

# Sintesi 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

---

- Pure Data
  - ✓ Setup, linguaggio, esempi
- Oscillatore digitale
- > *MIDI (cenni)*
- Sintesi additiva
- > *Trasposizione vs Traslazione in altezza*

# Riferimenti

---

- [VALLE] Lombardo, Valle, "Audio e Multimedia – 5<sup>ed</sup>", 2024
  - ✓ Sez. 5 La sintesi del suono
    - Sez 5.1 L'oscillatore digitale, 5.2 Segnali di controllo,
    - Sez 5.3 Metodi di sintesi (Campionamento, Generazione diretta, Trasformazione, Modulazione)
  - ✓ Sez. 6 Musica e MIDI
    - Sez. 6.4 MIDI
- Altro
  - ✓ [DODGE] Dodge, "Computer Music: Synthesis, Composition, and Performance", 1997
  - ✓ [ROADS] Roads, "The Computer Music Tutorial - 2<sup>ed</sup>", 2023

---

# Introduzione

# Introduzione

---

- Definizione sintesi audio (digitale)
  - ✓ Processo con cui un segnale sonoro [sequenza numerica] viene generato artificialmente tramite algoritmi e calcoli numerici eseguiti da un computer o da un processore digitale, senza la necessità di una sorgente acustica reale.

Reference:

# Tipi di sintesi

---

- Sintesi per addizione (additiva)
  - ✓ Somma di segnali semplici (onde sinusoidali e/o rumore)
- Sintesi per sottrazione (sottrattiva)
  - ✓ Sottrazione di componenti da segnali ricchi (dente di sega, onda quadra, rumore) tramite filtri
- Sintesi per modulazione
  - ✓ Modulazione di frequenza, ampiezza o fase di un segnale
- Sintesi per campionamento
  - ✓ Combinazione di suoni campionati / registrati
- Sintesi per modelli fisici
  - ✓ Simulazione del comportamento fisico dell'oggetti

Reference:

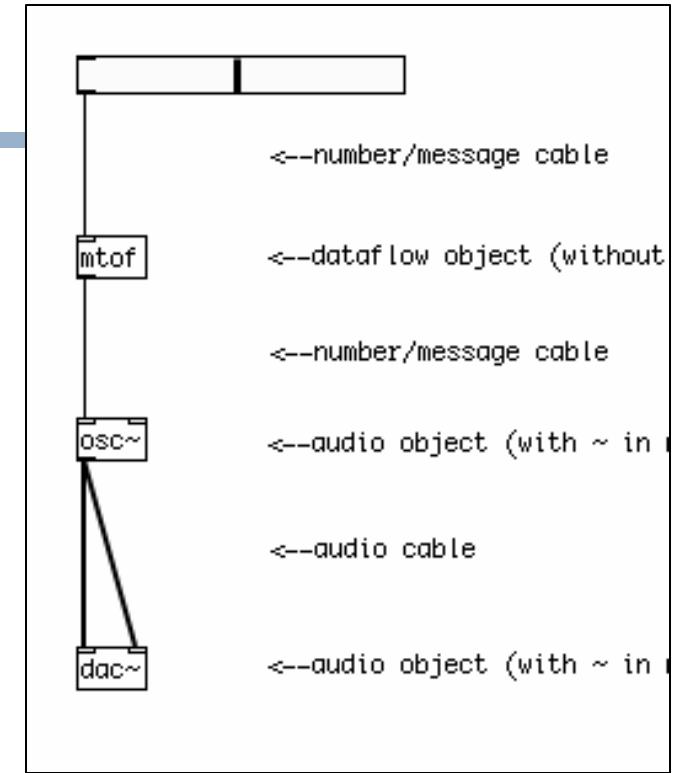
---

---

Pure Data

# Pure Data (Pd)

- Ambiente di programmazione visuale per la sintesi e l'elaborazione di suoni in tempo reale, in modo interattivo.
- Sviluppato da Miller Puckette negli anni '90
  - ✓ Progetto open source
  - ✓ Molto simile al programma Max sviluppato sempre da Puckette presso l'IRCAM e poi commercializzato come Max/MSP
  - ✓ I programmi sviluppati in Pure Data prendono il nome di "patch"
- E' utilizzato per musica elettronica, sound design, installazioni interattive e ricerca nell'ambito dell'audio digitale



Salvati come  
file di testo .pd

# Pure Data - setup

---

- Sito Pure Data "vanilla": <https://puredata.info/>
  - ✓ Download e un po' di documentazione
- Analizzatore di spettro in tempo reale
  - ✓ Friture - <https://friture.org/>
- Canali audio e MIDI virtuali
  - ✓ Virtual Audio Cable (lite) - <https://vac.muzychenko.net/en/download.htm>
  - ✓ LoopbeAudio - <https://nerds.de/en/loopbeaudio.html>
  - ✓ LoopBe1 - <https://nerds.de/en/loopbe1.html>
- Interfaccia MIDI
  - ✓ Virtual MIDI Piano Keyboard - <https://vmpk.sourceforge.io/>

# Pure Data - documentazione

---

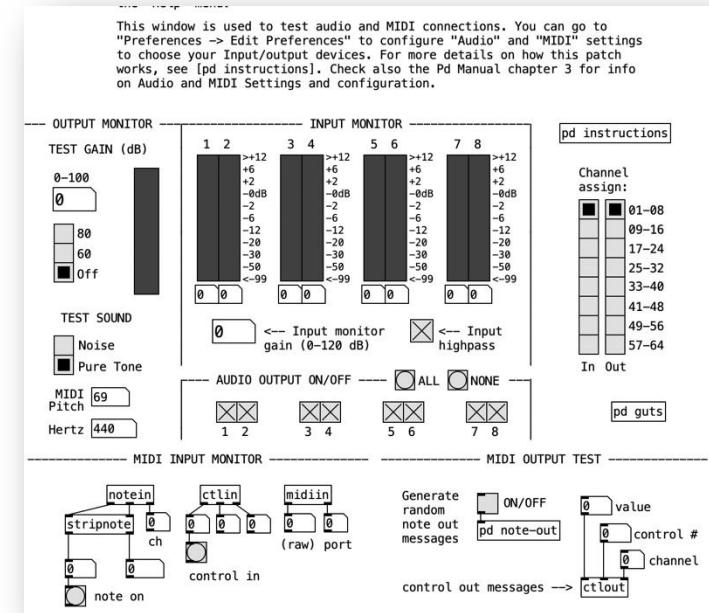
- Manuale
  - ✓ [https://msp.ucsd.edu/Pd\\_documentation/index.htm](https://msp.ucsd.edu/Pd_documentation/index.htm)
- Programming Electronic Music in Pd
  - ✓ <http://pd-tutorial.com/english/index.html>
- Pure Data "Course"
  - ✓ <https://www.youtube.com/playlist?list=PL12DC9A161D8DC5DC>
- Approfondimenti
  - ✓ Bianchi, Cipriani, Giri, "Pure Data: Musica Elettronica e Sound Design"
  - ✓ Farnell, "Designing Sound", 2010

# Pure Data - setup

- Sito Pure Data "vanilla": <https://puredata.info/>
  - ✓ Download e un po' di documentazione

## ■ Configurazione

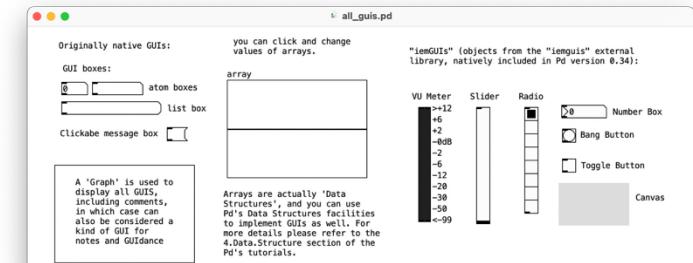
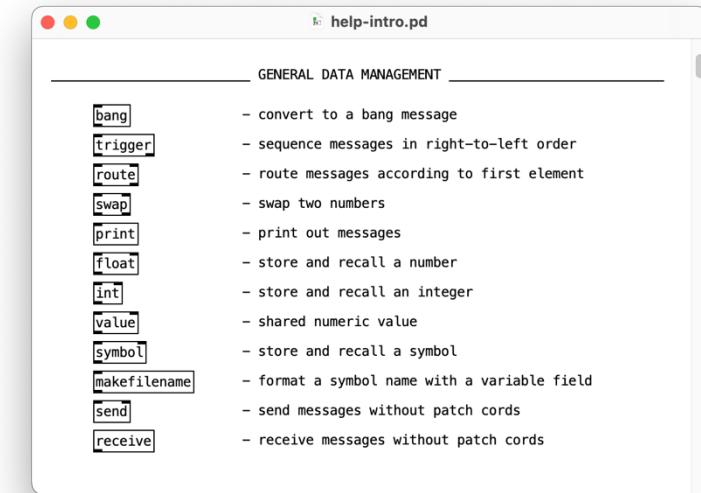
- ✓ Audio ON / OFF
- ✓ Media -> Audio settings:
  - input & output devices
- ✓ Media -> MIDI settings
  - -> input: loopbe (che riceve da interfaccia midi)
  - -> output: -> wavetable synth (per la sintesi Windows)
- ✓ Media -> Test Audio and MIDI ...



