

# Elaborazione audio

---

Elaborazione dell'audio digitale

*Ingegneria del Cinema, Informatica e Telecomunicazioni*

**Antonio Servetti**

Internet Media Group

Dip. di Automatica ed Informatica

Politecnico di Torino

[servetti@polito.it](mailto:servetti@polito.it)

<http://media.polito.it>

# Sommario

---

- Modellazione sonora: sorgente + filtro
  - ✓ Analisi cepstrale
  - ✓ Applicazione al segnale vocale: analisi e sintesi di fonemi
  - ✓ (accenno) Mel frequency cepstral coefficients (MFCC)
  
- Effetto autotune
  - ✓ Identificazione del pitch (tecnica dell'autocorrelazione)
  - ✓ Mapping del pitch
  - ✓ Trasposizione del pitch (tecnica TD-PSOLA)

# Riferimenti bibliografici

---

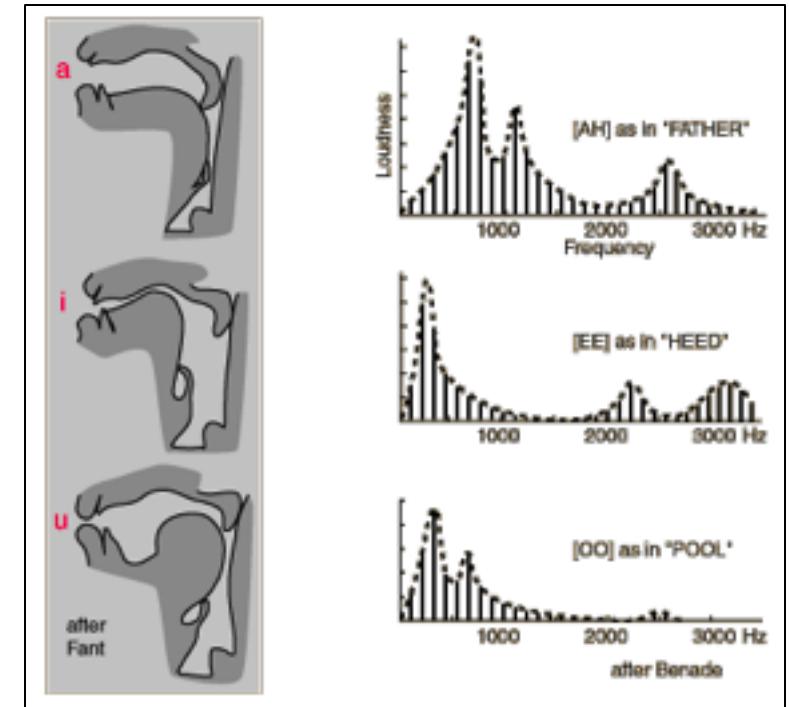
- Modello sorgente + filtro
  - ✓ De Poli, Algorithms for Sound & Music Processing, 2006
    - Sez. 2.5 Source-filter models
    - Sez. 5.2 Spectral envelope estimation
- Identificazione del pitch
  - ✓ Benesty, "Handbook of Speech Processing", 2007
    - Ch. 10 Pitch and Voicing Determination
- Time-domain Pitch Synchronous Overlap Add
  - ✓ Dutoit, "An Introduction to Text-to-Speech Synthesis", 1997
    - Ch. 10 Time-Domain Algorithms
- Muller, "A Review of Time-Scale Modification of Music Signals", 2016

---

## MODELLO SORGENTE + FILTRO

# Modellazione sonora

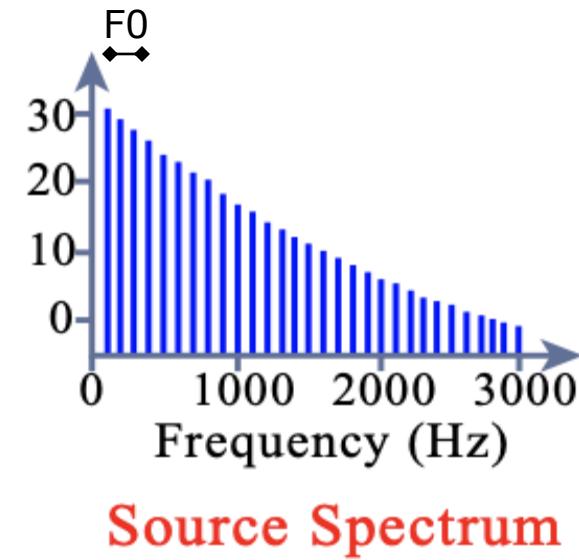
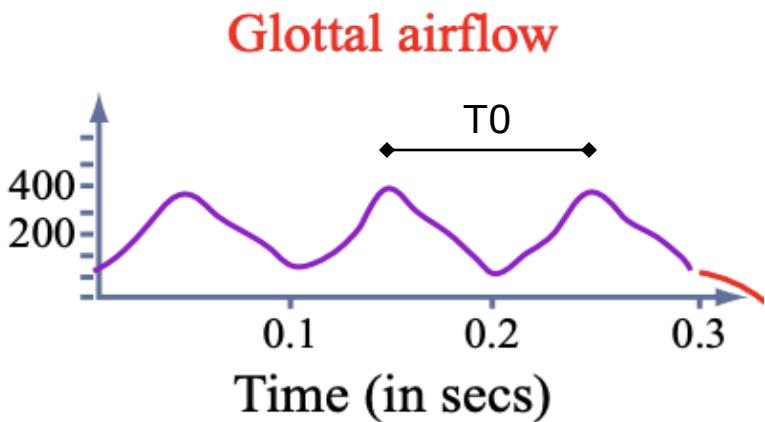
- Molti suoni, tra cui il segnale vocale e quello degli strumenti musicali può essere rappresentato tramite un modello composto da:
  - ✓ 1. una sorgente vibrante (eccitazione)
  - ✓ 2. una cassa di risonanza (filtro)
- Esempio voce:
  - ✓ Vibrazione delle corde vocali + filtro del tratto vocale  
(composto da faringe, cavità orale, lingua, cavità nasale e labbra)
- Esempio strumenti musicali:
  - ✓ Vibrazione della corda o dell'ancia + cassa di risonanza



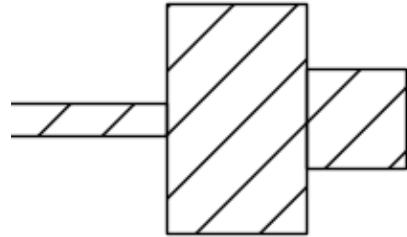
Reference: <https://hyperphysics.phy-astr.gsu.edu/hbase/Music/vowel.html>

# Voce: sorgente + filtro

- Sorgente: eccitazione prodotta dalle corde vocali
  - ✓ Serie di "sbuffi" simile ad una onda triangolare
  - ✓ Struttura "fine" dello spettro:
    - La trasformata è un treno di armoniche linearmente decrescente
  - ✓ Determina il periodo / pitch del suono

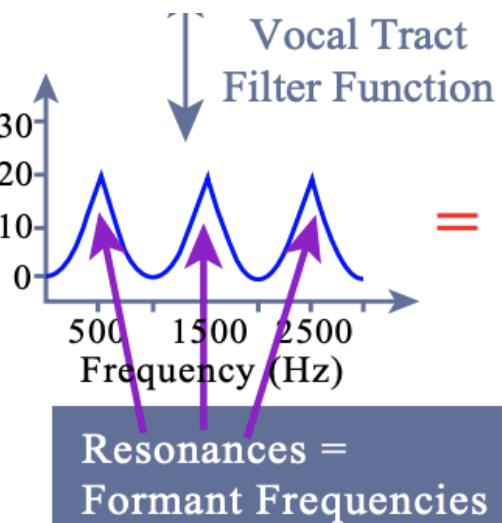
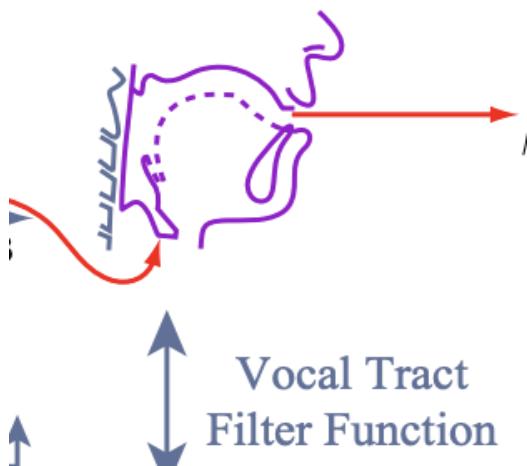


# Voce: sorgente + filtro



## ■ Filtro: cassa armonica del tratto vocale

- ✓ Effetto delle risonanze dovute alla lunghezza e larghezza del tratto vocale (modello: tubo a più sezioni)
- ✓ Agisce come una serie di filtri peak centrati sulle risonanze
- ✓ Determina le formanti, picchi dell'inviluppo dello spettro

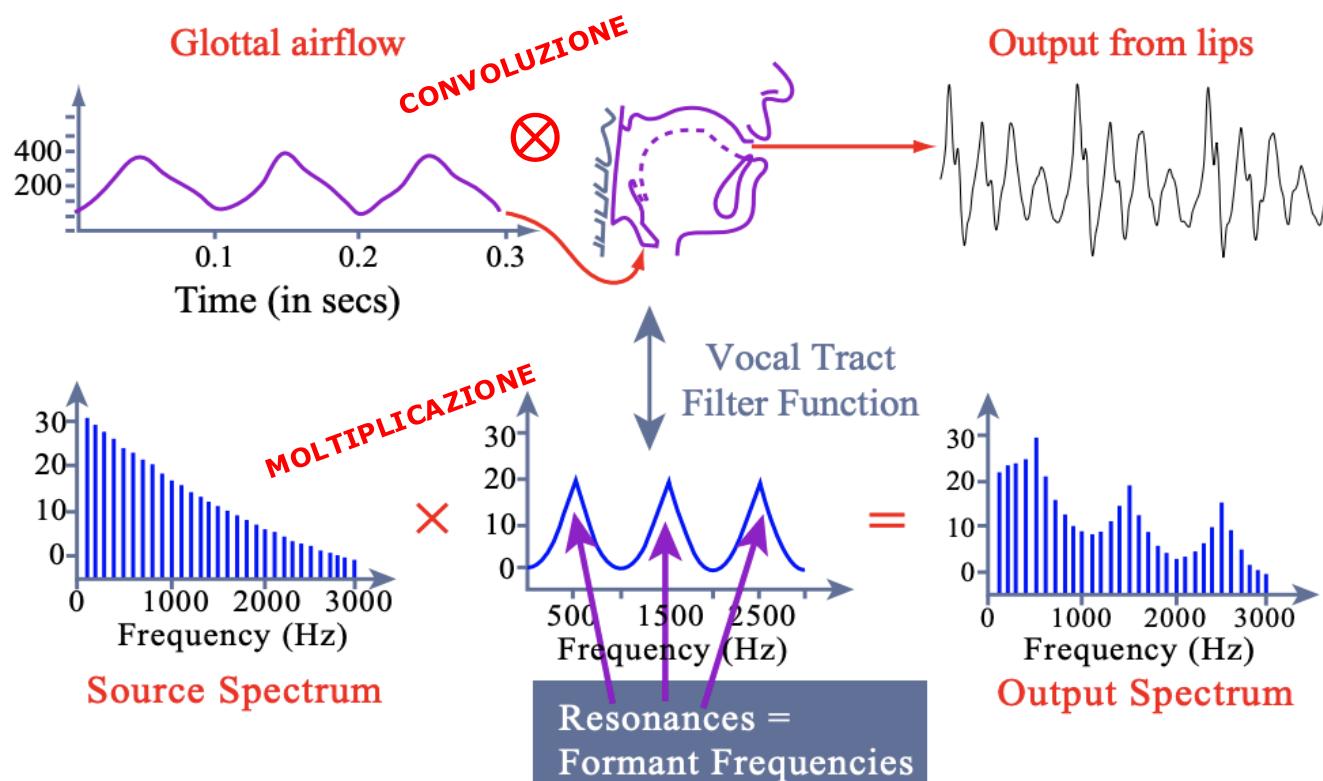


Prima risonanza  $F_1$  dipende dalla posizione verticale della lingua: lingua alta, cavità più lunga e stretta risonanza più bassa; lingua bassa cavità corta e aperta risonanza alta.

Seconda risonanza  $F_2$ : dipende dalla posizione orizzontale della lingua: lingua anteriore (verso i denti) risonanza più alta; lingua posteriore risonanza più bassa.

# Voce: sorgente + filtro

- L'azione congiunta di (prodotto in frequenza tra) sorgente e filtro determina la forma d'onda risultante



Esempi:

- vowel\_u.wav
- vowel\_o.wav
- vowel\_i.wav
- vowel\_e.wav
- vowel\_a.wav
- vowel-synthesizer-v1.py

Reference: [https://ocw.mit.edu/courses/linguistics-and-philosophy/24-915-linguistic-phonetics-fall-2015/lecture-notes/MIT24\\_915F15\\_lec4.pdf](https://ocw.mit.edu/courses/linguistics-and-philosophy/24-915-linguistic-phonetics-fall-2015/lecture-notes/MIT24_915F15_lec4.pdf)

# Applicazioni modello: sorgente + filtro

---

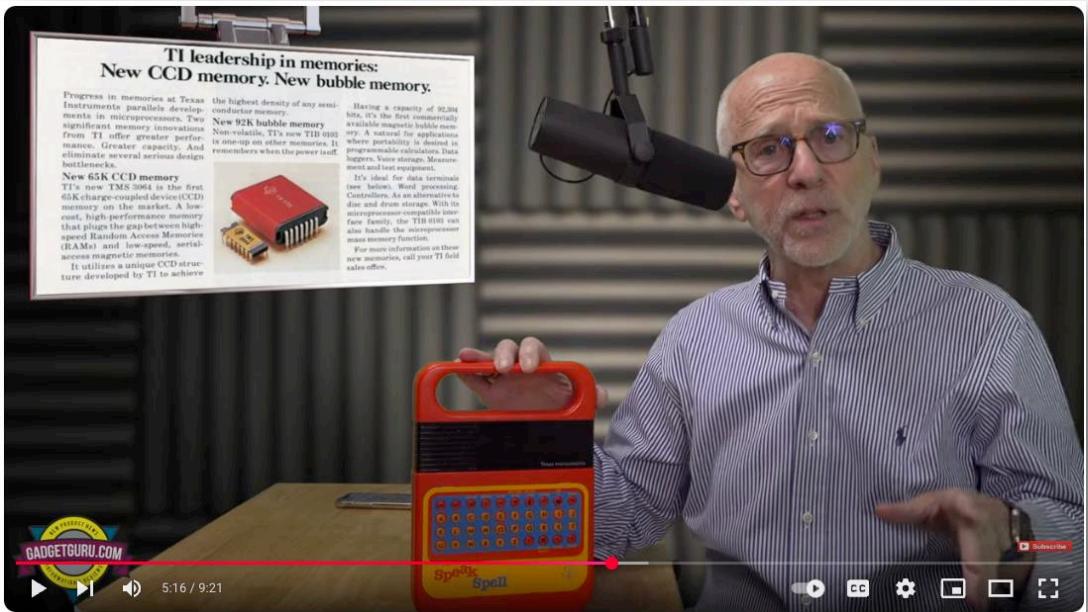
- Classificazione
  - ✓ Riconoscimento del parlato e del parlatore
  - ✓ Content-based retrieval
- Codifica della voce nella telefonia digitale
- Sintesi vocale/musicale ed effetti
  - ✓ Cambio di intonazione o del timbro vocale
  - ✓ Sintesi di strumenti sonori

# Speak and spell (il grillo parlante)

- Prima elaborazione audio digitale per sintesi della voce (1978)
  - ✓ Prodotto dalla Texas Instruments e distribuito dalla Clementoni

Video: Texas Instruments Speak and Spell - The Story Behind The Product

<https://www.youtube.com/watch?v=z5z4hWfNWA4>



Demo

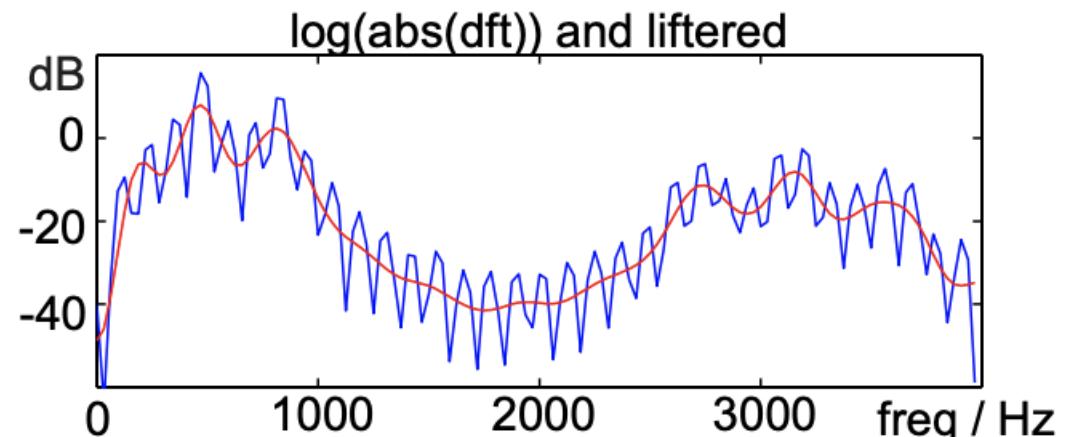
<https://www.youtube.com/watch?v=GvPAS6W3q78>



# Analisi cepstrale - intro

## ■ Analisi in frequenza dell'analisi in frequenza del segnale

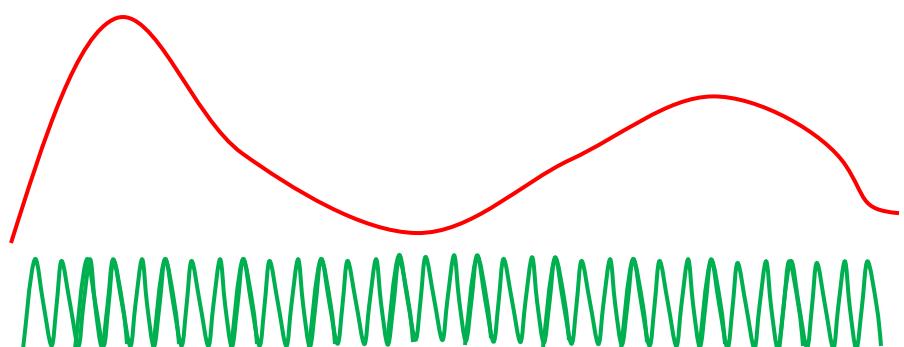
- ✓ Analisi in frequenza dello spettro
  - Periodicità delle armoniche (sorgente/eccitazione)
  - Periodicità delle formanti (filtro)



## ■ Analisi Cepstrale

- ✓ Il nome deriva dall'inversione delle prime quattro lettere di "spectrum" (idem per quefrency, saphe, etc.)
- ✓ Originariamente studiato, nel 1963, per l'analisi delle vibrazioni sismiche

