din semnalul original semnalul prezis.

Trebuie să ținem cont de modul în care funcția `lpc()` ne returnează valorile coeficienților. Și anume, acestea includ termenul liber și semnul minus dinaintea sumei de la numitor:

[7]:
```
from scipy.signal import lfilter
# Creăm setul de coeficienți pentru filtrul invers
a_hat =  -1*a
a_hat[0] = 0
# Cadru din vocală
frame = vowel_frame
# Filtrăm cu filtrul invers
y_hat = lfilter(a_hat, 1, frame)
# Calculăm eroarea
err = frame - y_hat
# Plot
time_axis = np.arange(0, window_fft)*1.00/sampling_frequency
pl.plot(time_axis, frame)
pl.plot(time_axis, y_hat)
pl.plot(time_axis, err);
pl.title("LPC prediction and error signals")
pl.legend(["Original frame", "Predicted frame", "Error"]);
```



Observăm că semnalul prezis este foarte apropiat de semnalul original, eroarea fiind aproape zero de-a lungul acestui cadru de analiză.

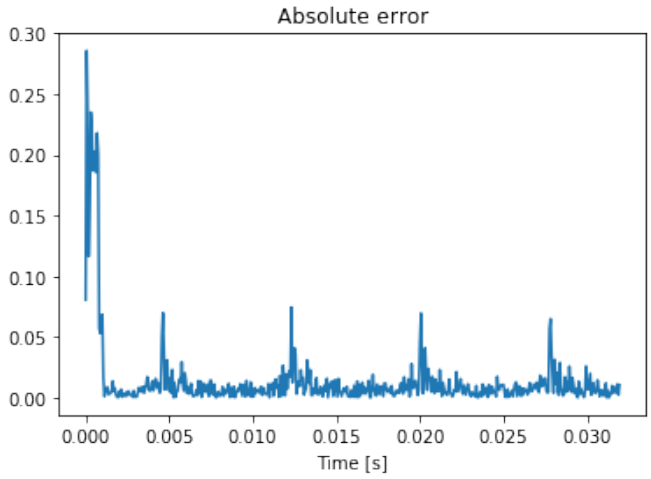
Exercițiu T8.2.4 Calculați eroarea de predicție și pentru un cadru al consoanei? Ce observați? ■

T8.3. Sinteza LPC

Până în momentul de față am reușit să extragem coeficienții LPC din semnal, să vizualizăm spectrul dat de acești coeficienți și să calculăm eroarea de predicție. Însă una dintre cele mai mari aplicații ale analizei LPC este cea de codare. Pe lângă extragerea coeficienților LPC este nevoie să se realizeze și sinteza semnalului vocal folosind cât mai puțini parametri transmiși sau stocați. Astfel că, dacă am reușit să utilizăm doar $Fs + 2$ coeficienți pentru a modela filtrul, trebuie să găsim și o modalitate de a reduce datele sursei.

Să vedem mai întâi ce informații putem regăsi în această eroare. Afișăm eroarea absolută însă pentru un cadru neponderat Hamming, pentru a fi mai evidentă informația:

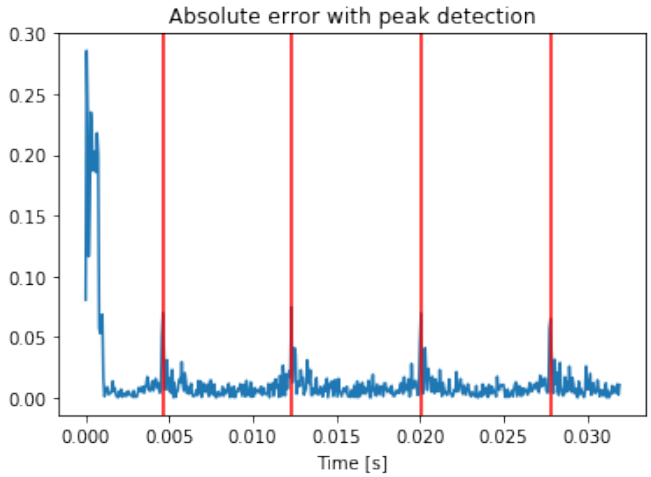
[8]:
```
# Cadru din vocală
frame = vowel_frame
a = lb.lpc(frame, lpc_order)
# Creăm setul de coeficienți pentru filtrul invers
a_hat =  -1*a
a_hat[0] = 0
# Filtrăm cu filtrul invers
y_hat = lfilter(a_hat, 1, frame)
# Calculăm eroarea
err = frame - y_hat
# Eroarea pătratică
err_square = abs(err)
pl.plot(time_axis, err_square);
pl.xlabel("Time [s]")
pl.title("Absolute error");
```