# Linguaggio di Pure Data

- Il linguaggio di Pure Data è basato su un sistema ad oggetti connessi graficamente che rappresentano
  - ✓ Il flusso del segnale (linee spesse)
  - ✓ Il flusso dei comandi di controllo (linee sottili)
- Gli oggetti con il nome seguito da "tilde" (`osc~`) processano il segnale audio digitale
- Creare un oggetto
  - ✓ Put -> object (o `ctrl-1`)
  - ✓ Scrivere il nome
  - ✓ Utile visualizzarne l'help con il tasto destro

*L'help con il tasto destro in una zona vuota della patch visualizza la lista degli oggetti disponibili.*



# Alcuni oggetti "base"

## ■ Messaggio

- ✓ Contiene valori o comandi da inviare ad altri oggetti quando viene "attivato" (e.g. cliccato)

## ■ Numero (Number)

- ✓ Mostra e memorizza un valore numerico
- ✓ Invia o riceve un valore quando viene modificato
- ✓ Si può modificare con il mouse (anche con shift)

## ■ Operatori (\*, /, -, +, mod, pow, log, sin, cos)

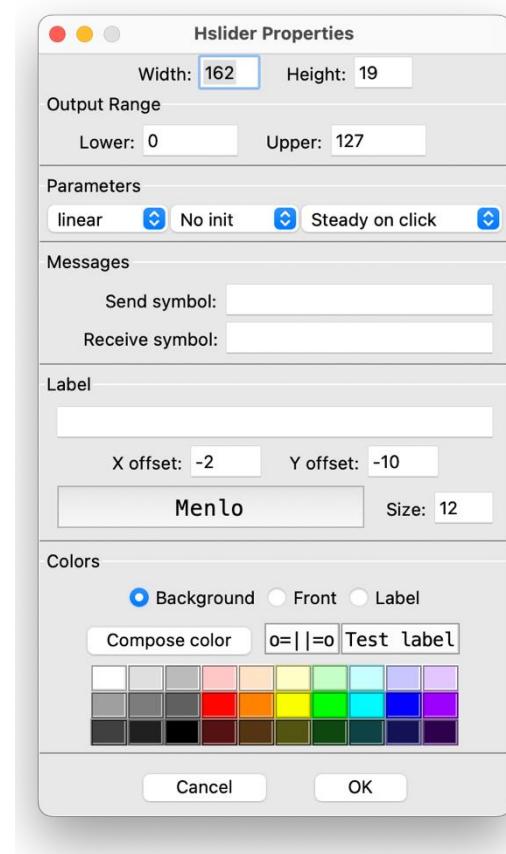
- ✓ Sono oggetti "come gli altri" ed eseguono calcoli

Put	Find	Media
Object	⌘ 1	
Message	⌘ 2	
Number	⌘ 3	
List	⌘ 4	
Symbol		
Comment	⌘ 5	
Bang	⌘ B	
Toggle	⌘ T	
Number2	⌘ N	
Vslider	⌘ V	
Hslider	⌘ J	
Vradio	⌘ D	
Hradio	⌘ I	
VU Meter	⌘ U	
Canvas	⌘ C	
Graph	⌘ G	
Array	⌘ A	

# Alcuni oggetti "base"

## ■ Slider

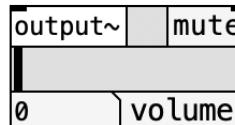
- ✓ Due oggetti slider VSlider e Hslider
- ✓ Come altri oggetti ha delle proprietà che possono essere personalizzate
  - Minimo: Lower
  - Massimo: Upper
  - Progressione: lineare o logaritmica



## ■ [adc~] e [dac~]



- ✓ Oggetti input e output
  - [output~] oggetto più evoluto con slider logaritmico



# Primi passi (Start Here)

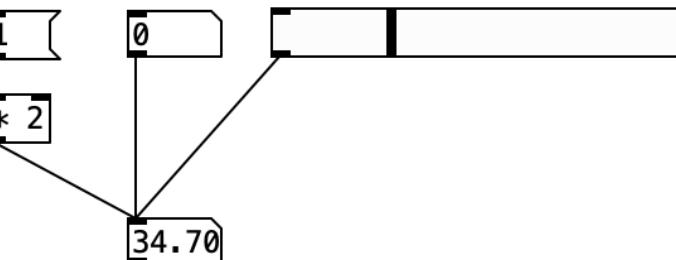
## ■ Edit Mode (on / off)

- ✓ In edit mode si "programma" / edita la patch
- ✓ Quando non si è in edit mode si usa la patch

## ■ Gli oggetti ricevono parametri

- ✓ Dinamici: tramite gli inlet (ingressi)
- ✓ Inizializzazione: come parametri testuali

## ■ Esempio: realizzare questa patch



Reference: <https://puredata.info/docs/StartHere>

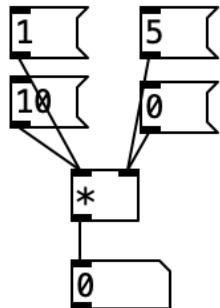
# Attivazione

---

## ■ Principio di attivazione

- ✓ Gli oggetti non calcolano i nuovi valori continuamente
- ✓ Eseguono un'operazione solo quando vengono attivati
- ✓ L'attivazione avviene quando ricevono un messaggio sull'inlet (ingresso) HOT, quello più a sinistra

## ■ Esempio



# Oscillatore digitale

## ■ Direct digital synthesizer (DDS)

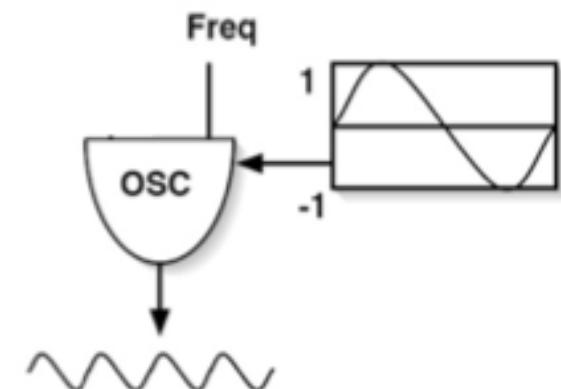
✓ Sistema discreto nel tempo ( $f_s$ ) che genera un segnale periodico (sinusoide, onda triangolare, quadra, dente di sega) calcolando numericamente (numeri reali) il valore dei campioni

✓ Controllato in frequenza  $f_0$  (*e in forma d'onda da generare*)

- Ad ogni clock  $\frac{1}{f_s}$  la fase incrementa di  $2\pi \frac{f_0}{f_s}$  e calcola il nuovo valore

$$\theta[n] = \theta[n - 1] + 2\pi \frac{f_0}{f_s} \quad , \quad x[n] = \sin(\theta[n])$$

- $f_0$ : frequenza del segnale
- $f_s$ : frequenza di campionamento
- $\theta[n]$ : fase accumulata (mod  $2\pi$ )



# Oscillatore digitale

## ■ Controllo in ampiezza

*Nei tool che utilizzeremo*

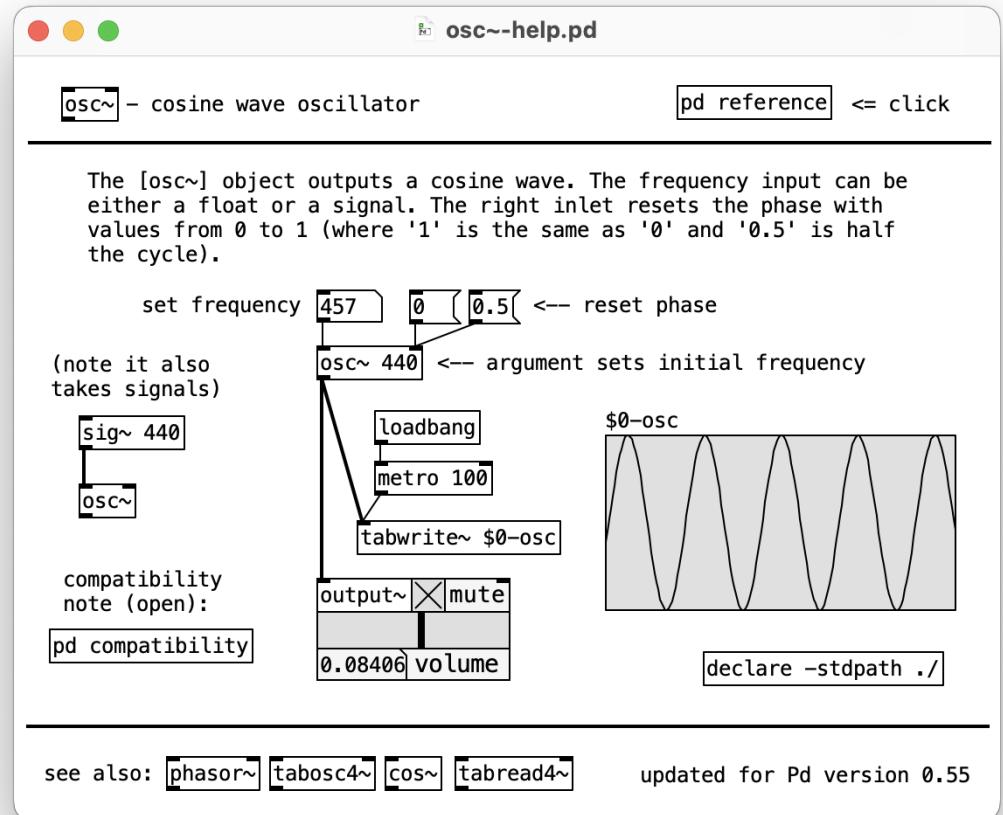
- ✓ Generalmente il controllo in ampiezza (volume) è applicato sul segnale in uscita dall'oscillatore