Imagine we were told that the log spectra was a waveform



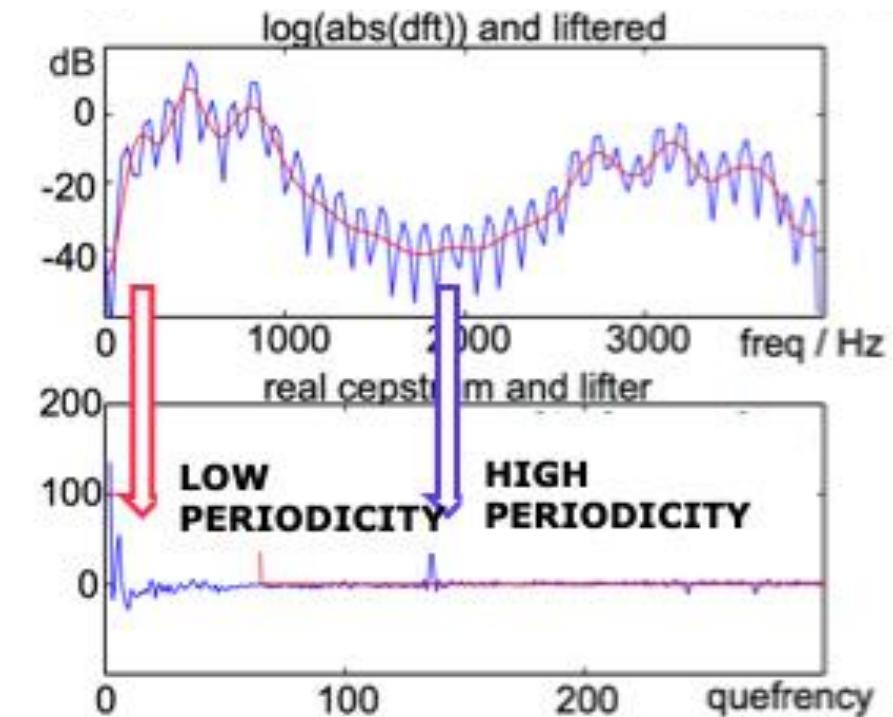
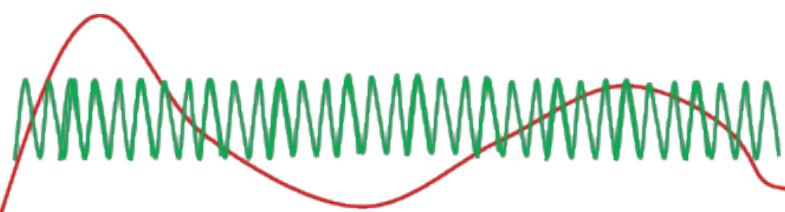
# Analisi cepstrale - procedimento

- Il *cepstrum*  $C(n)$  è l>IDFT del logaritmo dello spettro di potenza, cioè la trasformata del logaritmo della trasformata del segnale

$$C(n) = \text{IDFT}(\text{Log}(|\text{DFT}(x(n))|^2))$$

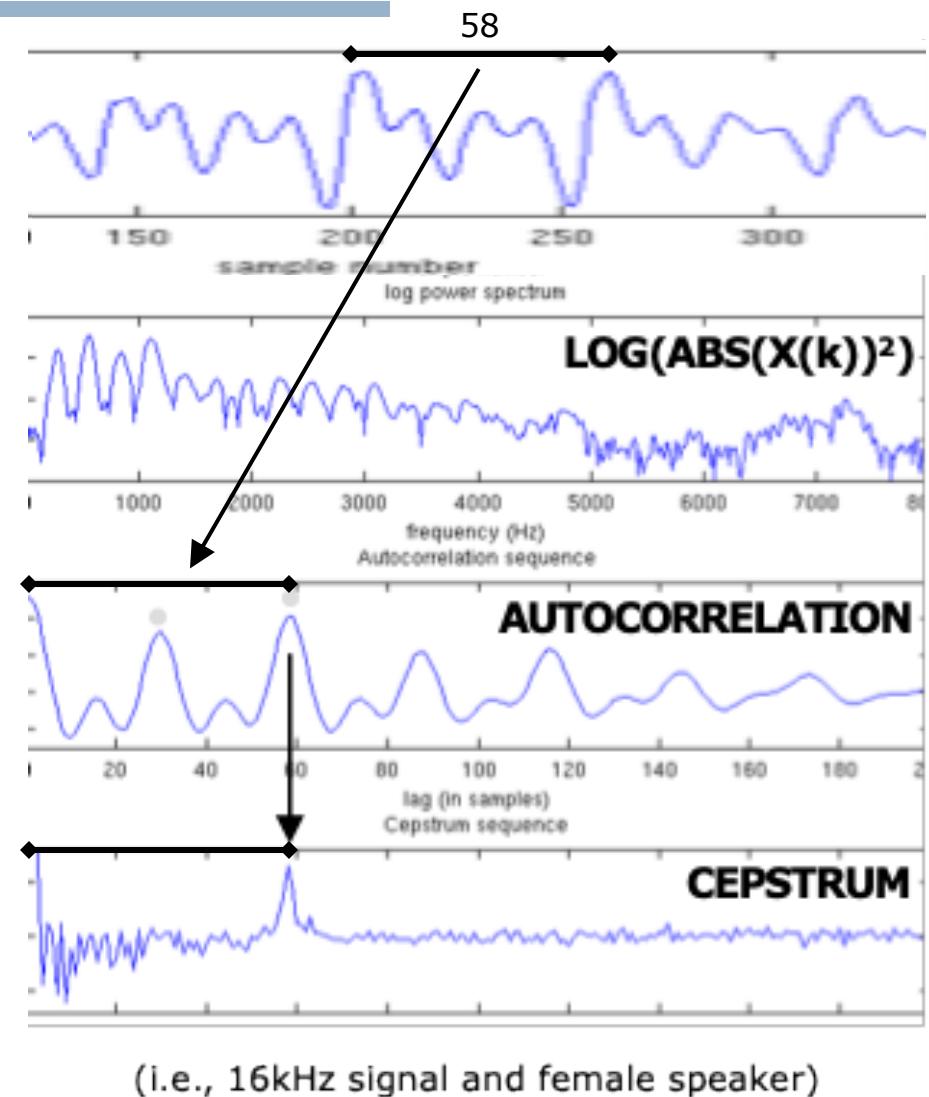
- La trasformata di Fourier (inversa) del logaritmo dello spettro

- ✓ Permette di separare
  - Le risonanze: picchi dell'inviluppo spettrale
  - Il segnale di eccitazione: lo spettro "a pettine" delle armoniche



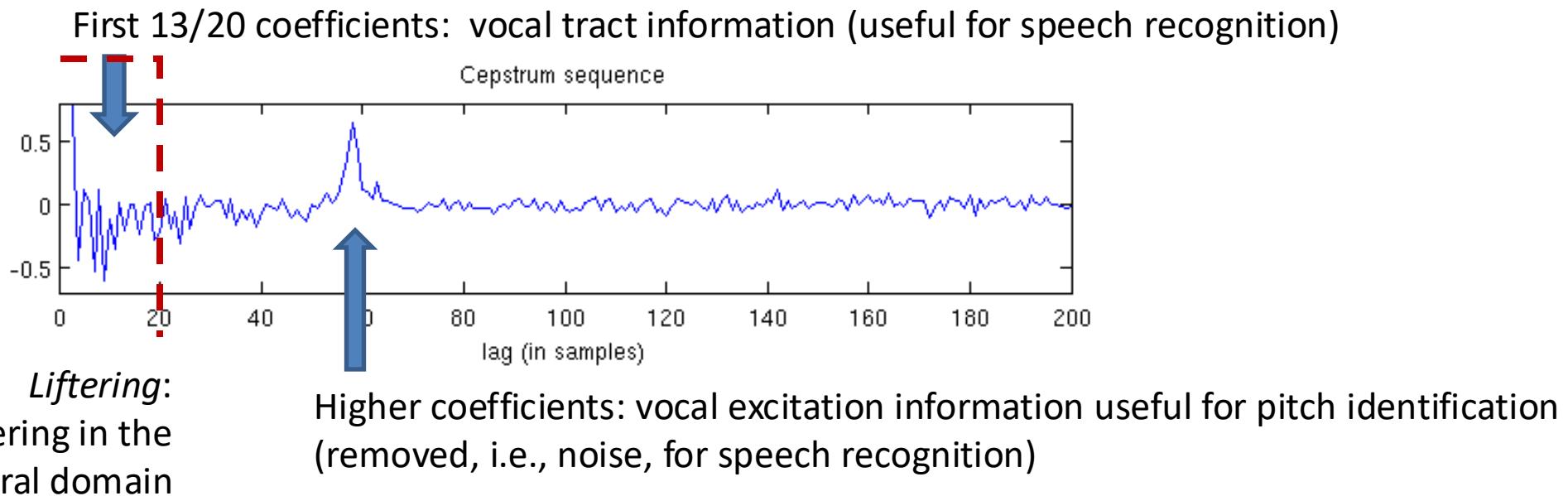
# Estrazione del pitch

- Il cepstrum è spesso utilizzato per identificare il *pitch*
  - ✓ Il picco (del pitch) nel cepstrum, è più identificabile di quello nell'autocorrelazione (che si ripete per ogni multiplo)
- L'asse delle x del cepstrum, è detto *quefrency*, ma rappresenta il tempo/campioni/ritardo
  - ✓ Il picco del pitch ad un "ritardo" di 58 campioni, con  $F_s=16\text{kHz}$ , corrisponde a  $(16000/58) = 275\text{Hz}$



# Analisi cepstrale - dettaglio

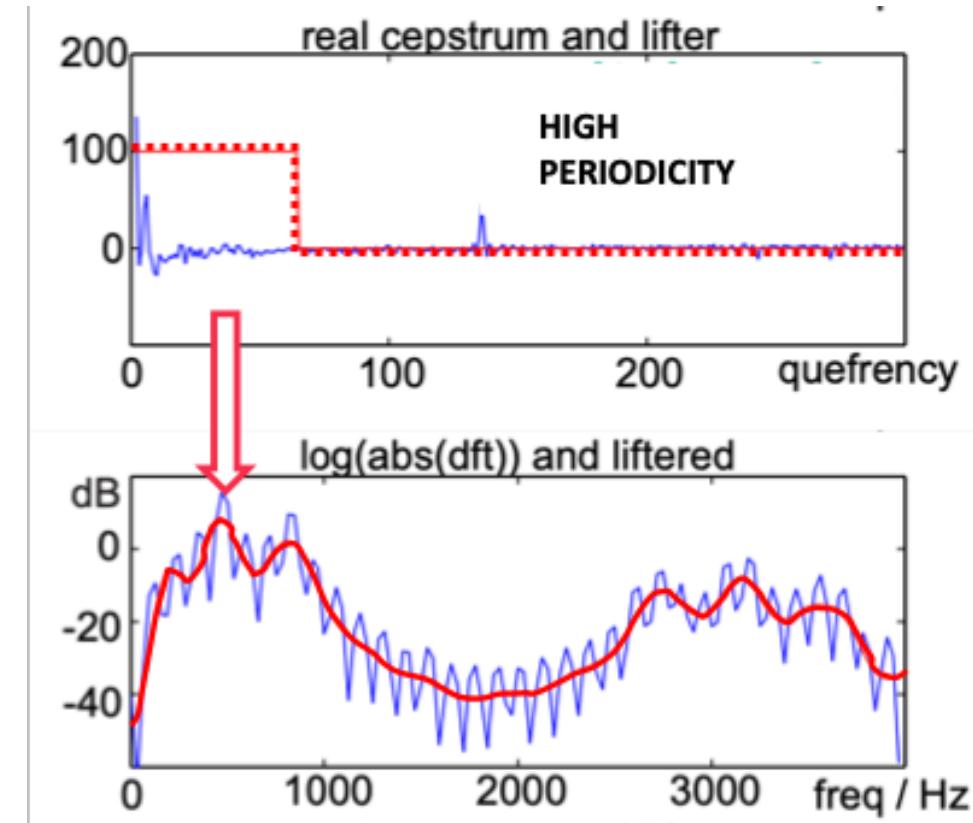
- Separazione lineare delle informazioni su sorgente e filtro
  - ✓ Nel dominio del logaritmo dello spettro di potenza il contributo della *sorgente* (l'eccitazione del tratto vocale) e del *filtro* (il tratto vocale) diventano additivi (invece che moltiplicativi) e si collocano in regioni separate del cepstrum



# Lifting (i.e. filtering)

- Nel dominio cepstrale il filtering è chiamato "liftering"
- Il filtraggio (passa basso) del cepstrum
  - ✓ Permette di ottenere (invertendo la DFT) l'inviluppo dello spettro di potenza
  - ✓ Che, nel modello, è la *risposta in frequenza del filtro*

N.B. L'operazione di estrazione del cepstrum dallo spettro non è invertibile perché si è scartata la fase (invertendola si ottiene solo il modulo dello spettro)



# Cepstrum as deconvolution

$$\begin{array}{c} x(n) = h(n) * e(n) \\ \downarrow \qquad \downarrow \\ c[n] = c_H[n] + c_E[n] \end{array}$$

- Secondo il modello sorgente + filtro
  - ✓ Il segnale è la convoluzione di segnale di sorgente e filtro:  $x(n) = h(n) * e(n)$
- Visto nel dominio della trasformata diventa una moltiplicazione
- Applicando il logaritmo allo spettro diventa una addizione
- La IDFT separa i due contributi in due regioni distinte della quefrency:

$$c = \mathcal{F}^{-1}(\log|X(\omega)|) = \mathcal{F}^{-1}(\log|H(\omega)|) + \mathcal{F}^{-1}(\log|E(\omega)|)$$

$$c[n] = c_H[n] + c_E[n]$$

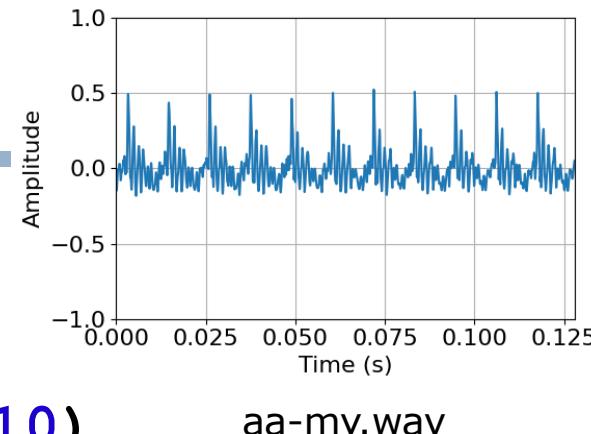
Note: Matlab real cepstrum: `y = real(ifft(log(abs(fft(x))))`; Many texts define the process as  $\text{FT} \rightarrow \text{abs}() \rightarrow \log \rightarrow \text{IFT}$ , i.e., that the cepstrum is the "inverse Fourier transform of the log-magnitude Fourier spectrum". (the difference between squaring or taking the absolute value amounts to an overall factor of 2). [from Wikipedia "Cepstrum"]

# Calcolo del cepstrum (python)

## ■ Calcolo del logaritmo dello spettro di potenza

✓ `spectrum = np.fft.fft(signal*win, NFFT)`

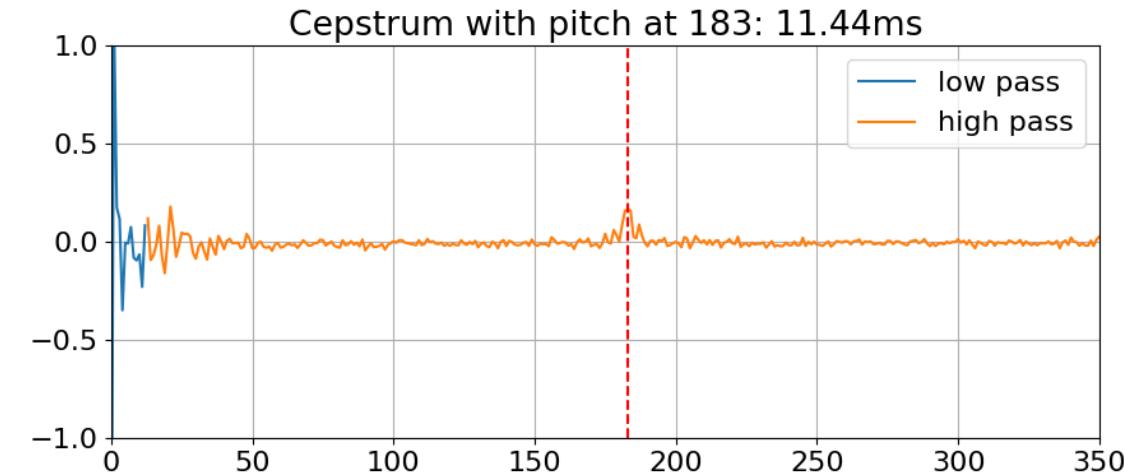
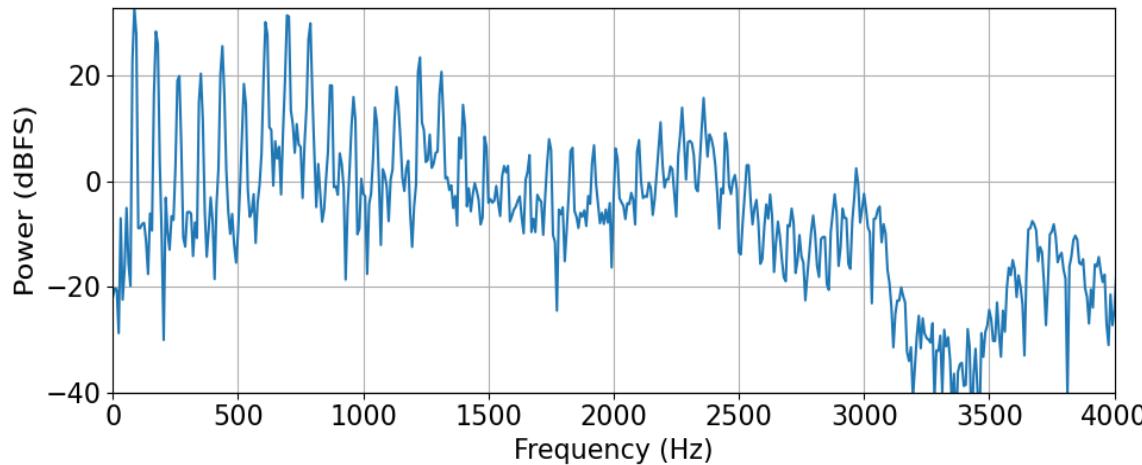
✓ `log_spectrum = np.log(np.abs(spectrum) + 1e-10)`