Se poate observa că pentru secvenţe sonore, în eroarea de predicţie apar maxime distanţate cu perioada fundamentală T0 (exceptând eroarea din primele câteva eșantioane). Să încercăm să extragem F_0 din eroarea de predicție:

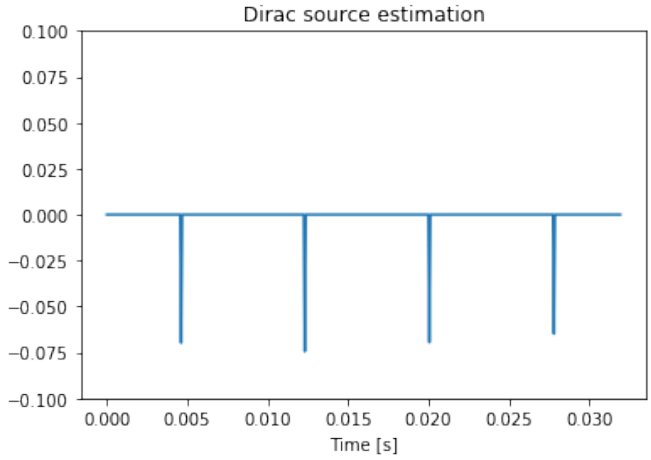
[9]:
```
from scipy.signal import find_peaks
# Reținem doar maximele distanțate cu minim
# 1/480Hz = T0 minim și ignorând eroare
# de început de cadru
peaks,_ = find_peaks(err_square[20:], distance=90)
# Corectăm indecșii returnați
peaks = peaks+20
# Calculăm distanța dintre indecșii returnați de funcție:
difs = [x-peaks[i-1] for i,x in enumerate(peaks)][1:]
# Determinăm media diferențelor
average_dist = np.mean(difs)
# Și o convertim în Hz
F0 = sampling_frequency/average_dist
# Afișăm
print ("F0 estimated from LPC error: %d Hz" %(int(F0)))
pl.plot(time_axis, err_square);
for z in peaks:
    pl.axvline(z*1.0/sampling_frequency, color = 'r')
pl.xlabel("Time [s]")
pl.title("Absolute error");
```

[9]:
```
F0 estimated from LPC error: 129 Hz
```

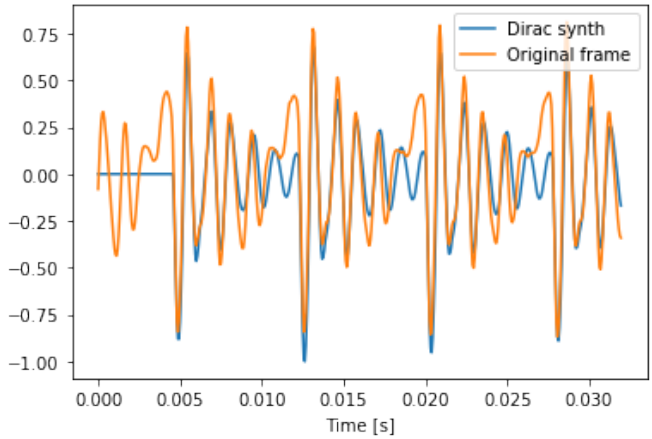


Dacă eroarea de predicție conține aceste maxime distanțate cu T_0, iar această eroare reprezintă sursa ideală de semnal pentru filtrul LPC, am putea să încercăm să modelăm această sursă cu impulsuri Dirac poziționate la indecșii maximelor și de amplitudine egală cu acestea:

[10]:
```
# Vector pentru sursa Dirac
dirac_source = np.zeros(window_fft)
dirac_source[peaks] = err[peaks]
pl.plot(time_axis, dirac_source)
pl.ylim([-0.10,0.10])
pl.xlabel("Time [s]")
pl.title("Dirac source estimation")
```



[11]:
```
# Filtrăm sursa Dirac cu filtrul LPC:
dirac_synth = lfilter([1.],a, dirac_source)
# Normalizăm pentru că nu am calculat
# câștigul filtrului LPC
dirac_synth = dirac_synth/(max(abs(dirac_synth)))
# Plot
time_axis = np.arange(0, window_fft)*1.00/sampling_frequency
pl.plot(time_axis, dirac_synth)
pl.plot(time_axis, vowel_frame)
pl.xlabel("Time [s]")
pl.legend(["Dirac synth", "Original frame"]);
```