- ✓ Nelle slides, per brevità, si utilizzerà invece il controllo in ampiezza direttamente in ingresso all'oscillatore

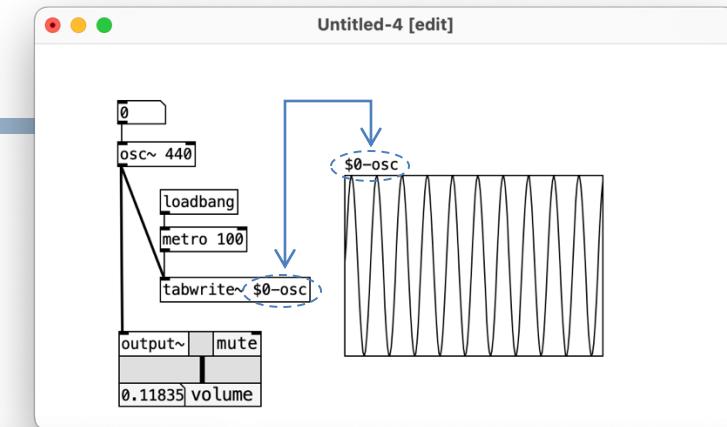
# Pure data – oscillatore digitale

- In Pure Data abbiamo (almeno) due oscillatori
  - ✓ [osc~] tono puro
  - ✓ [phasor~] dente di sega
- Esempio: help [osc~]

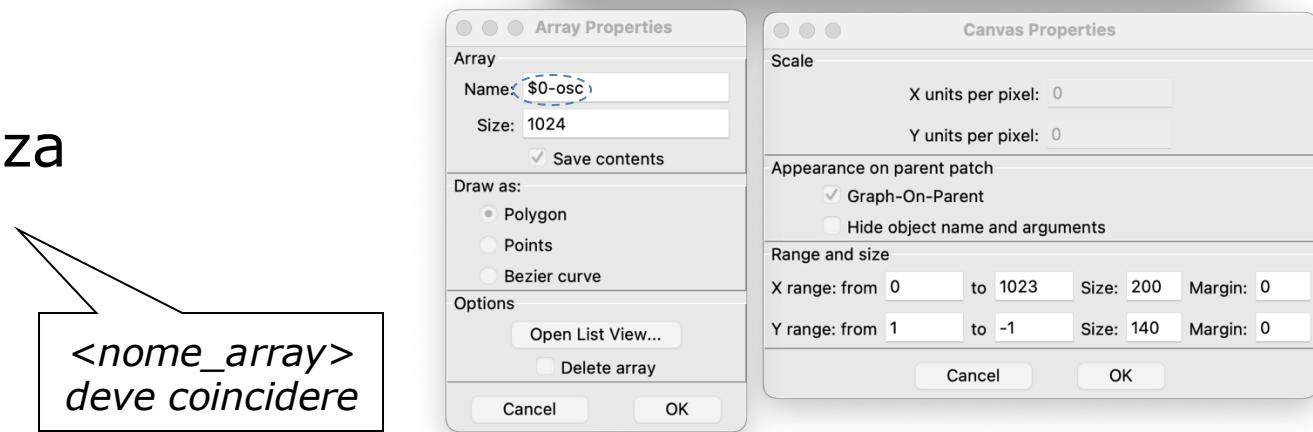


# Pure data - array

- Per osservare una forma d'onda possiamo scrivere i campioni in un array



- Oggetto array (Put > Array)
  - ✓ <nome\_array> e la lunghezza
- [tabwrite~ <nome\_array>]
  - ✓ Scrive i campioni nell'array di nome <nome\_array>



- E' utile sottocampionare i valori di una forma d'onda attivando [tabwrite~] ad intervalli regolari con un oggetto [metro <ms>]

---

MIDI (cenni)

# MIDI

---

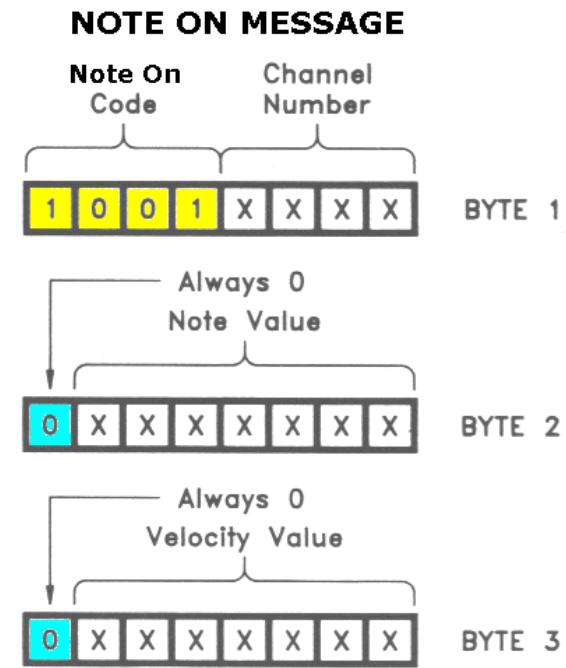
- Musical Instrument Digital Interface
  - ✓ Standard 1983 (Roland, Yamaha, Korg, etc.)
  - ✓ Comunicazione tra dispositivi di produttori differenti
  - ✓ Istruzioni per il sintetizzatore (es. provenienti da un sequencer o uno "strumento") che poi si occupa della sintesi dei suoni effettivi
- Rappresentazione dei dati musicali
- Sincronizzazione tra i dispositivi

Sequencer: sistema di registrazione e di esecuzione dotato di una memoria programmabile nella quale vengono memorizzati i dati di controllo operativi necessari alla (ri-)generazione di eventi musicali (memorizza i messaggi MIDI)

---

# MIDI "note"

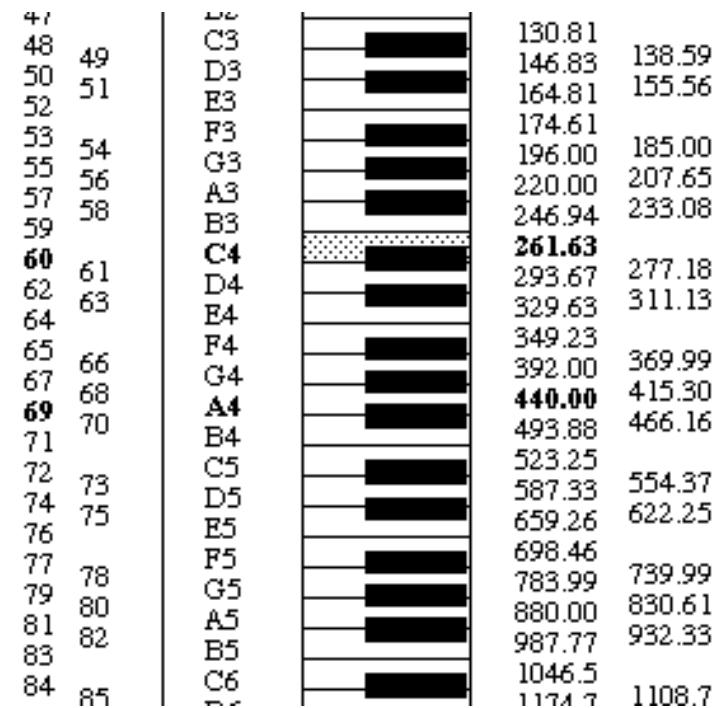
- Il protocollo MIDI definisce due messaggi "note" che rappresentano la nota da suonare
  - ✓ Note On: che attiva la nota
  - ✓ Note Off: che la disattiva (spegne)
    - Spesso si usa al suo posto un messaggio "note on" con velocity = 0
- I messaggi sono caratterizzati dai valori:
  - ✓ Number: da 0 a 127 (che corrisponde pitch / altezza)
  - ✓ Velocity: da 0 a 127 (che indica la forza con cui è suonata)
- In aggiunta il MIDI ha il concetto di \_canale\_ dove ogni canale rappresenta un flusso separato di messaggi



Reference: [FMP] [https://www.audiolabs-erlangen.de/resources/MIR/FMP/C1/C1S2\\_MIDI.html](https://www.audiolabs-erlangen.de/resources/MIR/FMP/C1/C1S2_MIDI.html)