aa-my.wav

## ■ Calcolo del cepstrum

✓ `cepstrum = np.fft.ifft(log_spectrum)`



# Liftering (python)

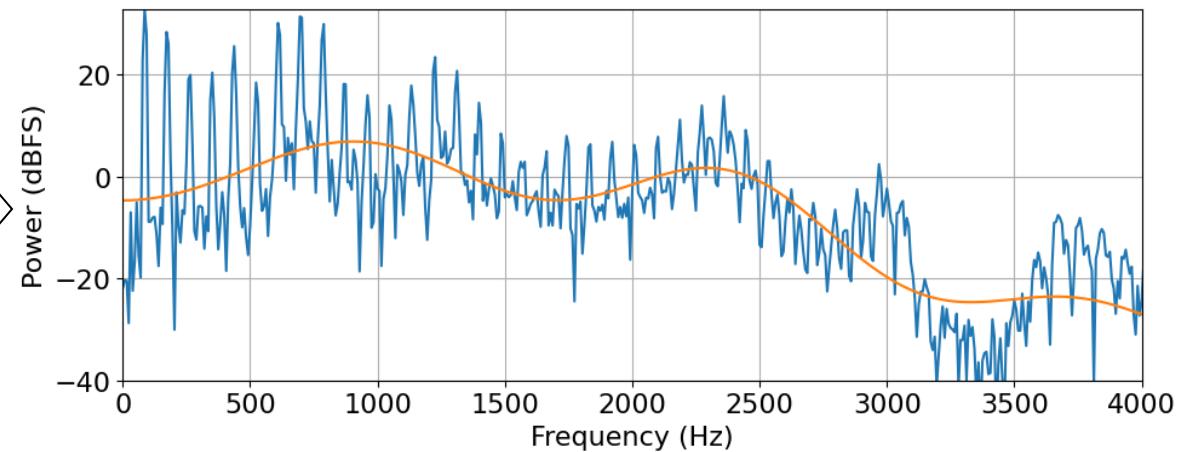
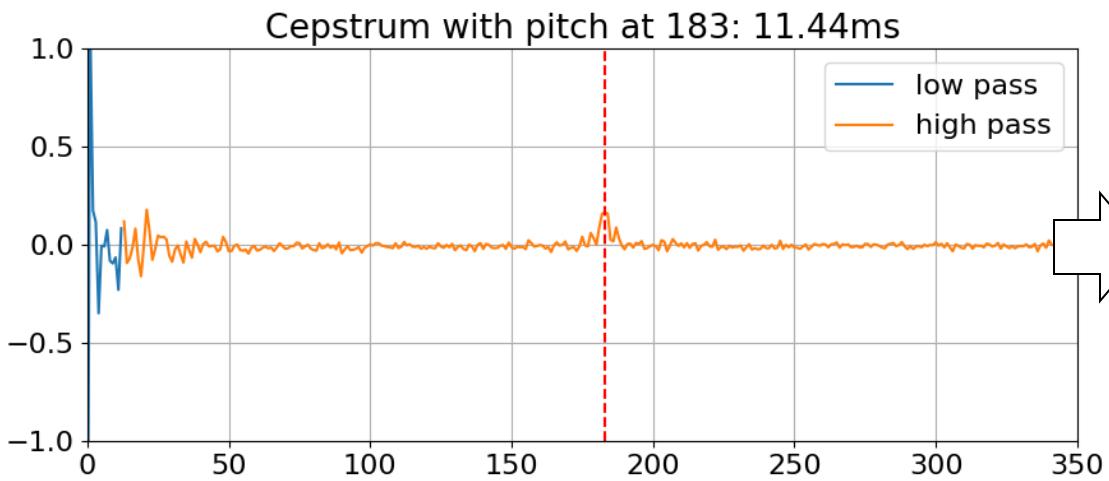
- Liftering low-pass del cepstrum con cutoff quefrency = 13

- ✓ `cutoff = 13`
  - ✓ `lifted_cepstrum[cutoff:NFFT-cutoff] = 0`

- Inversione nel dominio dello spettro

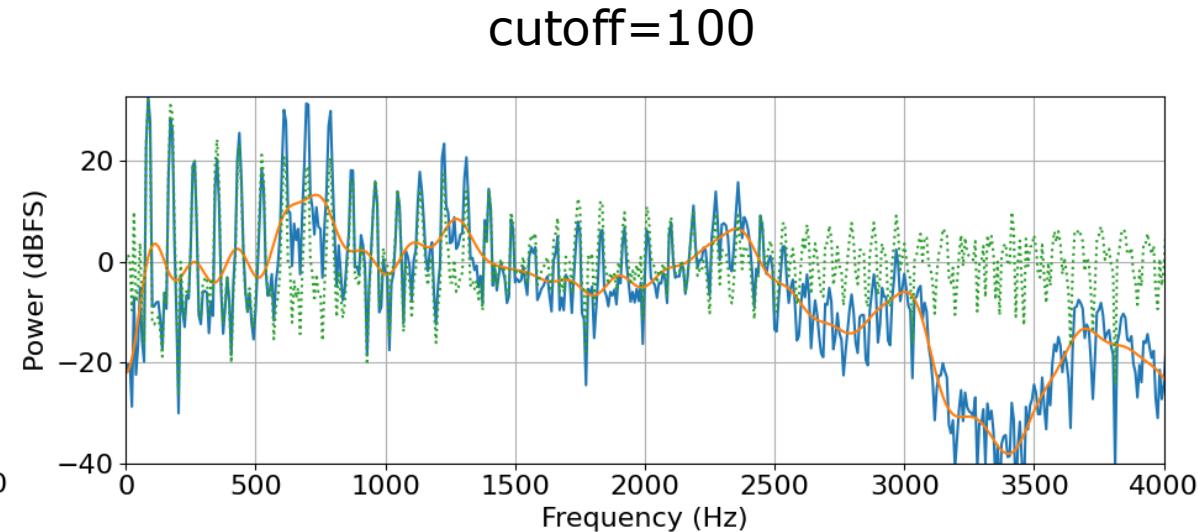
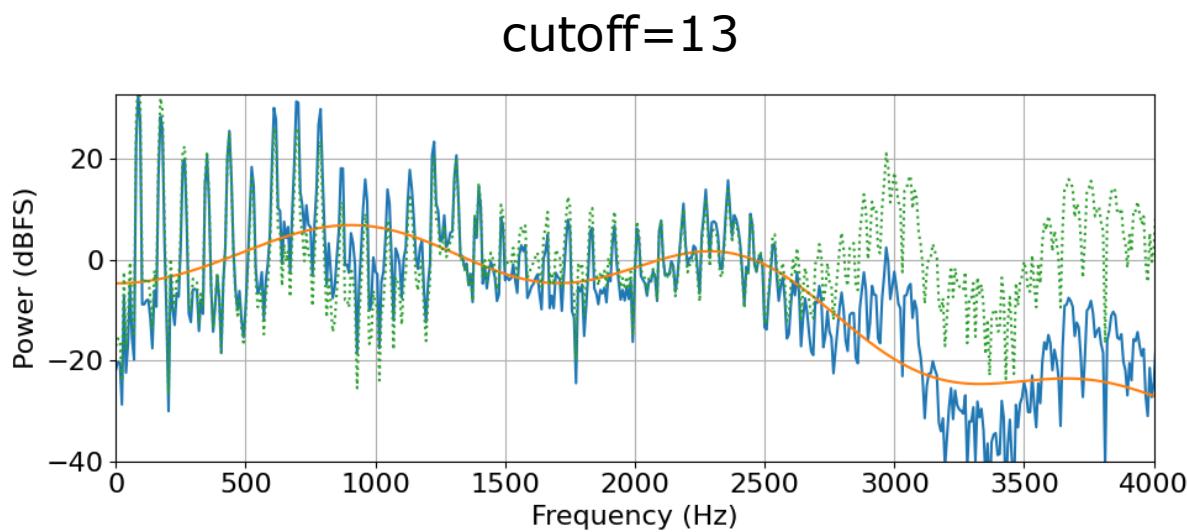
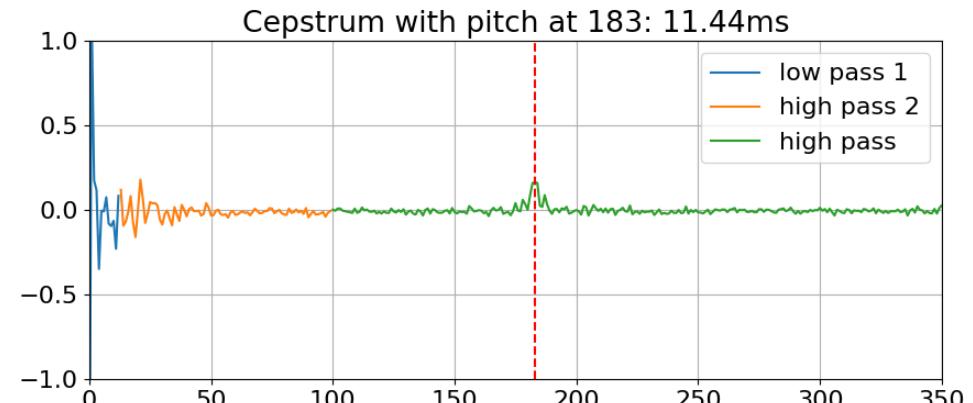
- ✓ `vocal_tract_spectrum = np.exp(np.real(np.fft.fft(lifted_cepstrum)))`

two sided spectrum

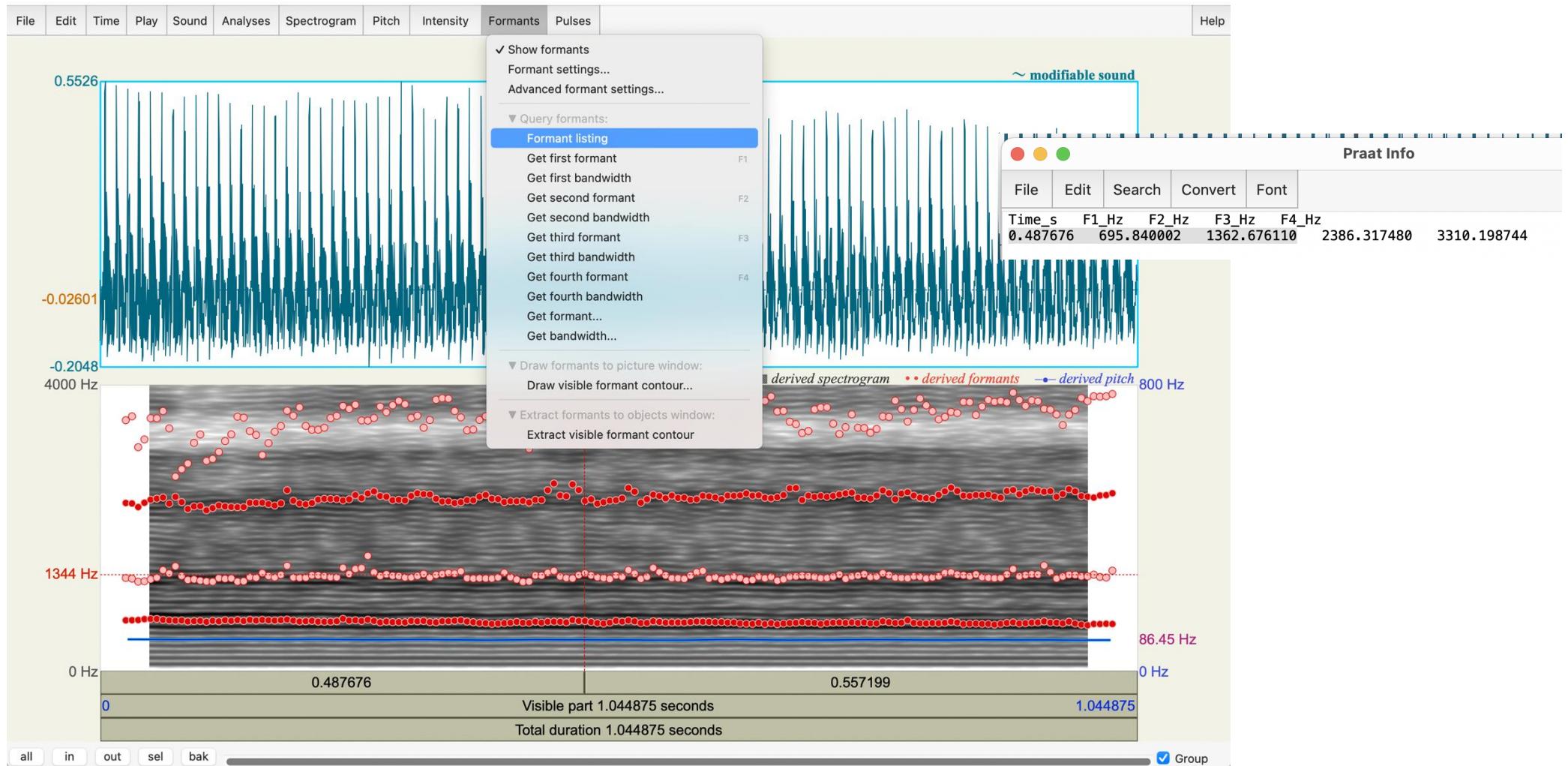


# Modello sorgente + filtro

- Spettro (blu)
  - ✓ Contributo del filtro (arancio)
  - ✓ Contributo della sorgente (verde)



# Analisi delle formanti (Praat)

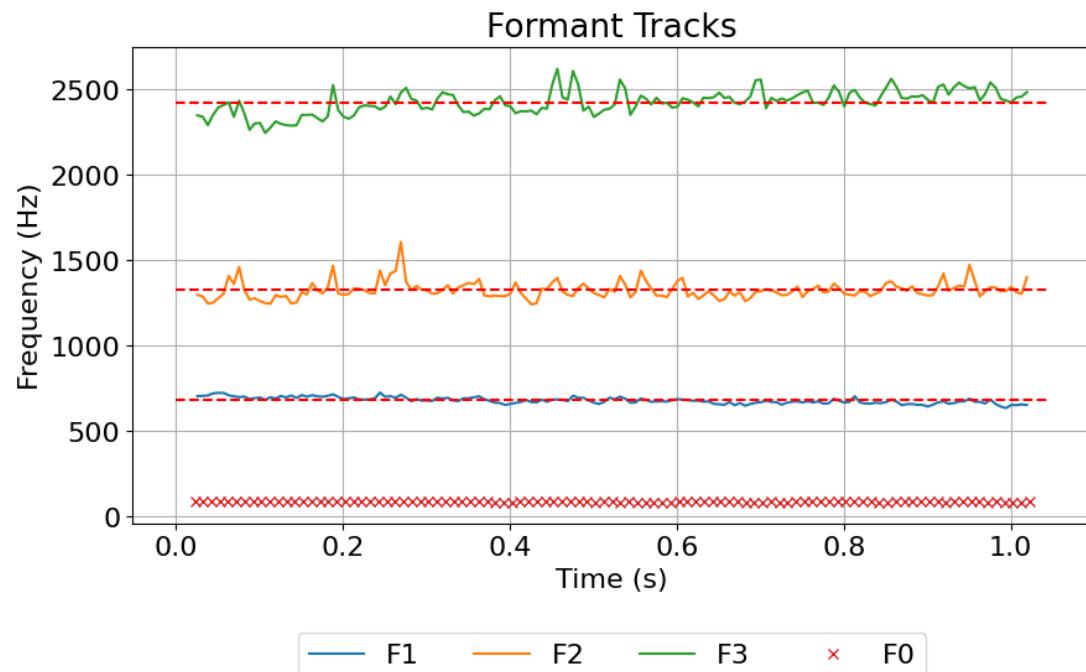


# Analisi delle formanti (Praat)

- Parselmouth implementa le funzioni di Praat

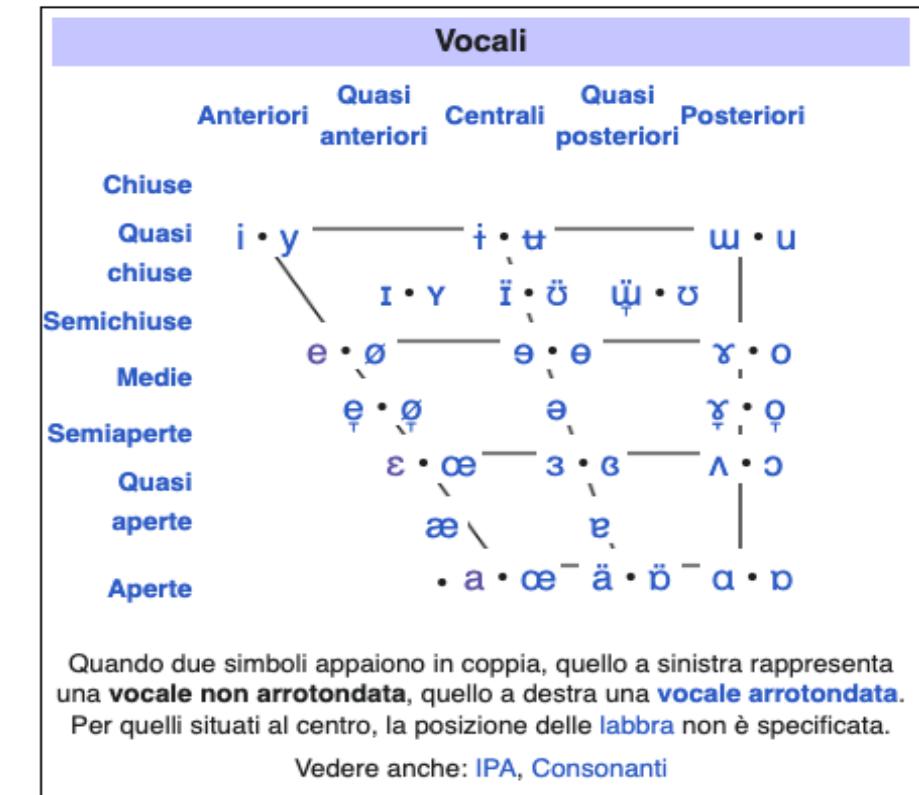
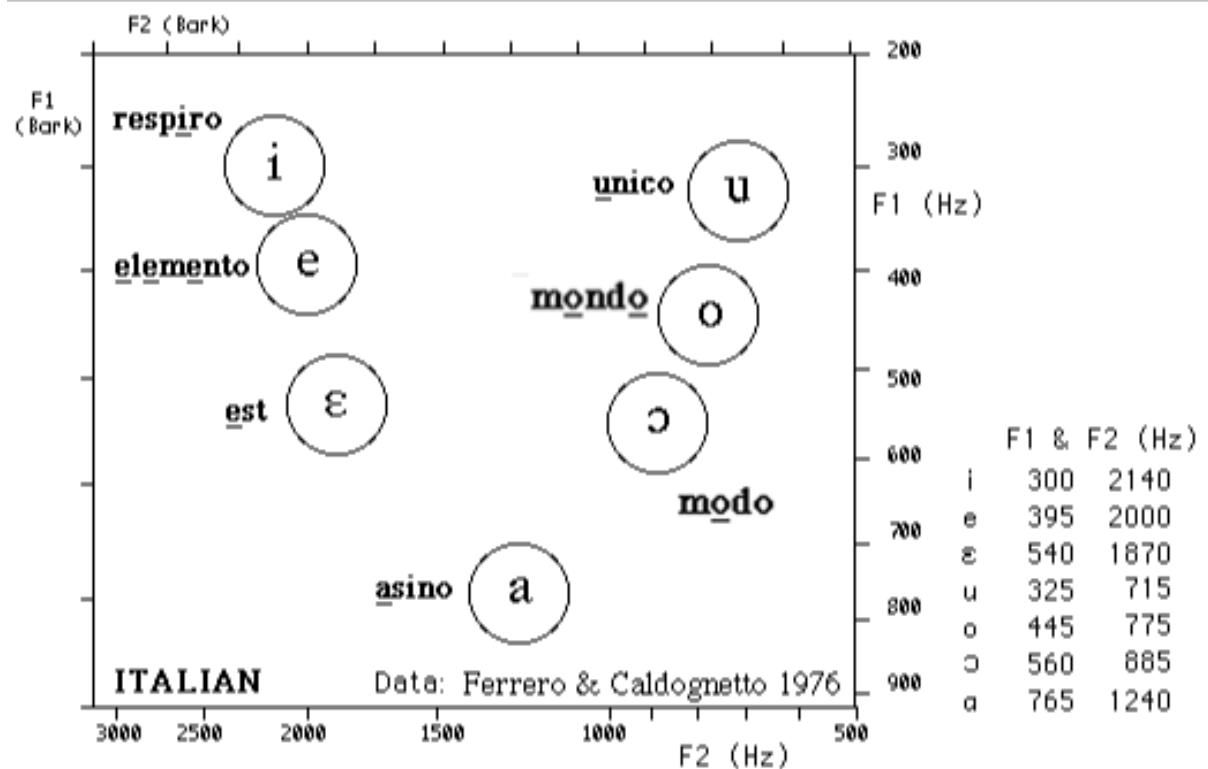
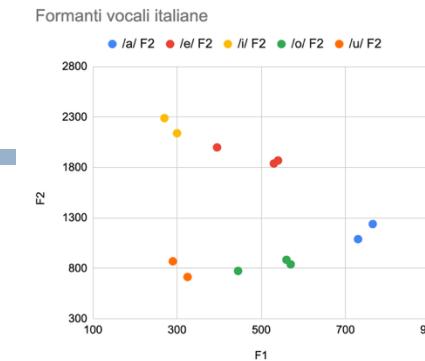
```
import parselmouth
filename = './media/aa-my.wav'
snd = parselmouth.Sound(filename) # Sound object from file
pitch = snd.to_pitch() # Pitch object
# Get pitch values
pitch_values =
    pitch.selected_array['frequency']
# Formant object
formants = snd.to_formant_burg()
# Get formants values
formant_times = formants.ts()
F1_values = [
    formants.get_value_at_time(1, t)
    for t in formant_times
]
```