[12]:
```
# Repetăm cadrul pentru a putea auzi rezultatul
dirac_synth_long = np.tile(dirac_synth, 3)
# Ascultăm sinteza
import IPython
IPython.display.Audio(dirac_synth_long,  \
        rate=sampling_frequency)
```

[12]:

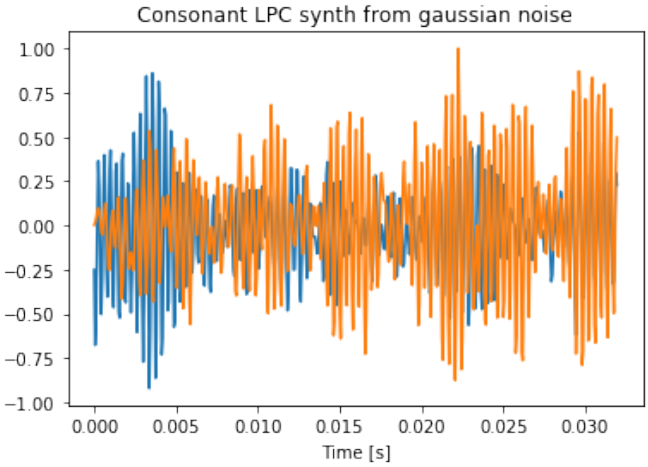

[13]:
```
# Ascultăm și semnalul original
IPython.display.Audio(wav_data_vowel,  \
        rate=sampling_frequency)
```

[13]:


Exercițiu T8.3.1 Reluați pașii de sinteză folosind altă vocală. Poate fi determinată identitatea vocalei din sinteza cu impulsuri Dirac? ■

Pentru consoane se folosește zgomot alb gaussian.

[14]:
```
# Cadrul consoanei
consonant_frame= wav_data_consonant[k*window_fft:  \
        (k+1)*window_fft]
noise = np.random.normal(scale=0.05* \
        np.max(consonant_frame), size=window_fft)
# Filtrare inversă consoană
a = lb.lpc(consonant_frame, lpc_order)
noise_synth = lfilter([1],a, noise)
noise_synth = noise_synth/max(abs(noise_synth))
noise_synth_long = np.tile(noise_synth, 10)
pl.plot(time_axis,consonant_frame)
pl.plot(time_axis, noise_synth)
pl.xlabel("Time [s]")
pl.title("Consonant LPC synth from gaussian noise");
```



[15]:
```
IPython.display.Audio(noise_synth, rate=sampling_frequency_c)
```

[15]:

[16]:
```
IPython.display.Audio(wav_data_consonant, \
        rate=sampling_frequency_c)
```

[16]:


Exerciţiu T8.3.2 Încercați să realizați sinteza LPC pe un întreg semnal vocal făcând distincția automat între consoane și vocale pe baza erorii de predicție. ■

În mod evident, simplificând atât de mult sursa de semnal va rezulta într-o degradare majoră a calității semnalului sintetizat. Pentru a evita acest lucru se folosesc combinații de impulsuri și zgomot cu ponderi variabile atât pentru consoane, cât și pentru vocale.

T8.4. Concluzii

În cadrul acestui tutorial am introdus analiza prin predicție liniară. Această analiză permite separarea sursei de filtru din modelul liniar-separabil de producere a vorbirii. Am văzut totodată și modul în care putem calcul formanții segmentelor sonore pornind de la spectrul dat de coeficienții LPC, precum și modul în care putem realiza sinteza LPC minimizând informația din sursa de semnal. De altfel, una dintre cele mai importante aplicații ale analizei prin predicție liniară este cea de codare. Metoda de codare din GSM - o variantă a Code Excited Linear Prediction - utilizează acest tip de analiză.

BIBLIOGRAFIE SUPLIMENTARĂ

- Wai C. Chu, "Speech Coding Algorithms", Wiley&Sons, 2003
- Paul Taylor, ”Text to speech synthesis”, Cambridge University Press, 2009
- Benesty et al, “Springer Handbook of Speech Processing”, Springer, 2008

RESURSE MEDIA

- ETSI GSM Standard 2g - online: `https://www.etsi.org/technologies/mobile/2g`
- IRCAM, "AudioSculpt 3.0 - user manual" - online `http://support.ircam.fr/docs/AudioSculpt/3.0/co/LPC.html`
- Coursera, "Internet of Things: Communication Technologies Course", online: `https://www.coursera.org/lecture/internet-of-things-communication/linear-predictive-coding-of-speech-LZ10C`