# Scala MIDI

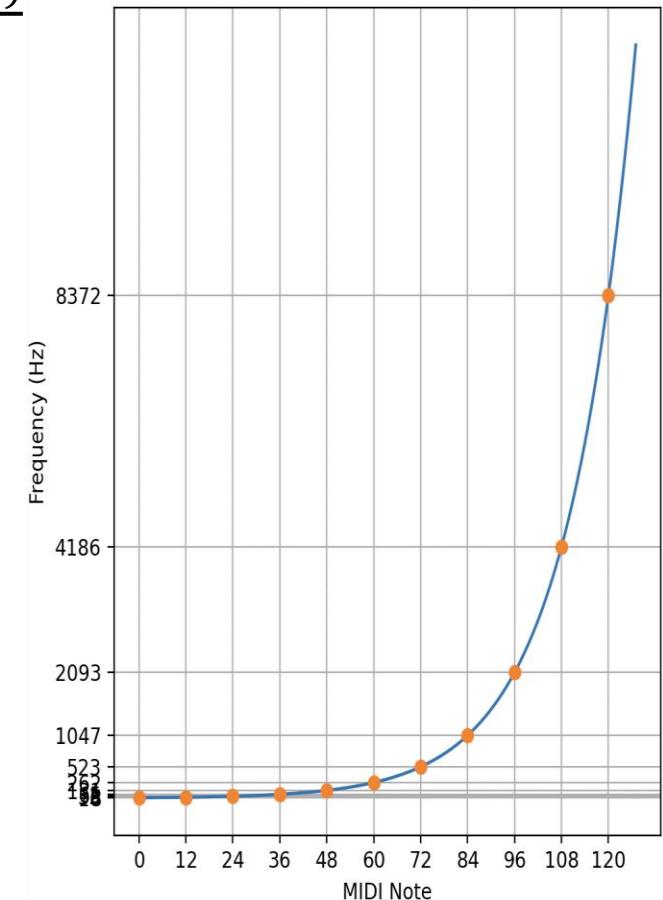
- Il valore della nota MIDI può essere immaginato come corrispondente alla tastiera di un pianoforte a 128 tasti
  - ✓ Il valore MIDI 69 corrisponde a A4
    - A = LA, 4 = IV ottava => 440 Hz
  - ✓ Due valori successivi distano 1 semitono
  - ✓ Ogni ottava è composta da 12 semitonni
    - C, C#, D, D#, E, F, F#, G, G#, A, A#, B, C
- L'incremento di una ottava corrisponde al raddoppio della frequenza
  - ✓ Suoni che distano un'ottava hanno lo stesso nome musicale (stessa nota)
  - ✓ L'orecchio li percepisce "uguali", ma più acuti o gravi



# Scala MIDI

- La corrispondenza tra nota midi (n) e frequenza (f) in Hz è descritta dalla formula
- Si crea quindi una scala lineare dove +12 (semitoni) corrisponde ad un raddoppio della frequenza
- L'oggetto [mtof] converte la nota MIDI in valore di frequenza (secondo la formula)

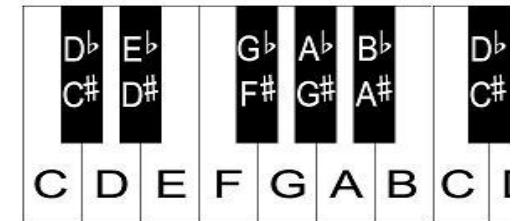
$$f = 440 \cdot 2^{\frac{(n-69)}{12}}$$



# Esercizi con la scala MIDI

$$f = 440 \cdot 2^{\frac{(n-69)}{12}}$$

*Definita la nota A4, midi=69, f=440 Hz  
e la distanza di 9 semitonni da C4 ad A4*



- Usando la scala MIDI, calcolare
  - ✓ Le frequenze delle note  $C_n$  e di conseguenza la larghezza delle ottave (western music)
  - ✓ Data una frequenza  $f_0$  (es. 1100 Hz)
    - a che ottava appartiene
    - quale è la larghezza di quell'ottava
    - quanto vale la distanza di semitono "a cavallo" di quella frequenza (cioè tra la chroma prima e la chroma dopo)

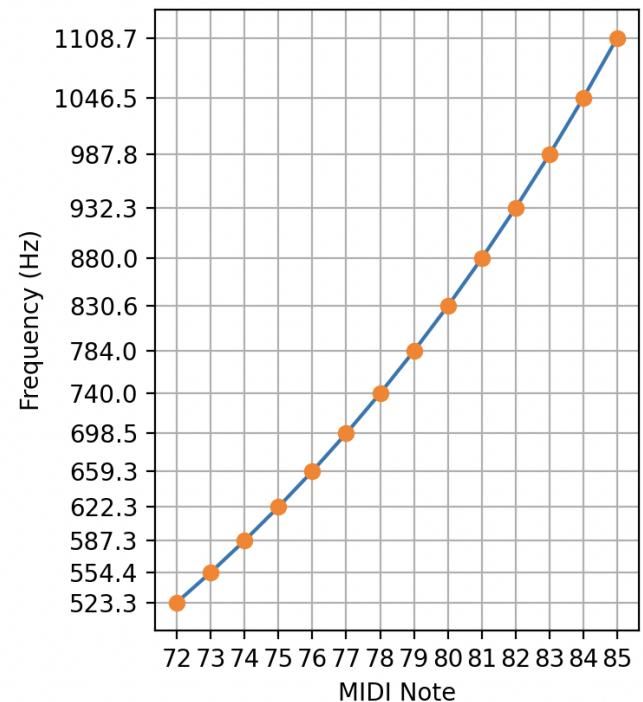
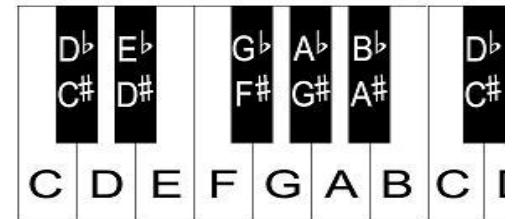
# Esercizi con la scala MIDI

$$f = 440 \cdot 2^{\frac{(n-69)}{12}}$$

Definita la nota A4, midi=69, f=440 Hz  
e la distanza di 9 semitonni da C4 ad A4

- Usando la scala MIDI, calcolare
  - ✓ Le frequenze delle note C<sub>n</sub>  
e di conseguenza la larghezza delle ottave  
(western music)
  - ✓ La scala MIDI è lineare  $m = 12*n + b$   
sappiamo che  $60 = 12*4 + b$   
quindi  $b=12 \Rightarrow m = 12*n + 12 = 12*(n+1)$
  - ✓ Le frequenze C<sub>n</sub> saranno

$$C_{nf} = 440 \cdot 2^{\left(\frac{(12(n+1)-69)}{12}\right)} = 440 \cdot 2^{\left(n-\frac{57}{12}\right)}$$

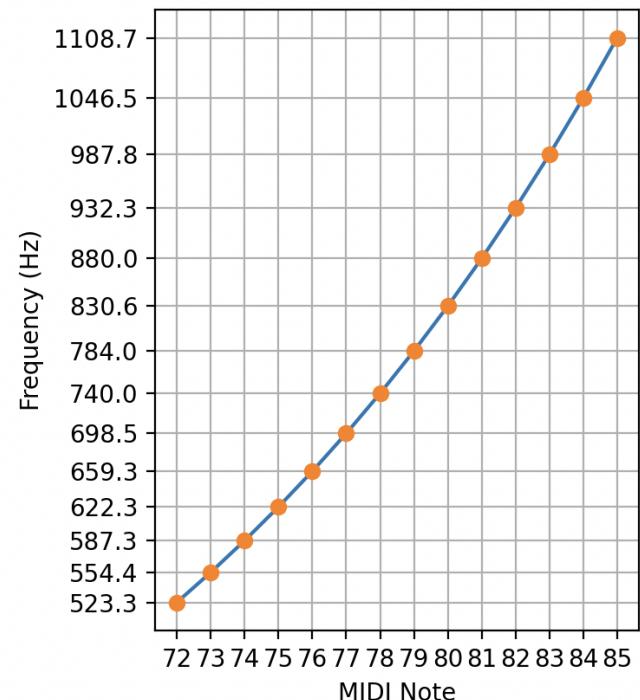
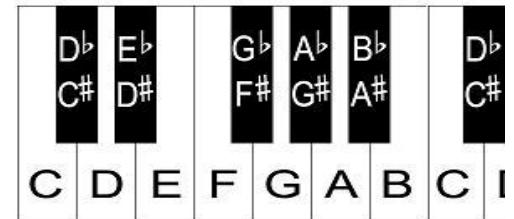


# Esercizi con la scala MIDI

$$f = 440 \cdot 2^{\frac{(n-69)}{12}}$$

Definita la nota A4, midi=69, f=440 Hz  
e la distanza di 9 semitonni da C4 ad A4