Pitch: F0: 86 Hz,  
Formants:  
F1: 680 Hz,  
F2: 1326 Hz,  
F3: 2424 Hz



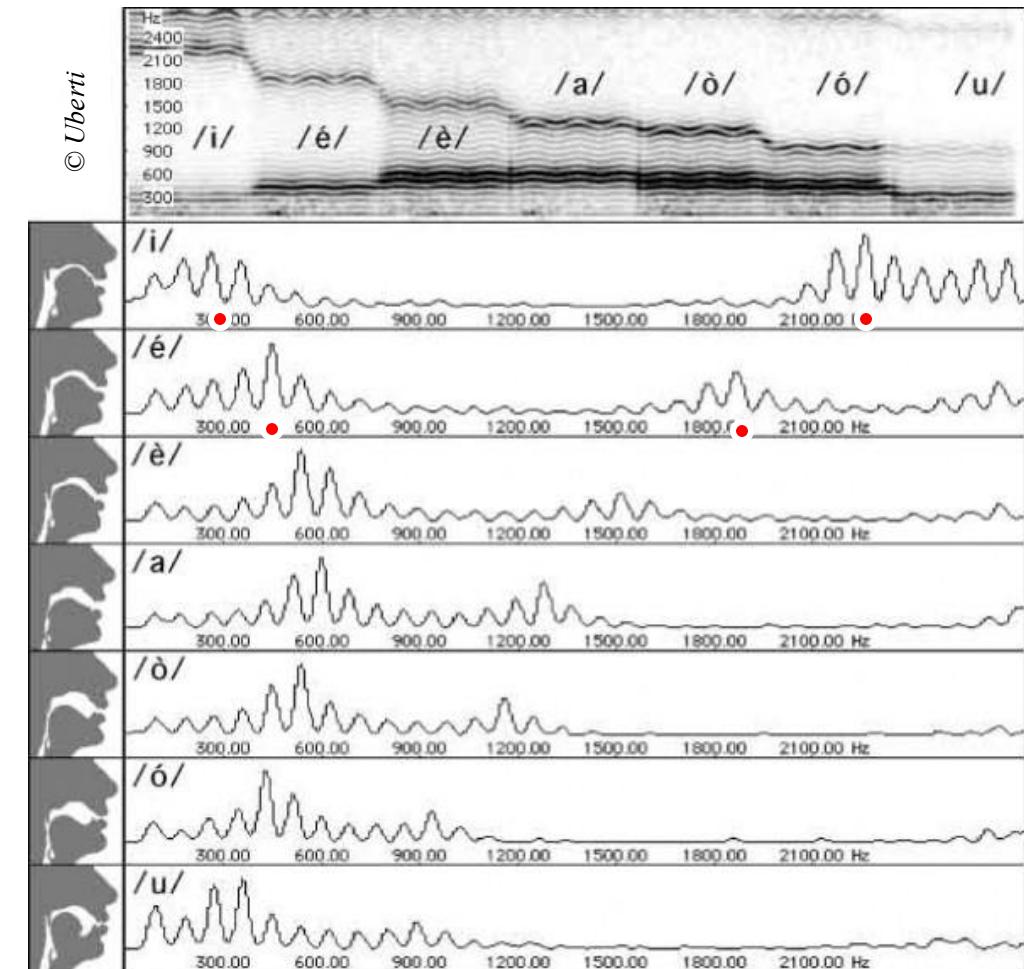
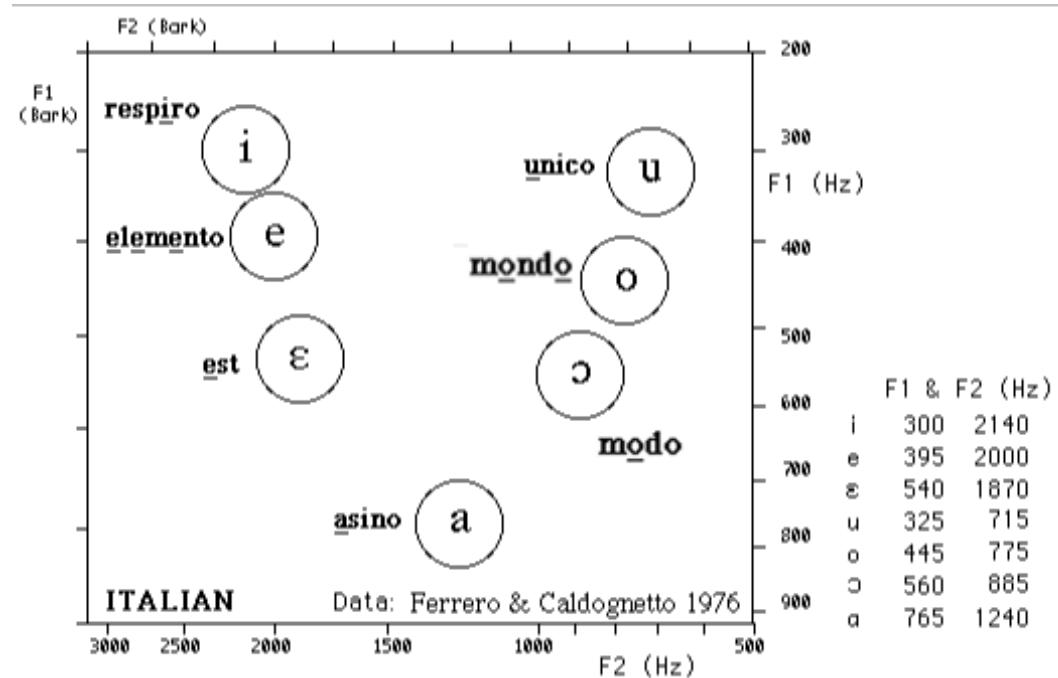
# Formanti (italiano)

- F1 – apertura (bocca) e altezza (lingua)
- F2 – posizione anteriore, inferiore (lingua)



# Formanti (italiano)

- F1 – apertura (bocca) e altezza (lingua)
- F2 – posizione anteriore, inferiore (lingua)

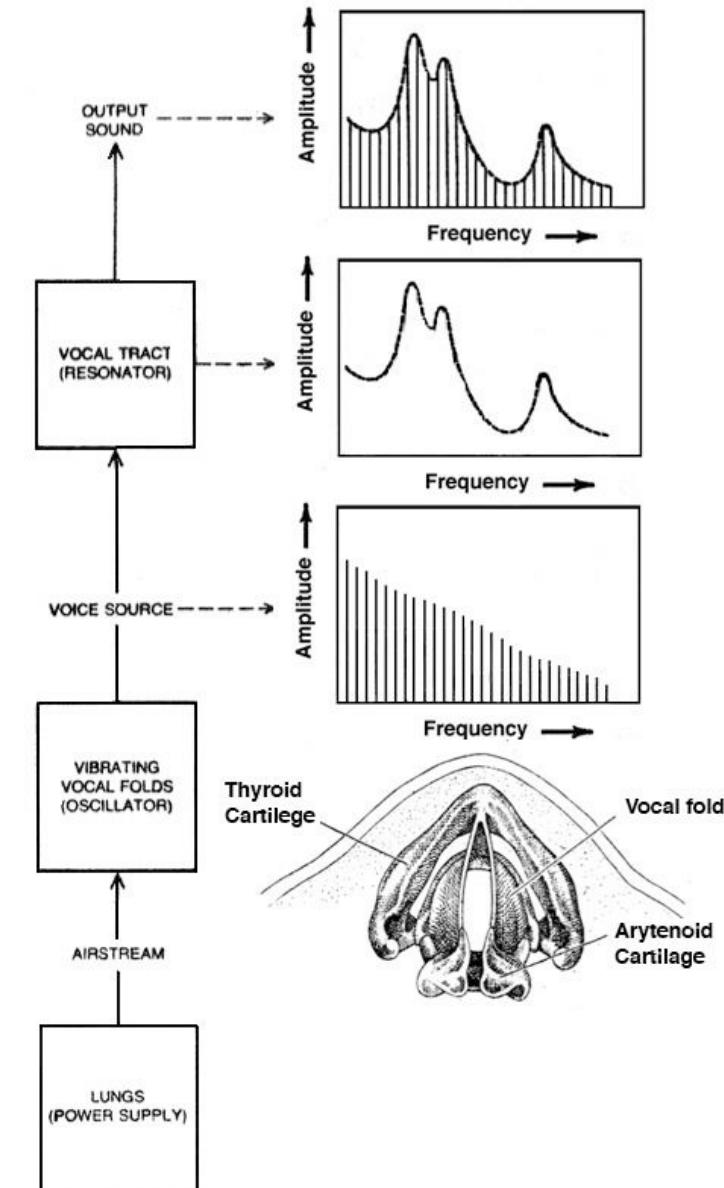
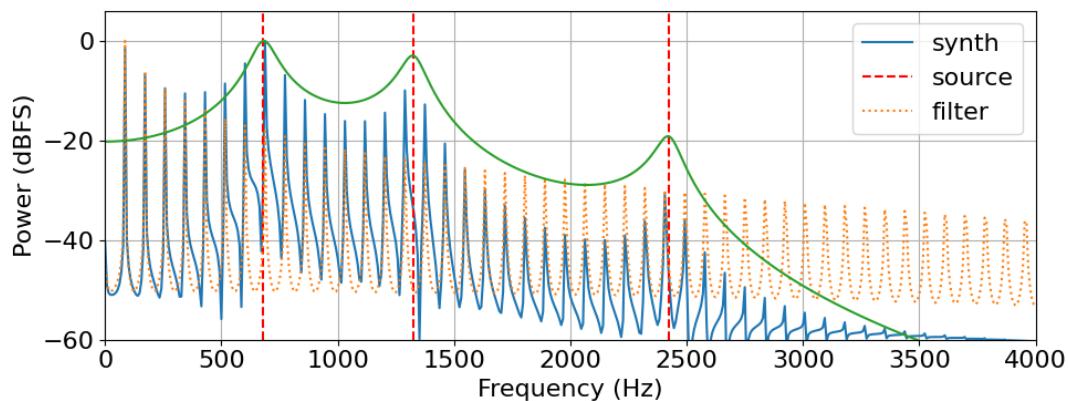
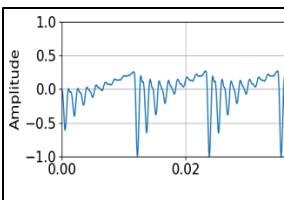


Reference: [https://www.marcotonini.org/analisi\\_vocale/SINGING%20FORMANT.html](https://www.marcotonini.org/analisi_vocale/SINGING%20FORMANT.html)

# Sintesi sottrattiva

- Generazione eccitazione/sorgente (F0)
  - ✓ `source=signal.sawtooth(2*np.pi*f0*t)`
- Generazione filtro (F1, F2, F3)
  - ✓ `formants=[F1,F2,F3]`
  - ✓ `b,a=create_formant_filter(formants, sr)`
- Convoluzione sorgente filtro
  - ✓ `output=signal.lfilter(b,a,source)`

aa-my.synth.wav

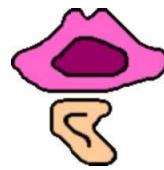
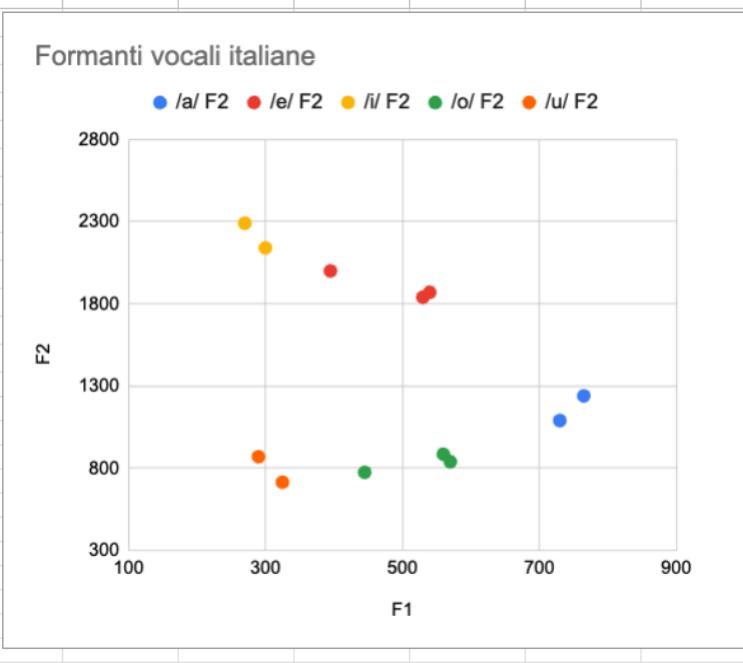


# Identificazione delle formanti

- Registrazione segnale vocale
- Analisi in frequenza
  - ✓ Identificazione delle formanti

GSHEET

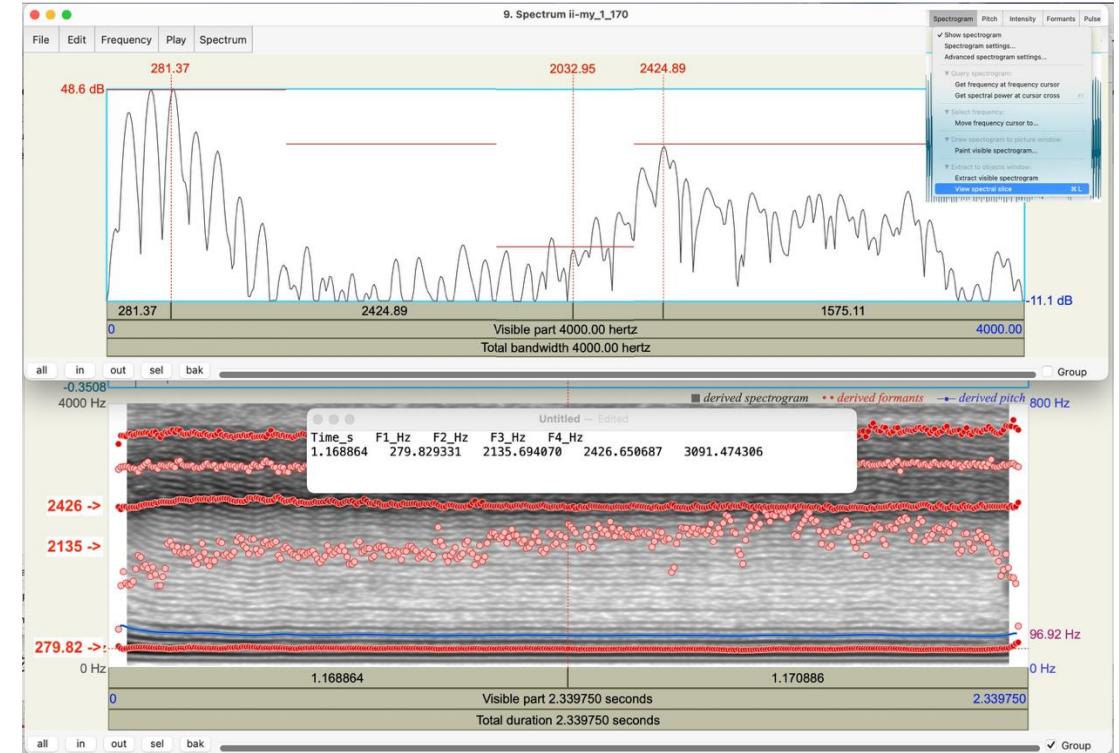
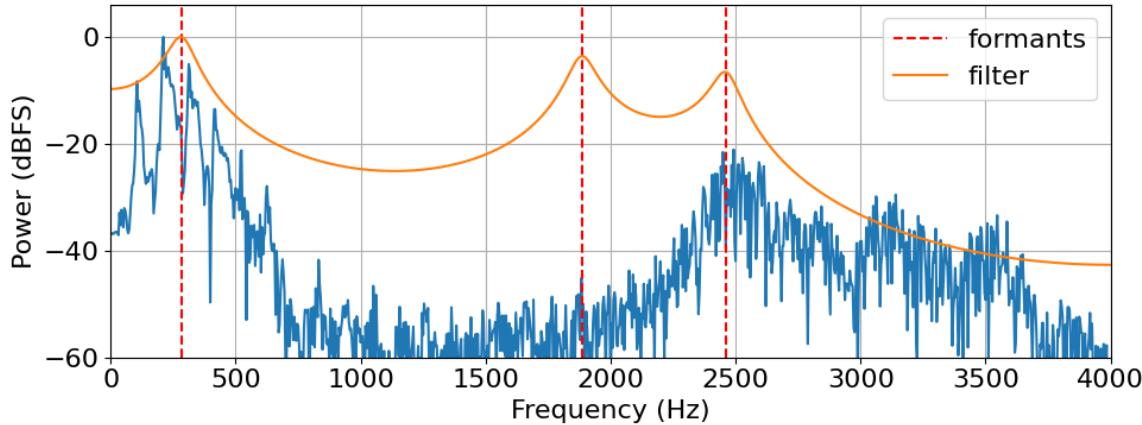
	A	B	C	D	E	F	G	H	I	J	K	L	M
1	F1	/a/ F2	/e/ F2	/i/ F2	/o/ F2	/u/ F2							
2	730	1090											
3	530		1840										
4	270			2290									
5	570				840								
6	290					870							
7	300			2140									
8	395		2000										
9	540		1870										
10	445			775									
11	560			885									
12	325				715								
13	765	1240											
14													
15													
16													
17													
18													
19													
20													
21													
22													
23													
24													
25													