- Usando la scala MIDI, calcolare
  - ✓ Data una frequenza f0 (es. 1100 Hz)
    - a che ottava appartiene
    - quale è la larghezza di quell'ottava
  - ✓  $m = \log_2(f/440) * 12 + 69 \approx 84-85$
  - ✓  $m = 12(n+1) \Rightarrow n = (m-12)/12 = m/12 - 1 \approx 6$
  - ✓  $C_n f = 440 \cdot 2^{\left(\frac{n-57}{12}\right)} = 1046.5 \text{ Hz}$

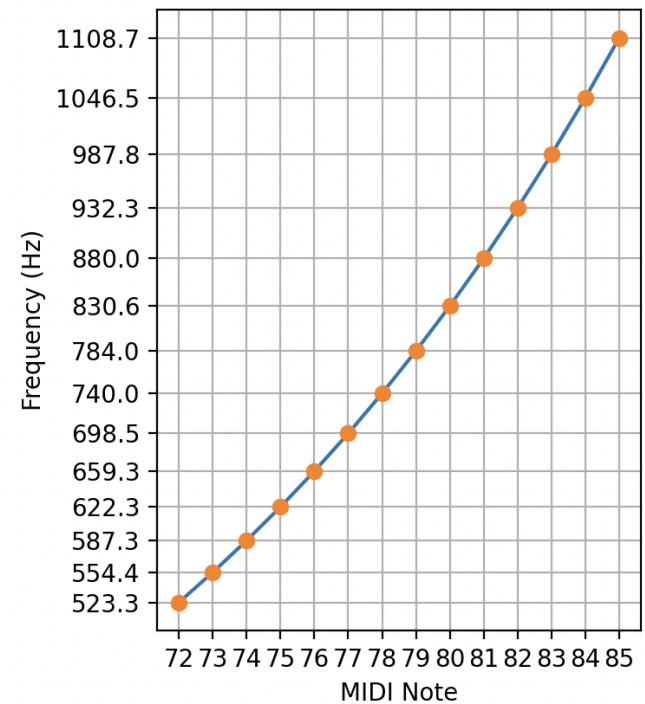
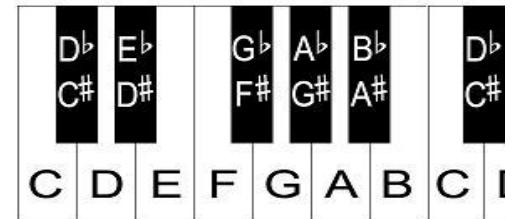


# Esercizi con la scala MIDI

$$f = 440 \cdot 2^{\frac{(n-69)}{12}}$$

Definita la nota A4, midi=69, f=440 Hz  
e la distanza di 9 semitonni da C4 ad A4

- Usando la scala MIDI, calcolare
  - ✓ Data una frequenza  $f_0$  (es. 1100 Hz)
    - quanto vale la distanza di semitono "a cavallo" di quella frequenza (cioè tra la chroma prima e la chroma dopo)
  - ✓  $m = 84-85$
  - ✓  $f^+ = c \cdot f^-$ ,  $f^+ - f^- = c \cdot f^- - f^- = (c-1)f^-$
  - ✓  $f_{84} = 440 \cdot 2^{((m-69)/12)} = 1046.5$  Hz  
semitono  $\Rightarrow [2^{(1/12)} - 1] * 1046.5 = 62.22$  Hz



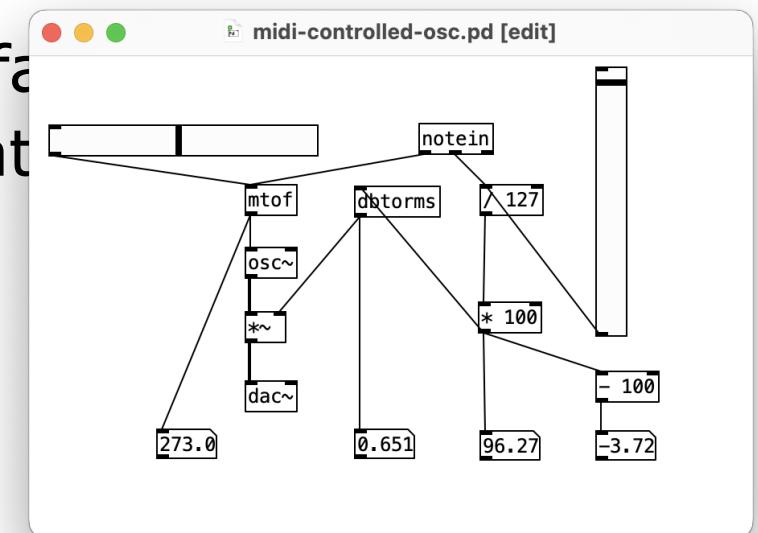
# Esercizio

---

- Realizzare una patch per controllare un oscillatore digitale tramite due slider in
  - ✓ frequenza (logaritmica) convertita tramite [motf]
  - ✓ ampiezza (logaritmica) convertita tramite [dbtorms]
- Il controllo in ampiezza va applicato come fattore moltiplicativo sul segnale in uscita dall'[osc~] usando l'operatore [\*~]
- L'oggetto [dbtorms] assume che
  - ✓ 100 corrisponda a 0 dBFS (RMS=1)
  - ✓ 0 corrisponda a -inf dBFS (RMS=0)

# Esempio - soluzione

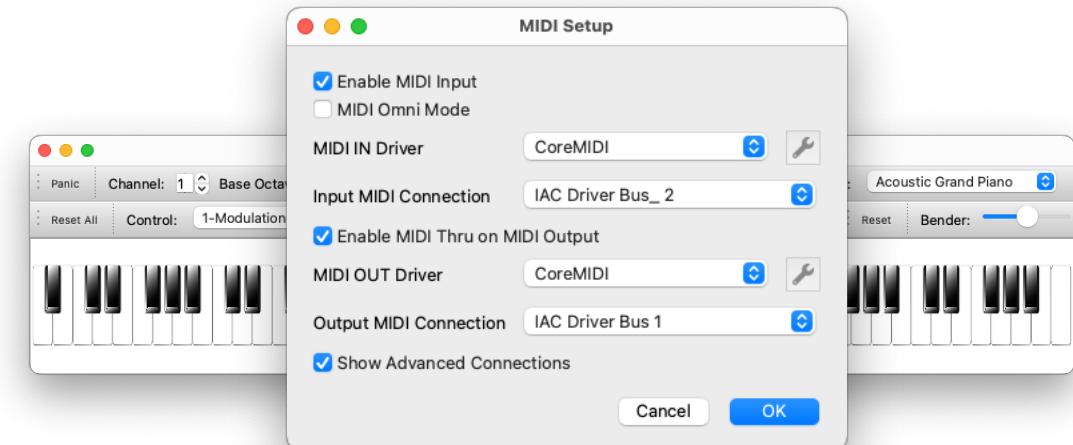
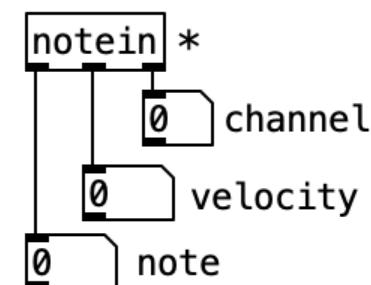
- Realizzare una patch per controllare un oscillatore digitale tramite due slider in
  - ✓ frequenza (logaritmica) convertita tramite [motf]
  - ✓ ampiezza (logaritmica) convertita tramite [dbtorms]
- Il controllo in ampiezza va applicato come facoltà di segnale in uscita dall'[osc~] usando l'operatore [\*~]
- L'oggetto [dbtorms] assume che
  - ✓ 100 corrisponda a 0 dBFS (RMS=1)
  - ✓ 0 corrisponda a -inf dBFS (RMS=0)



midi-controlled-osc.pd

# Pure data – notein

- L'oggetto [notein] provvede a "catturare" i messaggi MIDI
- L'oggetto ha tre outlet / uscite
  - ✓ Note, Velocity, Channel
- Il messaggio "note off" può essere individuato da velocity = 0
- I messaggi MIDI vanno inviati da una interfaccia MIDI (VMPK, Edit > MIDI connection) su un canale MIDI (LoopBe1)



# Pure Data – altri oggetti

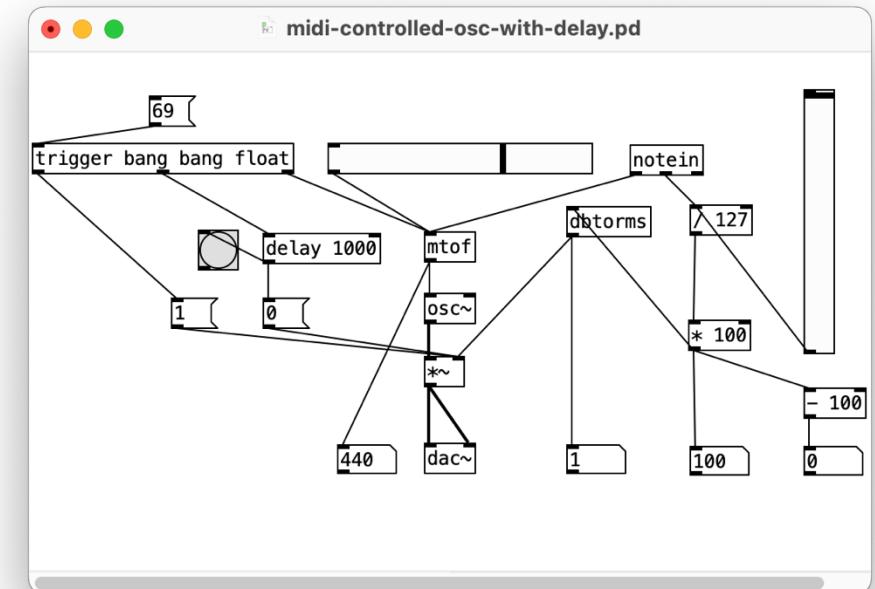
## ■ [delay <ms>]