PRAAT

# Identificazione delle formanti

- Registrazione segnale vocale
- Analisi in frequenza
  - ✓ Praat
  - ✓ Colab notebook

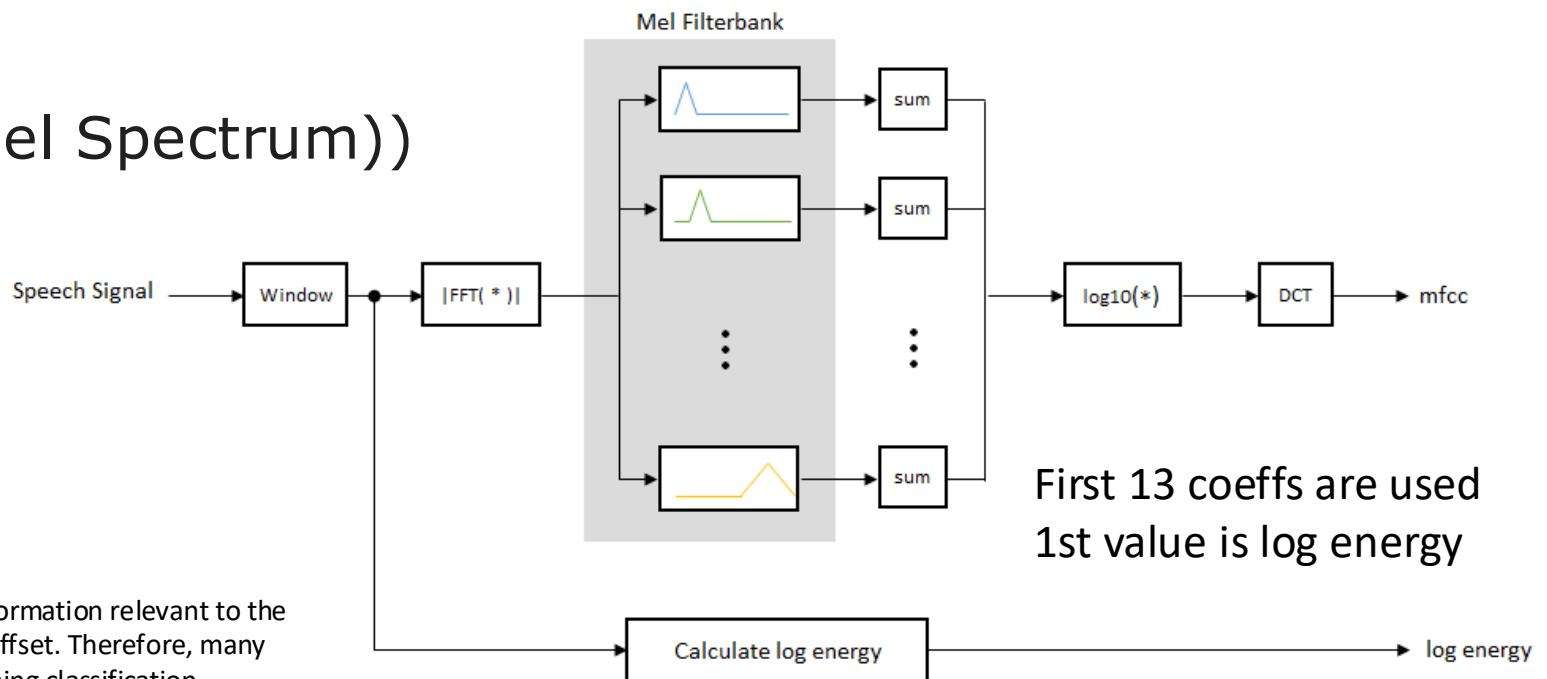


# Mel Frequency Cepstral Coefficients

- Rappresentazione robusta, molto compatta e percettiva dell'inviluppo dello spettro (una sorta di orecchio elettronico)
  - ✓ Riassume le caratteristiche utili per distinguere i suoni come farebbe l'orecchio umano
- Procedimento
  - ✓  $\text{MFCC} = \text{DCT}(\text{Log}(\text{Mel Spectrum}))$
  - ✓ DCT invece di DFT

You can think of the DCT as a “real” version of the DFT that also has a better energy compaction properties

The very first MFCC, the 0th coefficient, does not convey information relevant to the overall shape of the spectrum. It only conveys a constant offset. Therefore, many practitioners will discard the first MFCC when performing classification.

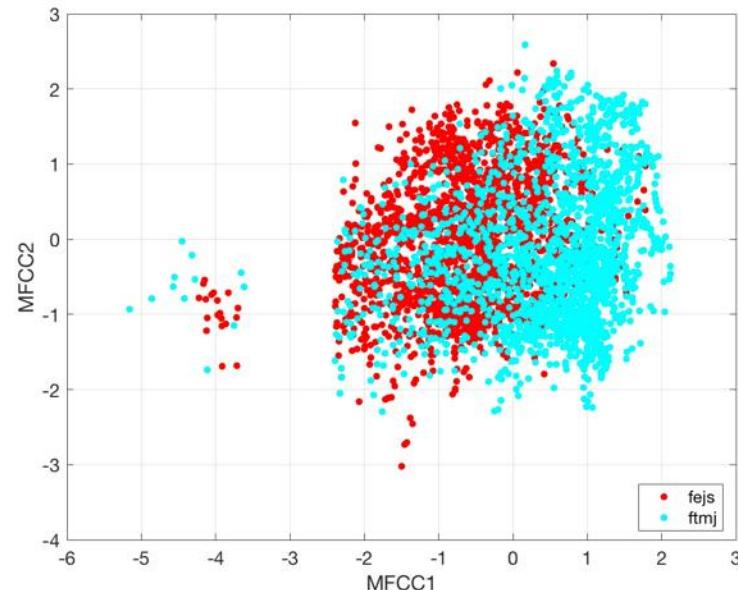


# MFCC speaker recognition

- Identify speaker based on pitch and 13 MFCCs
  - ✓ Extracted 10 segments from 10 speakers
    - <http://www.speech.cs.cmu.edu/databases/an4/>
  - ✓ Compute pitch and 13 MFCC of voiced frames
    - Frames of 30ms with 75% overlap
    - Exclude silence and unvoiced
  - ✓ First 0th MFCC coeff. replaced by pitch

Example MFCC1 and MFCC2 for  
fejs and ftmj speakers

[an4-reduced-database/ftmj/cen3-ftmj-b.flac](#)  
[an4-reduced-database/fejs/cen3-fejs-b.flac](#)



Note: <https://it.mathworks.com/help/audio/examples/speaker-identification-using-pitch-and-mfcc.html>

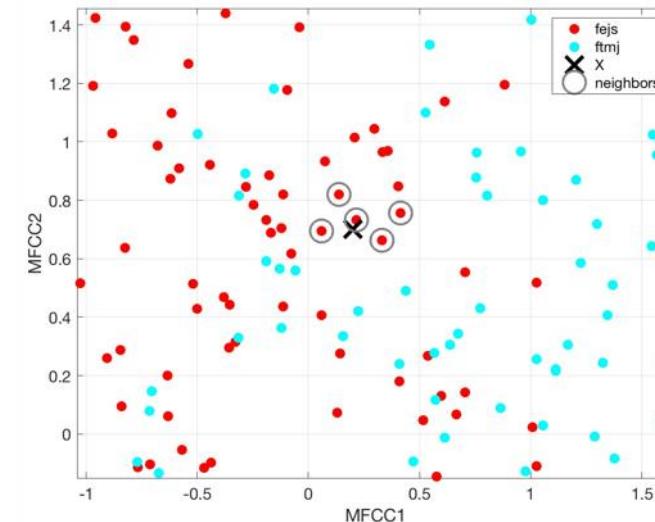
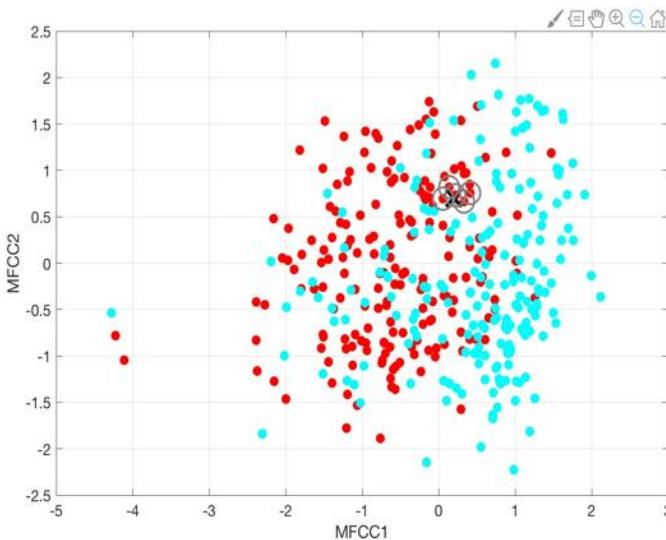
# MFCC speaker recognition

## ■ k-nearest neighbors algorithm

- ✓ Given the k closest training examples of an unknown sample
- ✓ The sample is classified by a majority vote of its neighbors

```
from sklearn.neighbors import KNeighborsClassifier  
neigh = KNeighborsClassifier(n_neighbors=5)  
neigh.fit(features, labels)  
score = neigh.score(test_features, test_labels)
```

Return the mean accuracy on the given test data and labels.



Note: <https://it.mathworks.com/help/audio/examples/speaker-identification-using-pitch-and-mfcc.html>

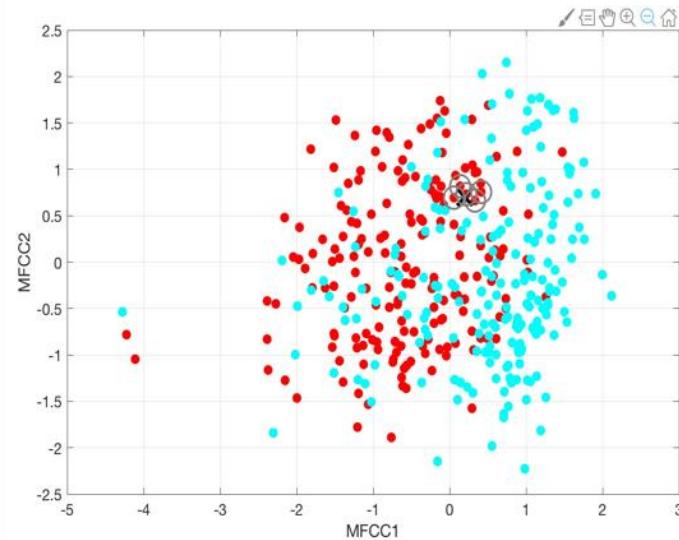
# KNN classifier with MFCC

## ■ Testing

- ✓ Predicted speaker corresponds to the most frequently occurring label
- ✓ *Prediction confidence:* the frequency of prediction of the label divided by the total number of (voiced) frames in the file

### File, prediction, confidence

fejs/an36-fejs-b.flac	fejs	91%
fmjd/cen4-fmjd-b.flac	fmjd	87%
msjr/an354-msjr-b.flac	msjr	73%
msmn/cen8-msmn-b.flac	msmn	73%



Example MFCC1 and MFCC2 for fejs and fmjd speakers

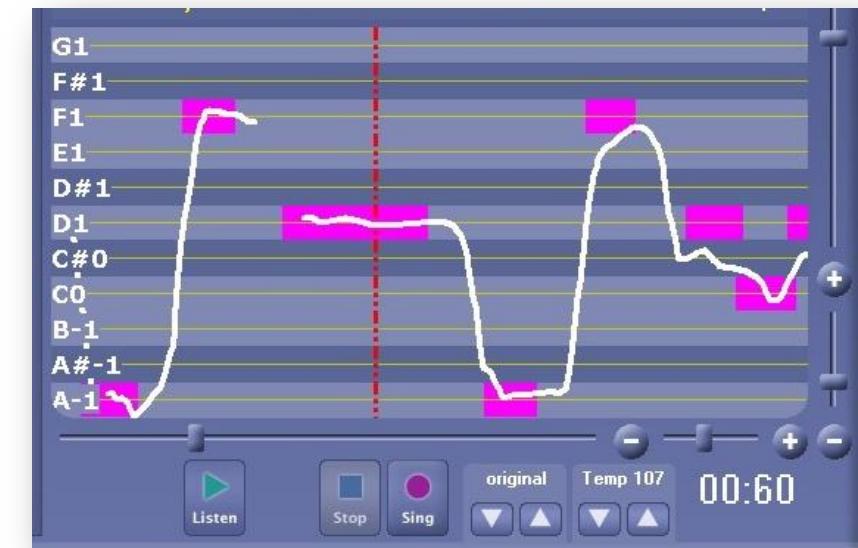
Matlab script: knn\_speaker\_classifier\_mfcc

---

## AUTOTUNE e ANALISI DEL PITCH

# Effetto Autotune

- Utilizzo di tecniche di **intonazione forzata e trasposizione del pitch**
- Algoritmo
  - ✓ Stima del pitch su porzioni di 10-40 ms
  - ✓ Definizione del pitch target
    - La nota più vicina o quella desiderata
  - ✓ Modifica dell'intonazione
    - Variazione pitch ma stessa durata



# Effetto Autotune

---

- Cheer effect – 1998 album "Believe"
  - ✓ Autotune con parametri "alterati" per non essere una correzione ma diventare un effetto (e.g., tempo di "attacco" immediato, timbro sintetico, robotizzazione)
  
- Plugins
  - ✓ Antares Auto-Tune software (Cheer 1998)
  - ✓ Melodyne (commerciale)
  - ✓ Graillon 3 (versione free)



Reference: <https://www.youtube.com/watch?v=nZXRV4MezEw>

---

# Scala musicale

- La scala musicale è definita come 7 note con una determinata relazione di altezza (pitch) a partire da una nota "tonica"
- Esempio: scala di Do maggiore
  - ✓ Do – Re – Mi – Fa – Sol – La – Si –
  - ✓ T T S T T T S
- Esempio: scala di La<sub>b</sub> maggiore
  - ✓ La<sub>b</sub> - Si<sub>b</sub> - Do - Re<sub>b</sub> - Mi<sub>b</sub> - Fa – Sol

L

Analogia con  
i tasti del pianoforte

MIDI						
T	T	S	T	T	T	S

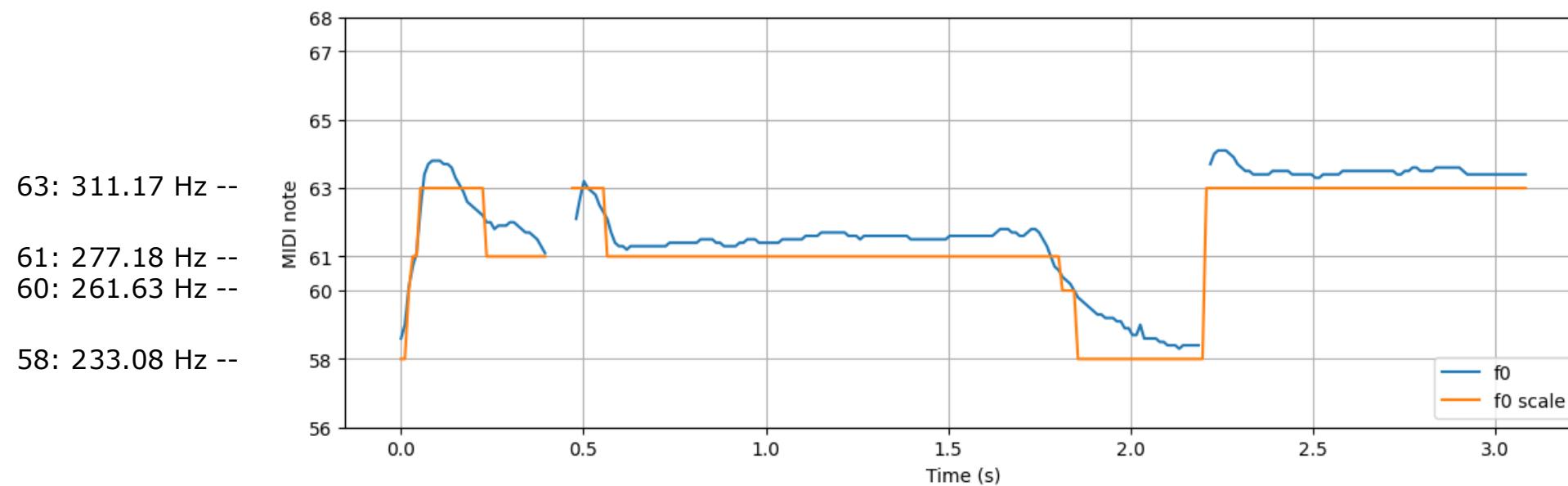


Reference: [https://it.wikipedia.org/wiki/Scala\\_maggiore](https://it.wikipedia.org/wiki/Scala_maggiore)

# Intonazione forzata

- Correzione dell'intonazione in modo da corrispondere sempre ad una nota della scala musicale usata

53	54	F3	174.01
55	56	G3	196.00
57	58	A3	220.00
59		B3	246.94
60	61	C4	261.63
62	63	D4	277.18
64		E4	329.63
65	66	F4	349.23
67	68	G4	392.00
69	70	A4	440.00
71		B4	493.88
		--	523.25