- ✓ Invia un "bang" dopo un certo ritardo (dopo la sua attivazione) specificato come argomento o inlet destro.
- ✓ Se riceve un messaggio con un valore nell'inlet sinistro emette un bang dopo quel tempo in ms, nell'inlet destro aggiorna solamente il tempo di ritardo

## ■ [trigger <> <> <>]

- ✓ Emette in uscita (copia) il suo input da destra a sinistra
- ✓ Gli argomenti specificano il numero di outlet e il tipo di dato emesso

[ ATTACCO / RILASCIO ]



# Standard MIDI File (SMF)

---

- Formato per memorizzare le sequenze MIDI
  - ✓ Definito nel 1988
  - ✓ Aggiunto il concetto di "tempo" assente nel protocollo MIDI (dove i device generano i messaggi quando vengono "suonati")
    - Sequenza di messaggi MIDI intercalati da informazioni sul ritardo di emissione
- Struttura
  - ✓ Header chunk: # tracce, # tick per nota da  $\frac{1}{4}$
  - ✓ Tracce: tempo delta + evento
    - Tempo delta: distanza tra eventi in PPQN

Durata in millisecondi di un tick:  $(60\ 000 / \text{bpm}) / \text{PPQN}$

---

# Standard MIDI File (SMF)

## ■ Formato per memorizzare le sequenze MIDI

- ✓ Definito nel 1988
- ✓ Aggiunto il concetto di "tempo" assoluto: i device generano i messaggi quando sono in esecuzione
  - Sequenza di messaggi MIDI interamente memorizzata all'istanza di emissione

## ■ Struttura

- ✓ Header chunk: # tracce, # tick per barra
- ✓ Tracce: tempo delta + evento
  - Tempo delta: distanza tra eventi in frame

MIDI Event List					
MIDI 1	21 Events	Length/Info	Event	Start	
8  1  67	↓ A2	111  64	0  2  309	1  3  000	⑧
1  3  000	↓ G3	64  64	0  1  000	1  3  000	⑨
6  2  510	△ 1	83  64	0  1  000	6  2  630	mod wheel
6  2  630	△ 1	81  64	mod wheel	6  3  660	mod wheel
6  3  660	△ 1	79  64	mod wheel	6  3  000	0  1  000
6  3  000	↓ G3	94  64	0  1  000	6  3  000	0  1  000
6  3  000	↓ D3	50  64	0  1  000	6  3  030	mod wheel
6  3  030	△ 1	69  64	mod wheel	6  3  550	mod wheel
6  3  550	→ 10	56  56	pan	6  3  600	mod wheel
6  3  600	△ 1	48  48	mod wheel	6  3  684	volume
6  3  684	↓ 7	9  9	volume	6  3  690	mod wheel
6  3  690	△ 1	44  44	mod wheel	6  3  720	mod wheel
6  3  720	△ 1	41  41	mod wheel	6  3  840	mod wheel
6  3  840	△ 1	39  39	mod wheel	6  4  000	0  0  281
6  4  000	↓ F#3	94  64	0  0  281	6  4  000	0  1  000
6  4  000	↓ Bb2	127  64	0  1  000	7  1  000	0  0  000

Durata in millisecondi di un tick:  $(60\ 000 / \text{bpm}) / \text{PPQN}$

# Sintetizzatore

---

- I messaggi MIDI non contengono audio
- Il sintetizzatore interpreta i messaggi MIDI e li trasforma in audio attraverso un meccanismo di sintesi
- I sintetizzatori possono essere sia HW sia SW (programmi o plugin)
- Sui sistemi operativi abbiamo (di default) basati su banchi di suoni campionati
  - ✓ Windows: Microsoft GS Wavetable Synth
    - Roland sound Canvas SC-55 del 1991
  - ✓ MacOS: CoreAudio (Apple DLSMusicDevice)
    - Deriva da QuickTime Musical Instruments del '90, aggiornato in macOS 10.4 (2005) e 10.7 (2011)

---

## Sintesi addittiva

# Sintesi additiva

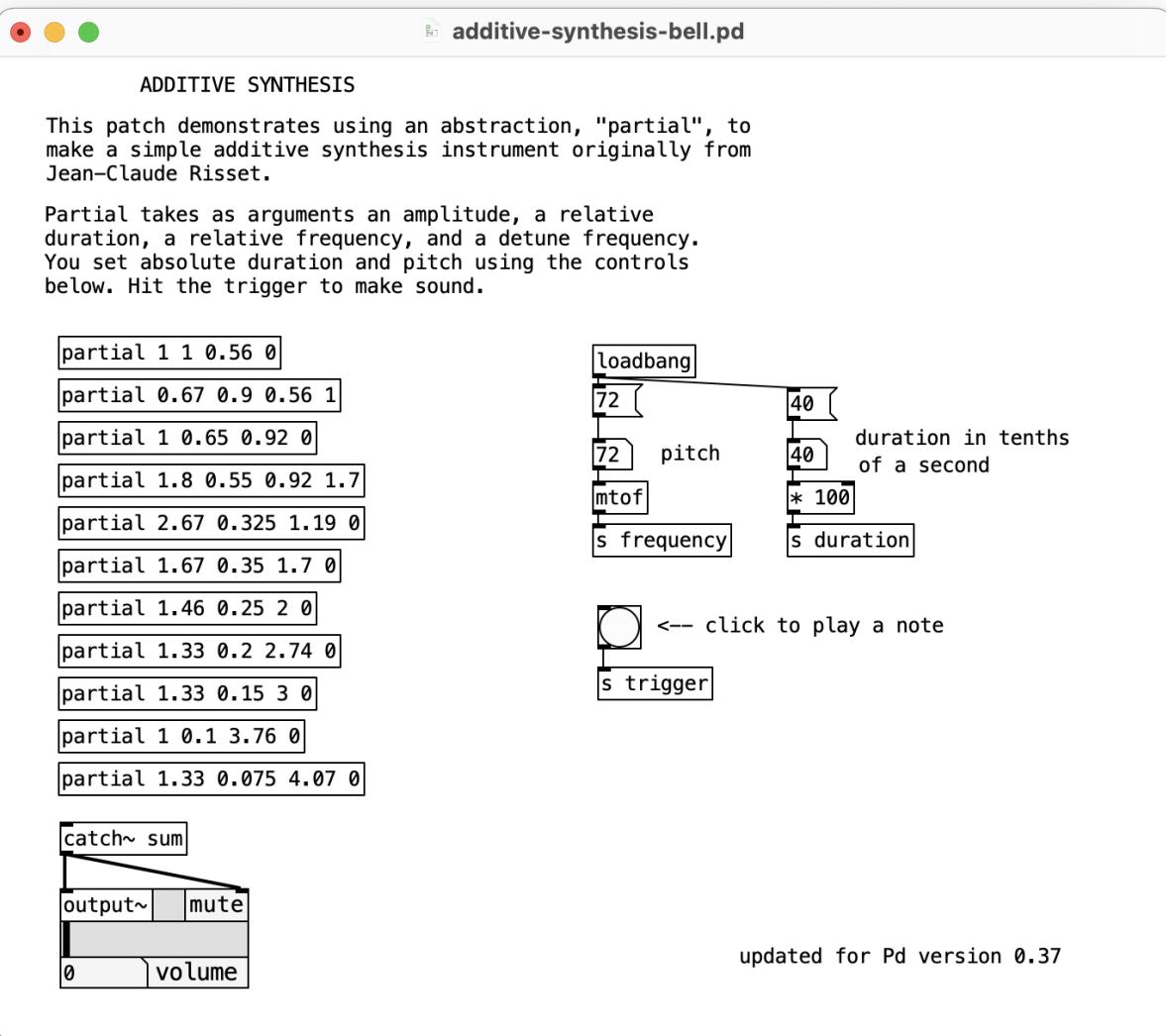
---

- Crea spettri complessi aggiungendo componenti in frequenza a partire da un tono semplice
  - ✓ Toni a differente ampiezza, frequenza e fase
  - ✓ L'ampiezza delle componenti può variare nel tempo (vedi ADSR)
- Vantaggi
  - ✓ I parametri delle componenti possono essere derivati dall'analisi spettrale del segnale
  - ✓ Flessibile: praticamente ogni suono può essere sintetizzato
- Svantaggi
  - ✓ Complessità algoritmica: servono molte (decine di) componenti per rendere sufficientemente bene un timbro