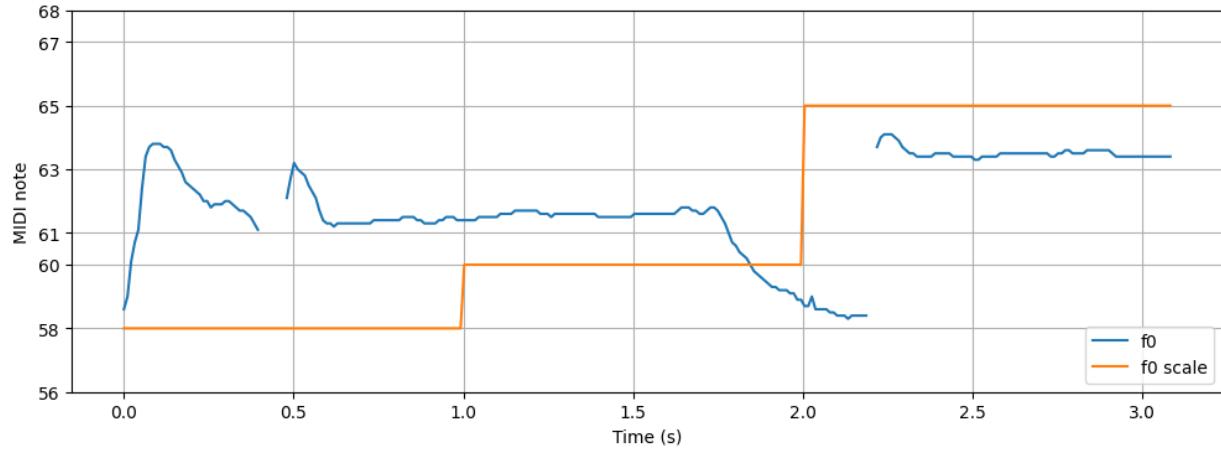


autotune-letitgo-bad.wav autotune-letitgo-bad.tuned.wav

# Effetto pitch shifting - arbitrario

- Correzione dell'intonazione secondo valori arbitrari

autotune-letitgo-bad.custom.wav

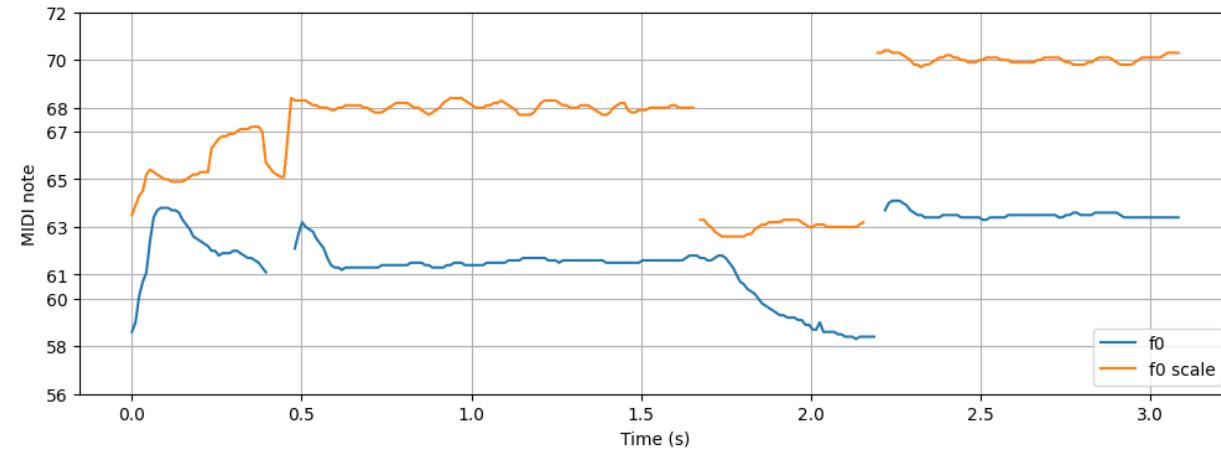


- Correzione dell'intonazione secondo valori "clonati"

autotune-letitgo-good.wav



autotune-letitgo-bad.clone.wav



# Fasi dell'autotune

---

- 1. Identificazione del pitch
- 2. Mapping del pitch originale sul nuovo pitch
  - ✓ Intonazione forzata
- 3. Trasposizione del pitch

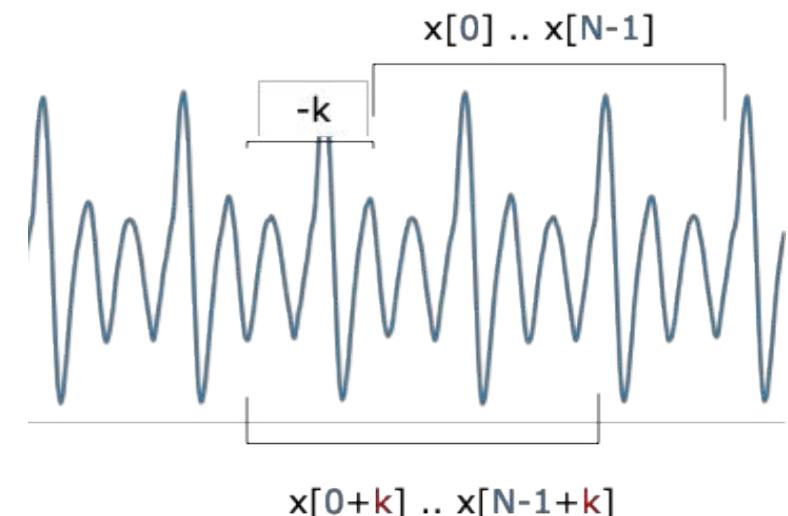
---

## Identificazione del pitch

# Identificazione del pitch (1/periodo)

- Periodo: "l'intervallo più lungo per cui il segnale si ripete uguale"
- L'identificazione del periodo consiste nel
  - selezionare un intervallo del segnale e cercare a che distanza si trova la replica più simile**
- Occorre introdurre un criterio di somiglianza (MSE)
  - ✓ Usiamo  $k < 0$  perchè guardiamo nel passato

$$MSE[k] = \frac{1}{N} \sum_{n=0}^{N-1} (x[n] - x[n+k])^2$$

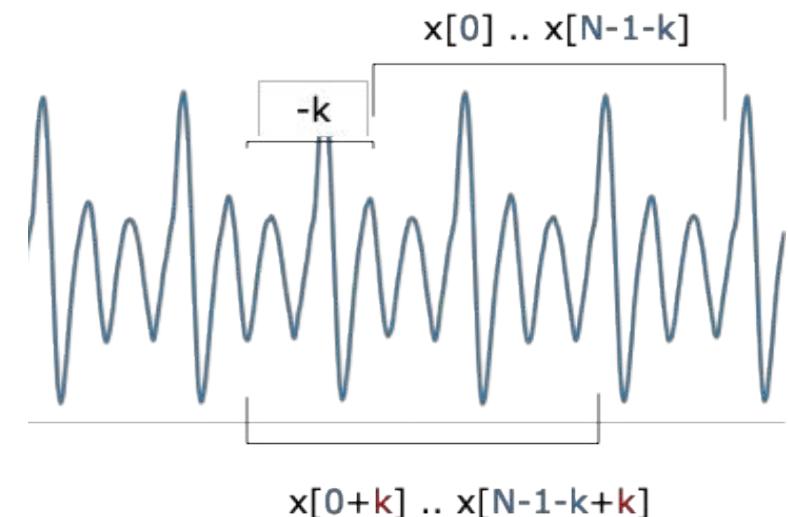


# Identificazione del pitch (1/periodo)

## ■ Metodo della differenza, MSE (corretto)

- ✓ Criterio di somiglianza / differenza: MSE
- ✓ Ritardo (-k) tra 0 e N
- ✓ Normalizzazione corretta
  - Solo per la parte di segnale che si sovrappone ( $N-k$ ) che equivale al numero di somme che vengono effettuate

$$MSE[k] = \frac{1}{N-k} \sum_{n=0}^{N-1-k} (x[n] - x[n+k])^2$$



# Ricerca MSE minimo

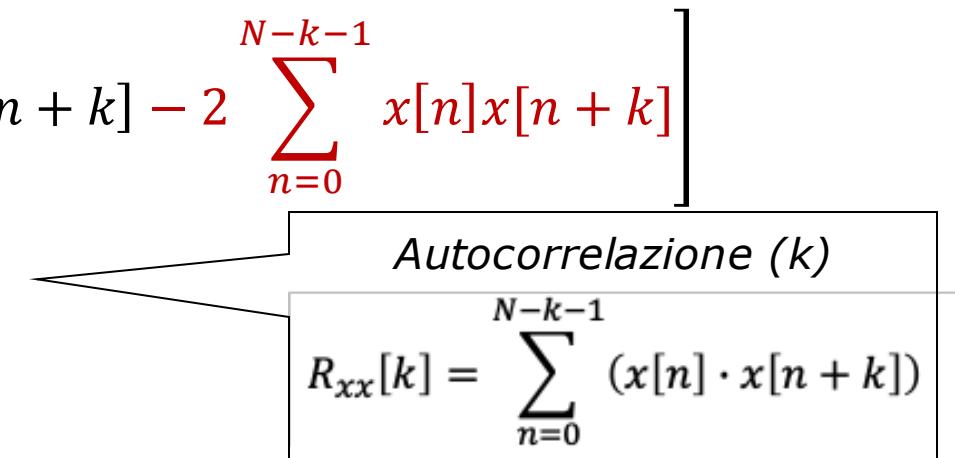
$$MSE[k] = \frac{1}{N-k} \sum_{n=0}^{N-k-1} (x[n] - x[n+k])^2$$

- Occorre trovare il valore di  $k$  che minimizza l'MSE
  - ✓ Cioè il valore di  $k$  per cui la derivata dell'MSE si annulla
- Espandendo i calcoli e considerando un segnale stazionario otteniamo che l'MSE equivale a ...

$$MSE[k] = \frac{1}{N-k} \sum_{n=0}^{N-k-1} (x^2[n] + x^2[n+k] - 2x[n]x[n+k])$$

$$MSE[k] = \frac{1}{N-k} \cdot \left[ \sum_{n=0}^{N-k-1} x[n]x[n] + \sum_{n=0}^{N-k-1} x[n+k]x[n+k] - 2 \sum_{n=0}^{N-k-1} x[n]x[n+k] \right]$$

$$MSE[k] = \frac{1}{N-k} \cdot [R_{xx}[0] + R_{x_k x_k}[0] - 2 \cdot R_{xx}[k]]$$



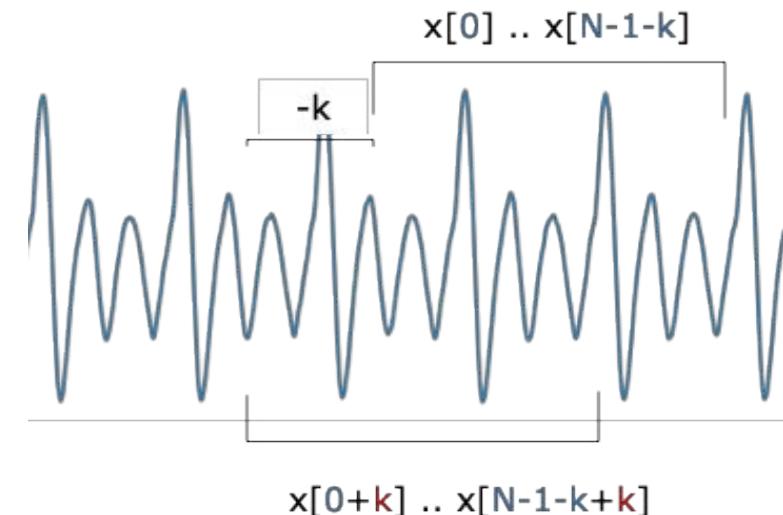
# Ricerca MSE minimo

$$R_{xx}[k] = \sum_{n=0}^{N-k-1} (x[n] \cdot x[n+k])$$

- Introducendo l'autocorrelazione di  $x$  con ritardo  $k$  come  $R_{xx}[k]$
- Se il segnale è stazionario l'autocorrelazione dipende solo dal ritardo  $k$  e non dal tempo assoluto:  $R_{xx}[0]=R_{x_k x_k}[0]$

$$MSE[k] = \frac{1}{N-k} \cdot [R_{xx}[0] + R_{x_k x_k}[0] - 2 \cdot R_{xx}[k]] = \frac{2}{N-k} \cdot [R_{xx}[0] - R_{xx}[k]]$$

- Quindi l'MSE è **minimo** quando è **massimo**  $R_{xx}[k]$
- Occorre individuare il ritardo/lag  **$k$**  per cui è massimo il prodotto tra  $x[n]$  e  $x[n+k]$



# Esempio: calcolo autocorrelazione

---

- Sia dato un segnale sig di lunghezza 10 campioni
- Dato un valore di k, e.g. -4
- Calcolare il prodotto
  - tra ciascun campione di  $x[n]$  e  $x[n+(-k)]$
- Sommare i valori ottenuti

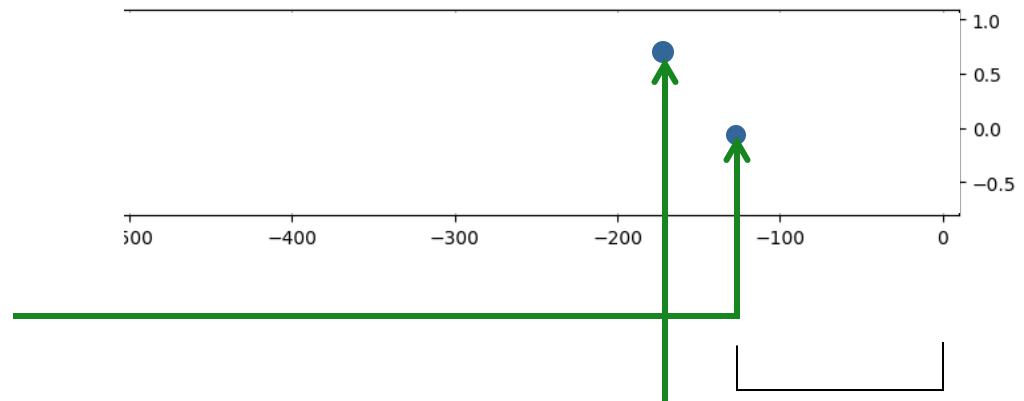
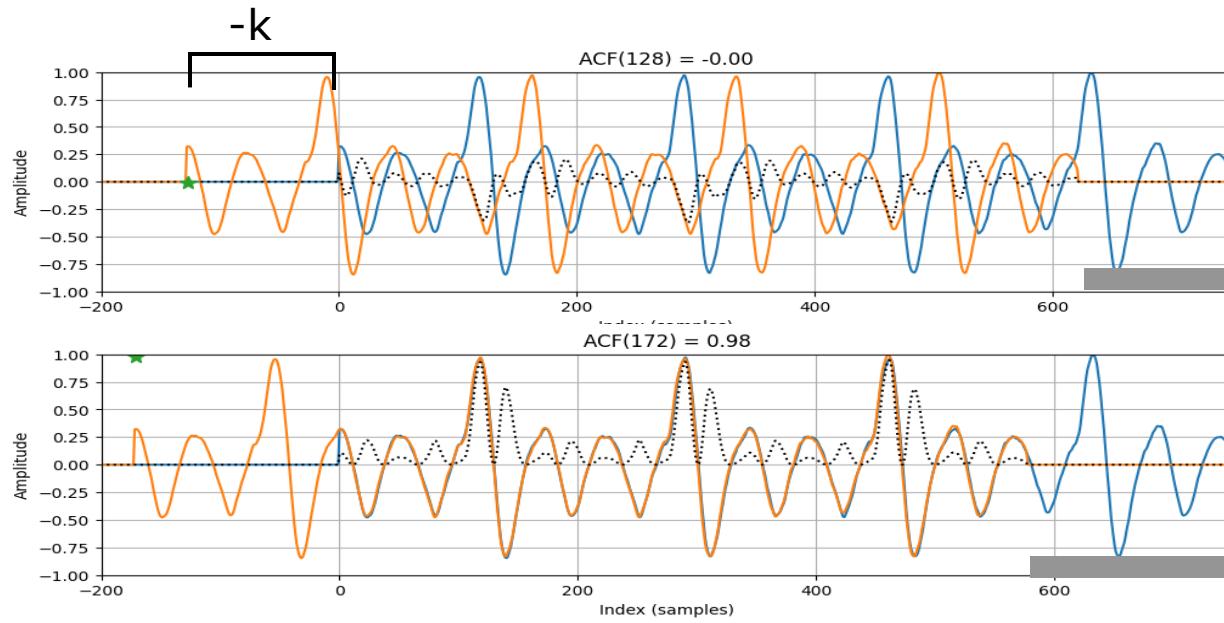
$$R_{xx}[k] = \sum_{n=0}^{N-k-1} (x[n] \cdot x[n + k])$$

```
#          01234567890123456789
# sig      -----*****      => sig[:-k]
# sig_k    -----*****----- => sig_k[k:]
# k        09876543210
```

# Autocorrelazione

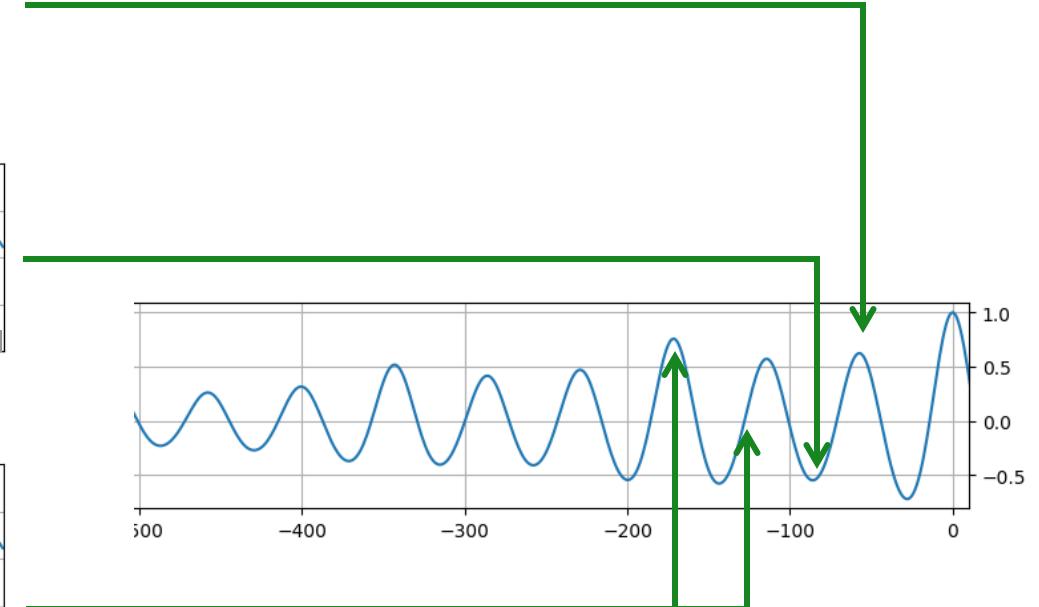
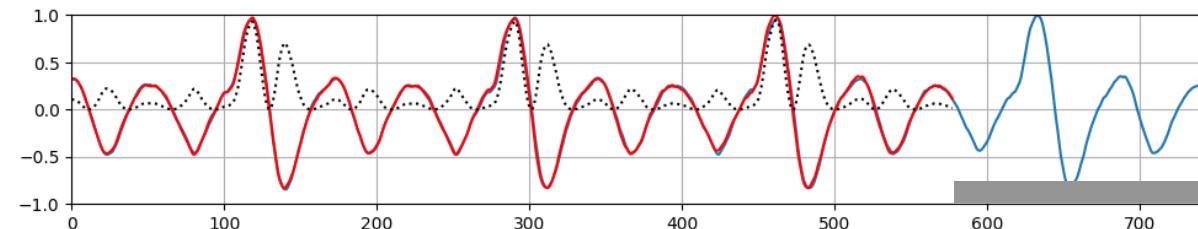
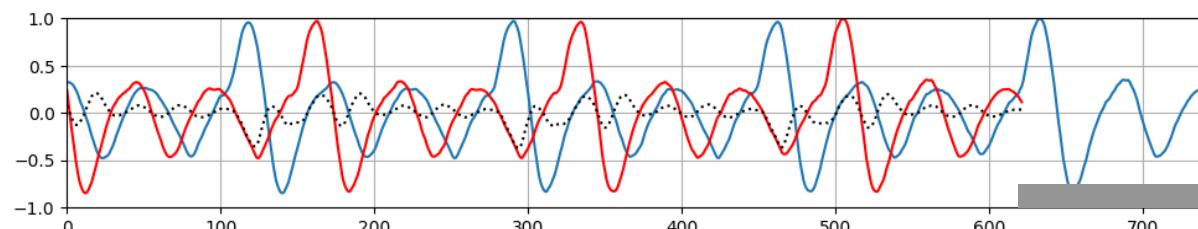
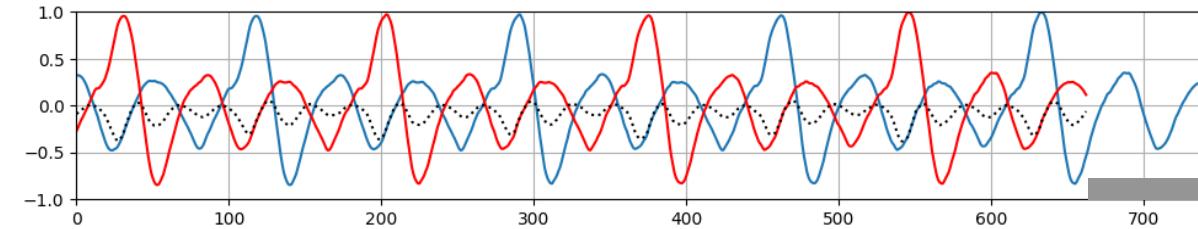
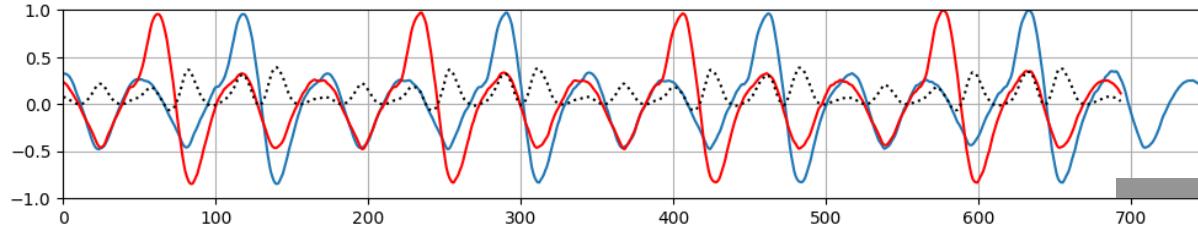
GSheet