# Esempio

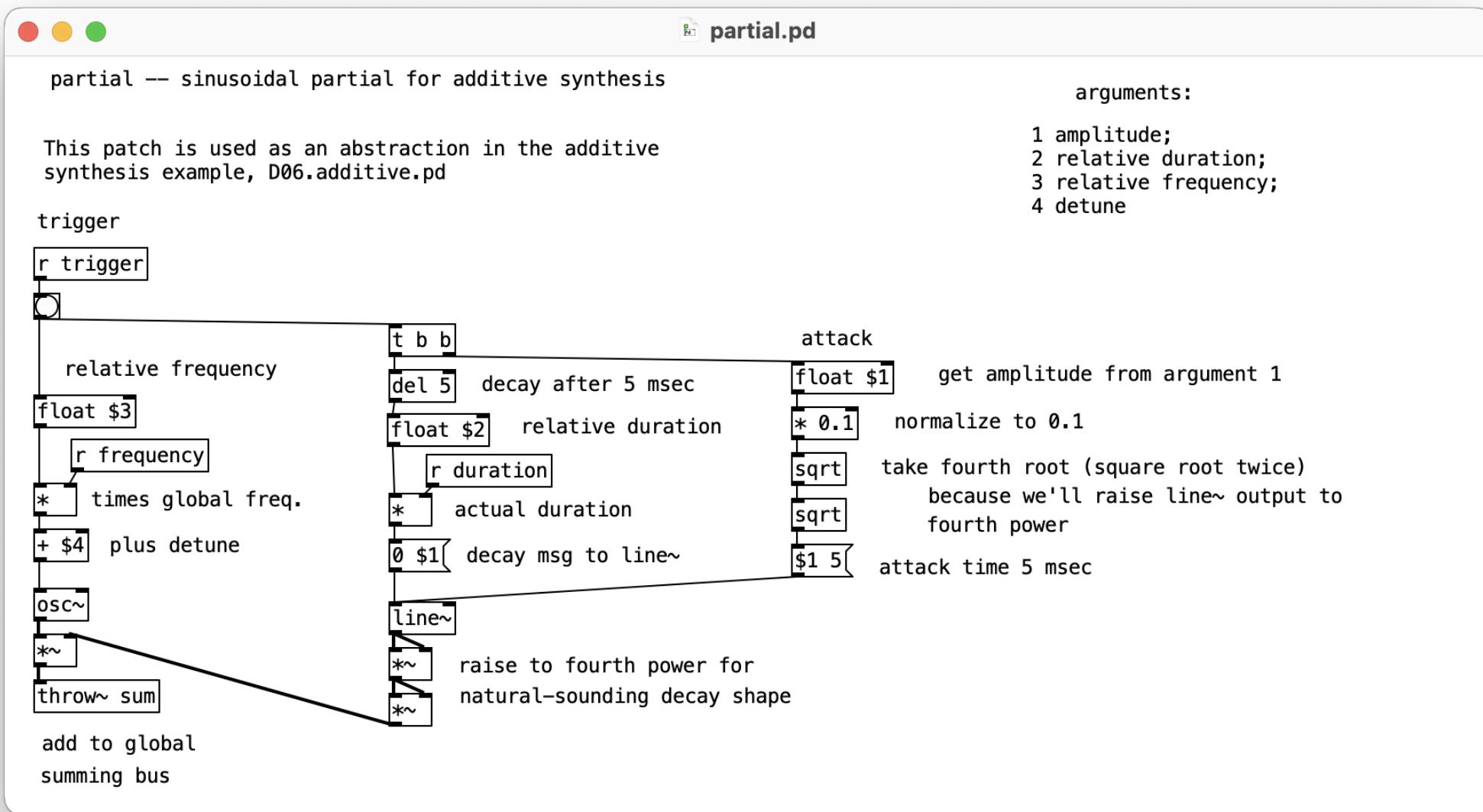
- Risset, Bell Labs, 1964
- Analisi e sintesi di suoni tramite componenti in frequenza
- Parziali
  - ✓ Frequenza multipla di F0
    - + detune
  - ✓ Ampiezza multipla di A0
  - ✓ Durate relative
  - ✓ Attacco lineare rilascio esponenziale decr.

"Bell sound" misto armonico/disarmonico:  
- F0 x: 0.56\*, 0.92\*, 1.19, 1.7, 2, 2.74, 3, 3.76, 4.07  
- A0 x: 1/0.67, 1/1.8, 2.67, 1.67, 1.46, 1.33, 1.33, 1, 1.33



# Pure Data – partial

*Non è una "vera" subpatch  
perchè usa variabili globali.*



# Pure Data – subpatch (funzioni)

---

- Le subpatch svolgono il ruolo di "funzioni"
- Sono memorizzate in un file separato con lo stesso nome (.pd)
  - ✓ [partial] , partial.pd (nel path)
- Comunicano tramite inlets ed outlets
  - ✓ Possono essere definiti come oggetti inlets ed outlets mappati graficamente
  - ✓ Gli inlets sono anche mappati, da sx a dx, su argomenti \$1, \$2, ...
- E' possibile definire variabili
  - ✓ Poichè lo scope è globale, per evitare conflitti si consiglia di assegnare nomi preceduti da \$0 (il nome della subpatch), \$0-myvar

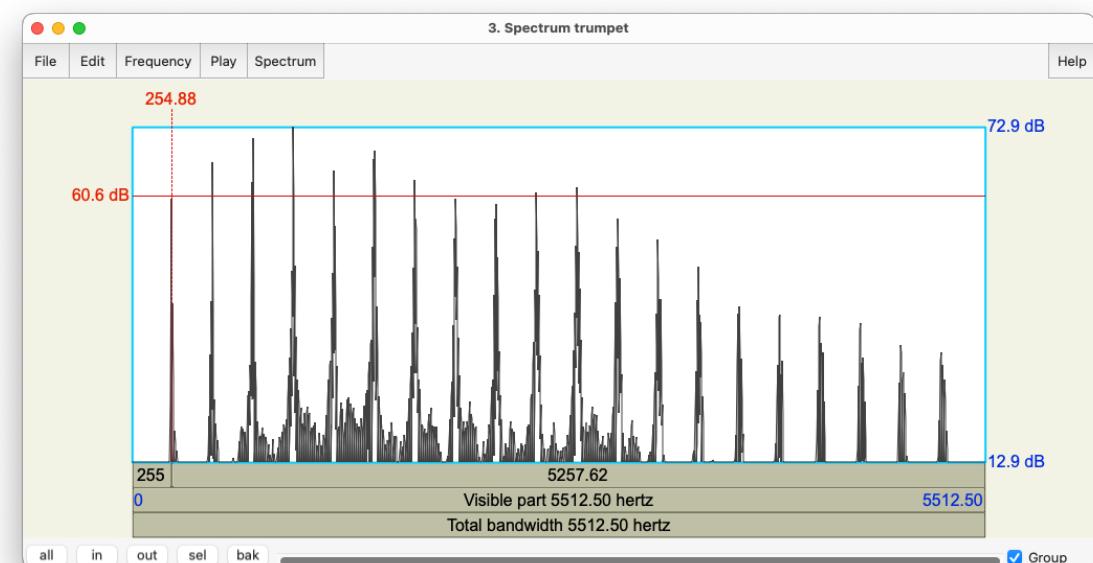
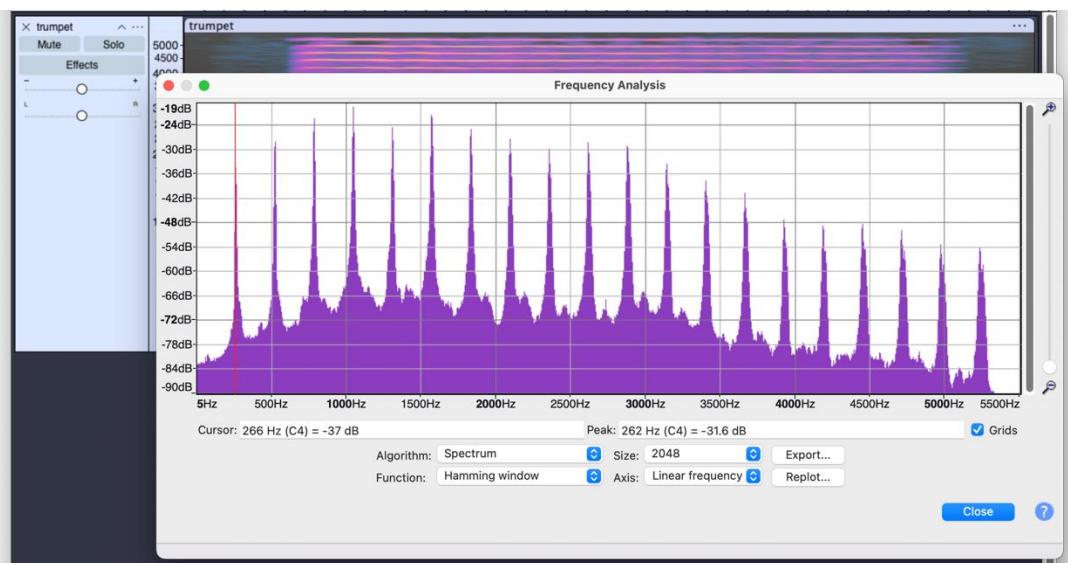
# Pure Data – send/receive throw/catch