- Calcolo dell'autocorrelazione con lag  $-k$ :  $R_{xx}[-k]$ 
  - ✓ Media del prodotto (tratteggiato) tra  $x[n]$  (blu) e  $x[n-k]$  (rosso) sulla finestra di analisi
- Da ripetere per diversi valori di  $k$



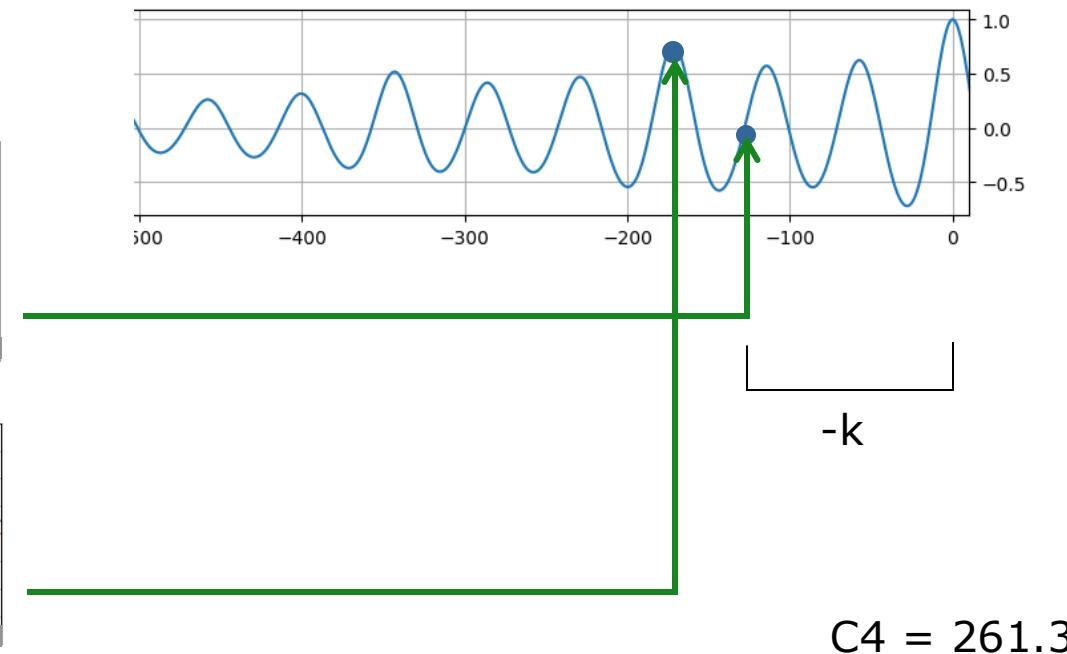
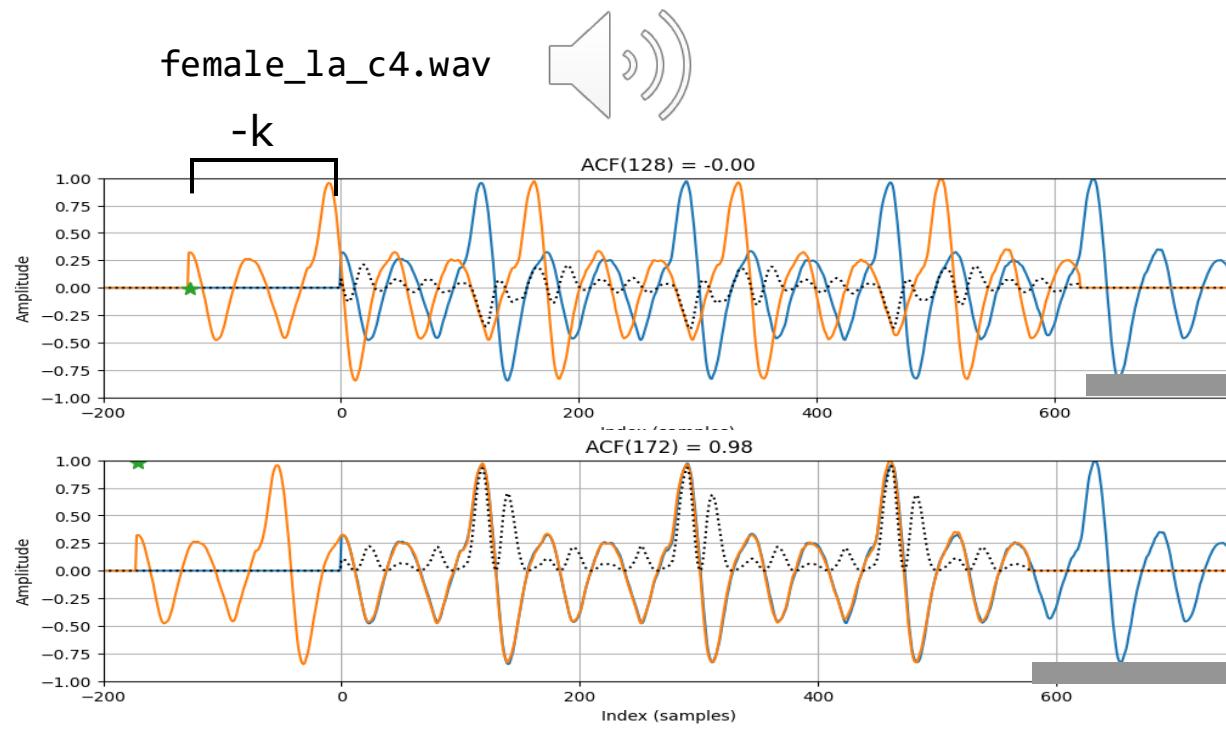
# Autocorrelazione

56, 87, 128, 172



# Autocorrelazione

- Il picco dell'autocorrelazione identifica il periodo  $T$
- Il pitch è l'inverso del periodo  $1/T$
- Esempio:  $k = 172$ ,  $fs = 44100$  Hz,  $fs \cdot 1/k = 256,39$  Hz



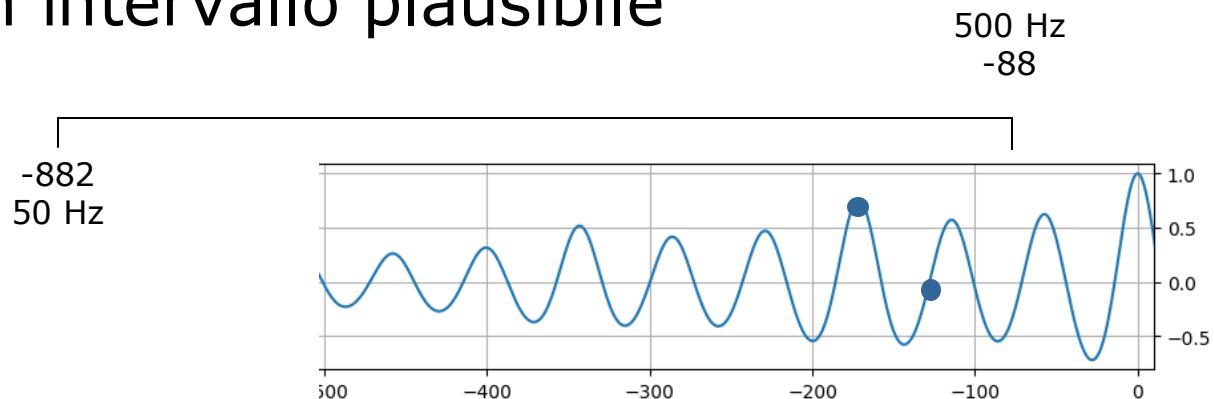
# Identificazione del pitch - caveats

---

- Spesso si preferisce lavorare con l'autocorrelazione normalizzata, così, indipendentemente dal volume, il valore di picco sarà 1
- L'algoritmo non è infallibile, anzi ...
- La presenza delle armoniche
  - ✓ Porta spesso a identificare valori che sono multipli del pitch effettivo perchè le armoniche creano picchi, talvolta più alti, a lag minori
  - ✓ E' conveniente definire un intervallo "ragionevole" in cui effettuare la ricerca del "lag" corretto
- Il segnale è spesso poco-stazionario
  - ✓ La ricerca del periodo, per essere corretta, richiede che questo si ripeta un certo numero di volte e non ci siano transitori

# Pitch range

- L'autocorrelazione ha, per definizione, il massimo in  $k=0$ 
  - ✓ Quando il segnale è moltiplicato per sè stesso
- E' buona norma non "prendere" qualsiasi massimo, ma ricercare il massimo in un intervallo plausibile
- Per la voce il pitch è
  - ✓ 50-150 Hz maschi adulti
  - ✓ 150-250 Hz femmine adulte
  - ✓ 300-500 Hz bambini
- A che ritardo corrisponde?
  - ✓  $f_0 = F_s / k \Rightarrow k = F_s / F_0$

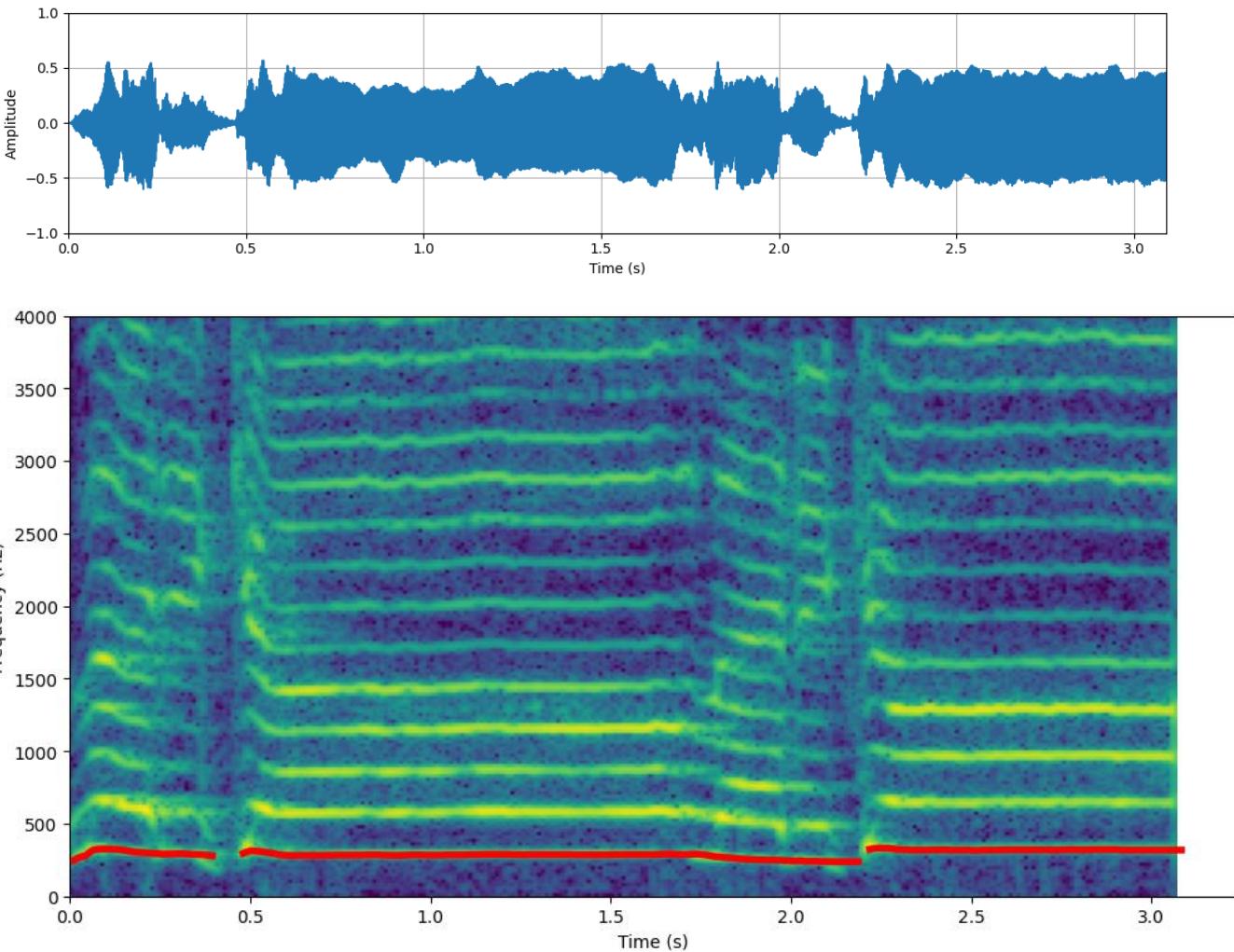


$$F_s = 44100$$

$$\frac{F_s}{f_{0\max}} \leq k \leq \frac{F_s}{f_{0\min}}$$

# Esercizio

- autotune-letitgo-bad.wav

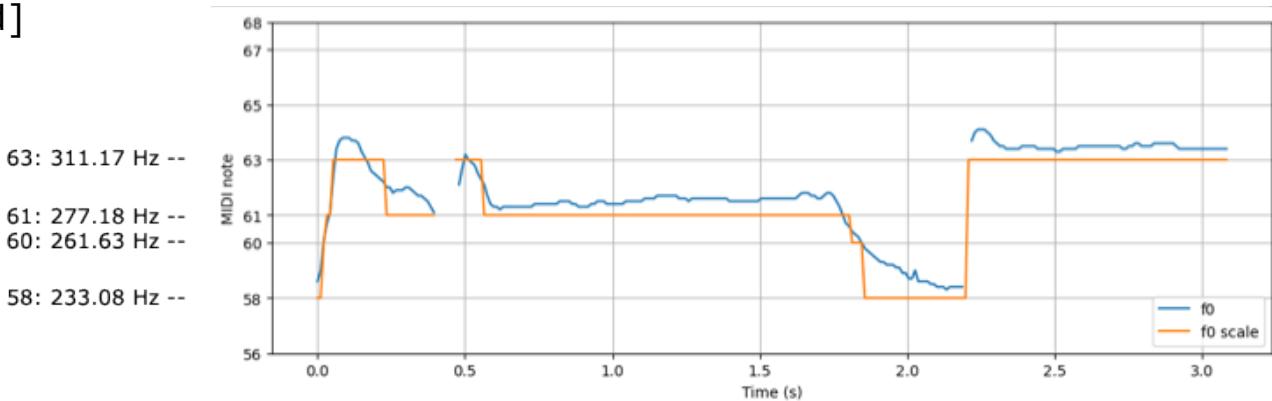


---

Mapping del pitch

# Mapping del pitch

```
def pitch_to_scale(pitch, scale):
    # To properly perform pitch rounding to the nearest degree from the scale, we need to repeat
    # the first degree raised by an octave. Otherwise, pitches slightly lower than the base degree
    # would be incorrectly assigned.
    degrees = librosa.key_to_degrees(scale)
    degrees = np.concatenate( (degrees, [degrees[0] + 12])) # add next octave key degree
    midi_note = librosa.hz_to_midi(pitch)
    # Subtract the multiplicities of 12 so that we have the real-valued pitch class of the input pitch.
    degree = midi_note % 12
    # Find the closest pitch class from the scale.
    degree_id = np.argmin(np.abs(degrees - degree))
    # Calculate the difference between the input pitch class and the desired pitch class.
    degree_difference = degree - degrees[degree_id]
    # Shift the input MIDI note number
    # by the calculated difference.
    midi_note -= degree_difference
    # print('midi_note', midi_note)
    # Convert to Hz.
    return librosa.midi_to_hz(midi_note)
```



---

Trasposizione del pitch

# Trasposizione del pitch

---

- L'obiettivo della *trasposizione del pitch* è quello di modificare l'altezza fondamentale f0 senza alterare la durata del segnale
- Esistono diversi algoritmi
  - ✓ Nel dominio del tempo: PSOLA e varianti
  - ✓ Nel dominio della frequenza: Phase Vocoder e varianti (\*)
  - ✓ Basati su modelli del segnale (analisi e ri-sintesi)
    - Sine plus noise modeling
    - LPC vocoder (eccitazione + filtro)
    - Reti neurali

*N.B. Diventano problematici se "c'è più di un pitch"*  
*(\*) altera le formanti, effetto chipmunk*

---

# Time-domain Pitch-Synchronous OLA

- Approccio di pitch shifting implementato nel dominio del tempo
- Si basa sull'identificazione di intervalli del segnale "sincroni" con il pitch (periodo  $T$ ) del segnale originale (cioè dei periodi)
- E sulla copia degli stessi a distanza  $T'$  nel segnale sintetizzato per creare l'effetto pitch trasposto  $1/T'$

- E. Moulines, F. Charpentier,  
"Pitch synchronous waveform processing  
techniques for text-to-speech synthesis  
using diphones",  
Speech Communications, 1990

Speech Communication 9 (1990) 453–467  
North-Holland

453

## PITCH-SYNCHRONOUS WAVEFORM PROCESSING TECHNIQUES FOR TEXT-TO-SPEECH SYNTHESIS USING DIPHONES

Eric MOULINES\* and Francis CHARPENTIER\*\*

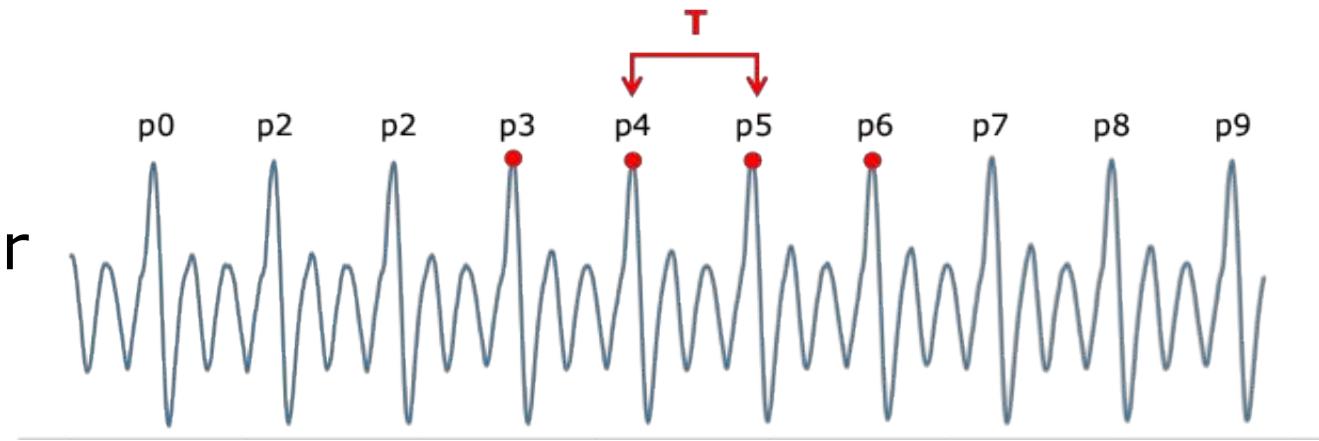
Centre National d'Etudes des Télécommunications, Département Signal, 46 rue Barrault, F-75643 Paris Cedex, France

Received 1 August 1990

**Abstract.** We review in a common framework several algorithms that have been proposed recently, in order to improve the voice quality of a text-to-speech synthesis based on acoustical units concatenation (Charpentier and Moulines, 1988; Moulines and Charpentier, 1988; Hamon et al., 1989). These algorithms rely on a pitch-synchronous overlap-add (PSOLA) approach for modifying the speech prosody and concatenating speech waveforms. The modifications of the speech signal are performed either in the frequency domain (FD-PSOLA), using the Fast Fourier Transform, or directly in the time domain (TD-PSOLA), depending on the length of the window used in the synthesis process. The frequency domain approach is capable of a great flexibility in modifying the spectral characteristics of the speech signal, while the time domain approach provides very efficient solutions for the real time implementation of synthesis systems. We also discuss the different kinds of distortions involved in these different algorithms.

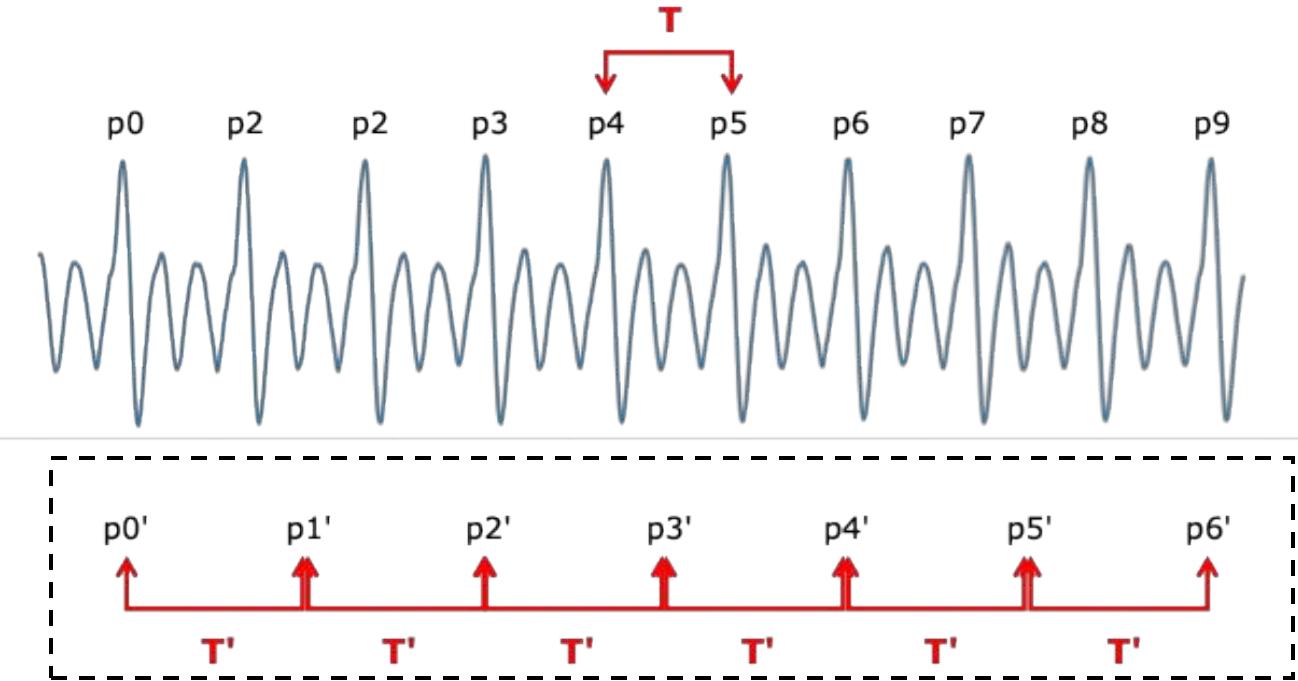
# TD-PSOLA

- Identificazione del pitch originale:  $1/T$
- Identificazione dei picchi per ogni periodo del segnale originale



# TD-PSOLA

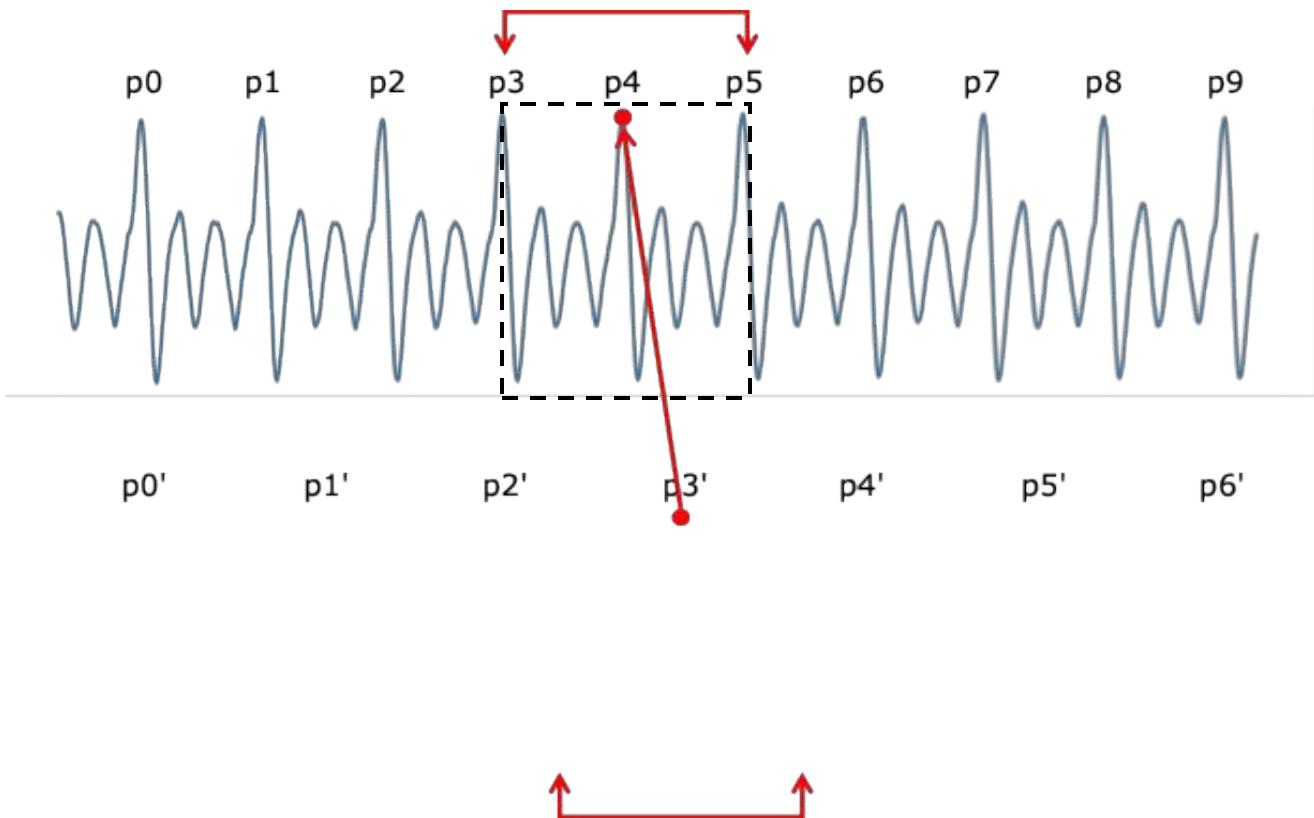
- Definizione del pitch target:  $1/T'$
- Identificazione dei nuovi picchi del segnale trasposto
  - ✓ Se  $T' > T$  saranno in numero minore, spaziati di  $T'$



# TD-PSOLA – overlap/add

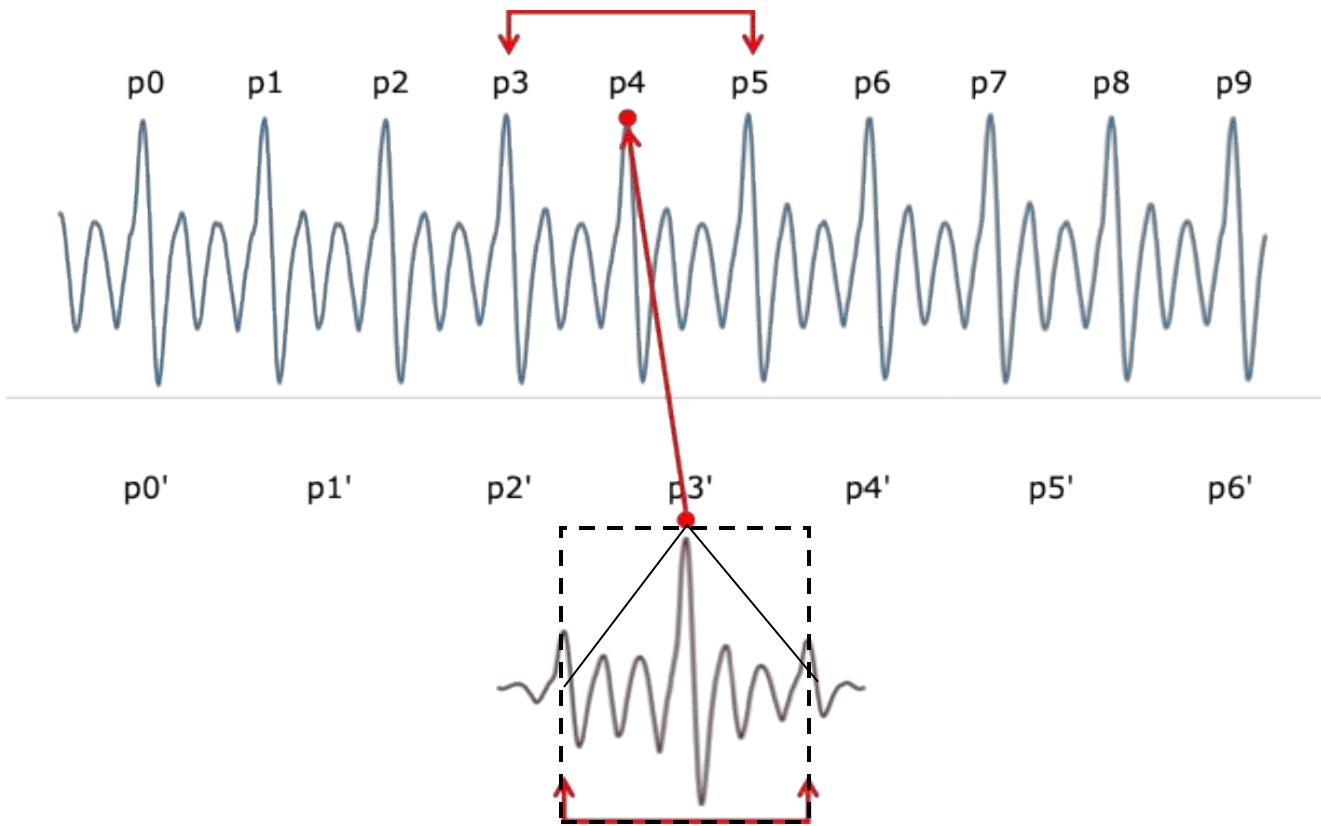
**Come viene creato  
il nuovo segmento  
centrato in  $p'[i]$ ?**

- Viene identificato il picco  $p[j]$  più vicino del segnale originale
- Da tale picco si estrae un segmento di due periodi



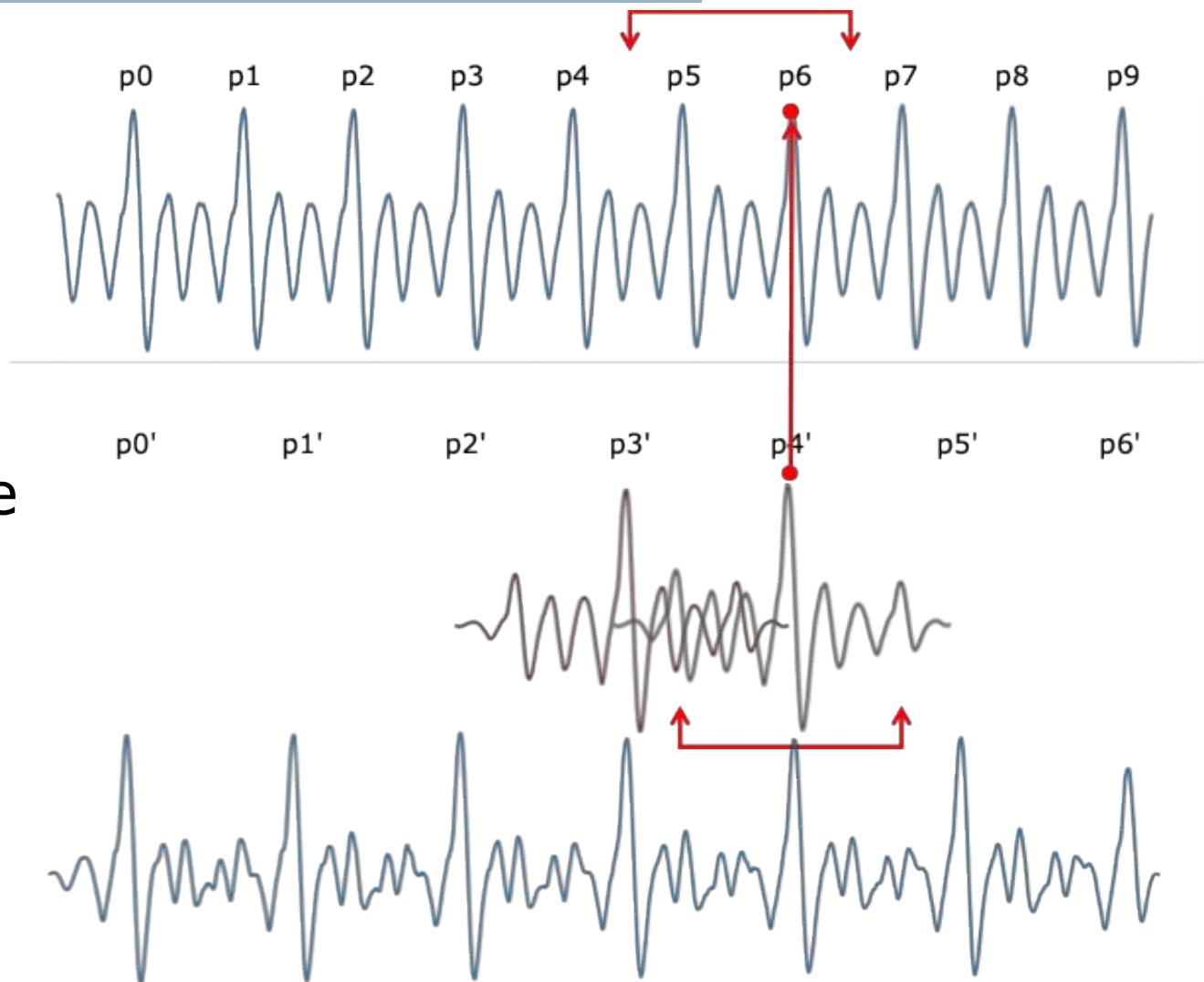
# TD-PSOLA – overlap/add

- Il segmento viene pesato con una finestra triangolare
- E sommato al segnale in uscita nella posizione centrata in  $p[i]'$



# TD-PSOLA – overlap/add

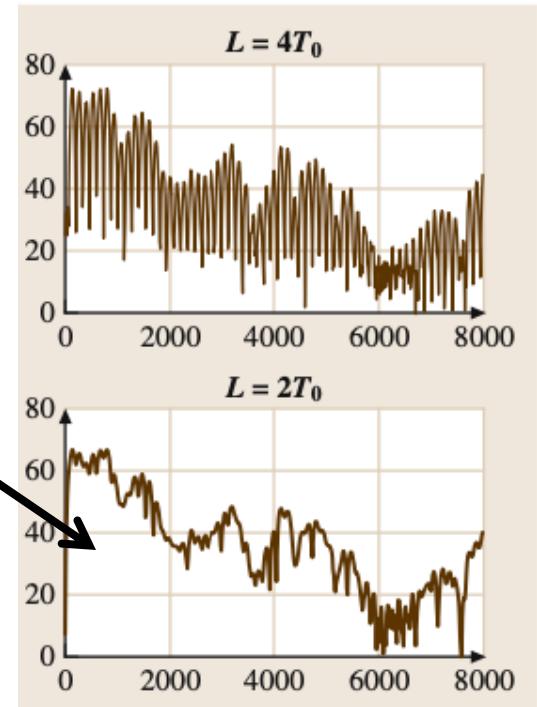
- Il tutto si ripete anche per il picco successivo
- Le operazioni successive di somma portano alla creazione del nuovo segnale con i picchi spaziati  $T'$



# TD-PSOLA - Limiti

*Tutt'altro che scontato*

- Serve la presenza di un pitch e capacità di identificarne correttamente i periodi
  - ✓ Voce umana o strumento singolo
  - ✓ Algoritmo del pitch robusto (~)
- Mantenimento del timbro
  - ✓ Utilizzo di segmenti  $L=2T_0$ , FFT circa inviluppo
  - ✓ Centratura sugli impulsi della glottide (~)
- Caveats
  - ✓ Ratio comprese tra 0.5 e 2 per la voce
  - ✓ Attenzione a porzioni non vocalizzate e transitori



Reference: Zolzer, "DAFX: Digital Audio Effects"

# Esercizio

## ■ Trasposizione del pitch

- ✓ Custom
- ✓ Autotune
- ✓ Re-tune