---

- In Pure Data è possibile inviare/ricevere audio senza necessariamente creare una patch (linea) tramite
- `send~ <var>` / `receive~ <var>`
  - ✓ COPIA il segnale da send a receive (con la stessa `<var>`)
- `throw~ <var>` / `catch~ <var>`
  - ✓ Il segnale inviato da throw viene SOMMATO nel catch (con la stessa `<var>`)
  - ✓ Crea un bus / mix

# Esercizio – sintesi additiva

- Recuperare i **rapporti** di ampiezza e frequenza delle parziali dall'analisi dello spettro di un segnale
  - ✓ Se la misura è in dB occorre convertirlo in ampiezza



trumpet.wav

Riportare i rapporti di ampiezza e frequenza delle parziali sul Gsheet: ead<YY>.sintesi-additiva-trumpet

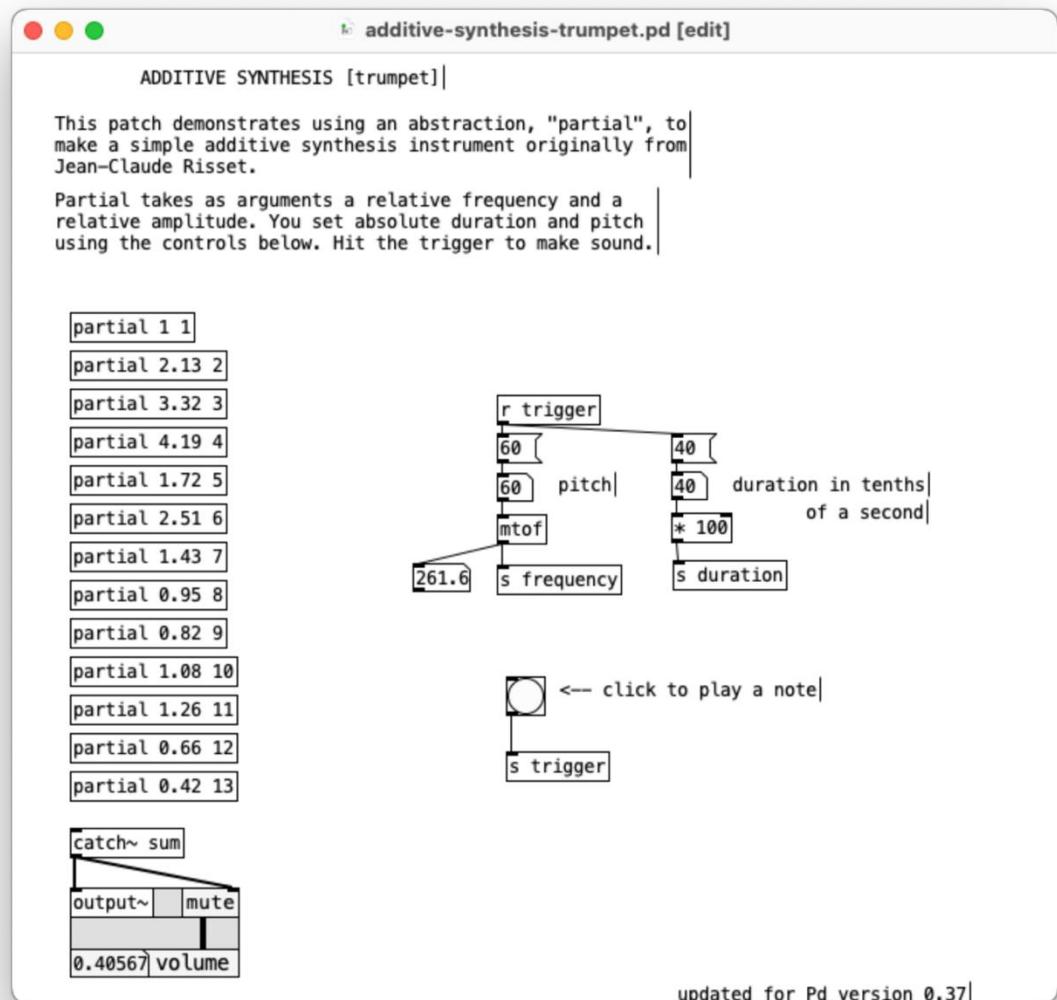
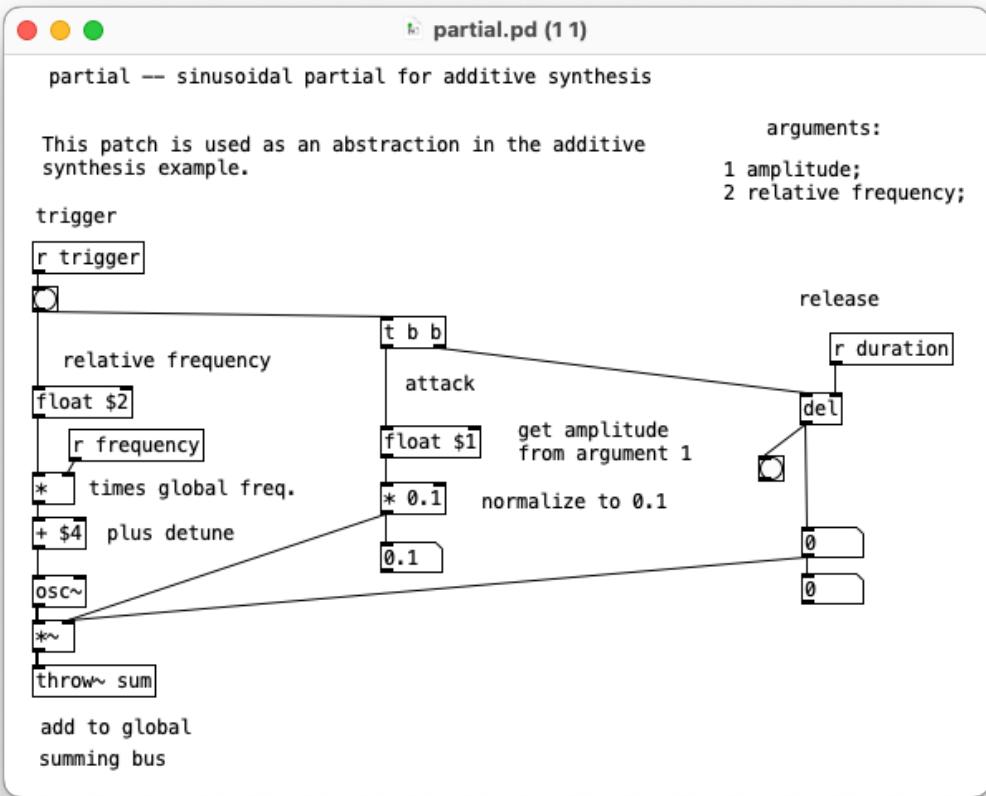
# Esercizio – sintesi additiva

---

- Sintetizzare il segnale come somma di parziali ciascuna con propria ampiezza e frequenza
- Implementare "partial" in pure data: [partial < $r_a$ > < $r_f$ >]
  - ✓ Riceve ampiezza e frequenza relative della componente spettrale
  - ✓ Sintetizza la componente tramite un oscillatore
    - L'oscillatore avrà frequenza  $r_f * F_0$
    - L'ampiezza varrà  $r_a * A_0$
    - Dopo una certa durata l'ampiezza diventerà 0 per "spegnere" il suono
  - ✓ Si utilizzino send/receive throw/catch per semplificare il patching
  - ✓ N.B. no detune, no durate relative, no inviluppo temporale

# Esercizio – sintesi additiva

## Soluzione



# Scipy – find\_peaks

---

## ■ parameters

- ✓ x: sequence
- ✓ height: minimal required height
- ✓ threshold: minimal vertical distance to its neighboring samples
- ✓ distance: required minimal horizontal distance *in samples* between neighbouring peaks
- ✓ ...

[Home](#) > SciPy API > Signal processing ([scipy.signal](#)) > [find\\_peaks](#)

## ■ return values

- ✓ peaks: indices of peaks in x
- ✓ properties (dict):
  - peak\_heights: the height of each peak
  - ...

scipy.signal.  
**find\_peaks**

**find\_peaks**(x, height=None, threshold=None, distance=None,  
prominence=None, width=None, wlen=None, rel\_height=0.5,  
plateau\_size=None)

[\[source\]](#)

Find peaks inside a signal based on peak properties.

This function takes a 1-D array and finds all local maxima by simple comparison of neighboring values. Optionally, a subset of these peaks can be selected by specifying conditions for a peak's properties.

# Scipy- find\_peaks

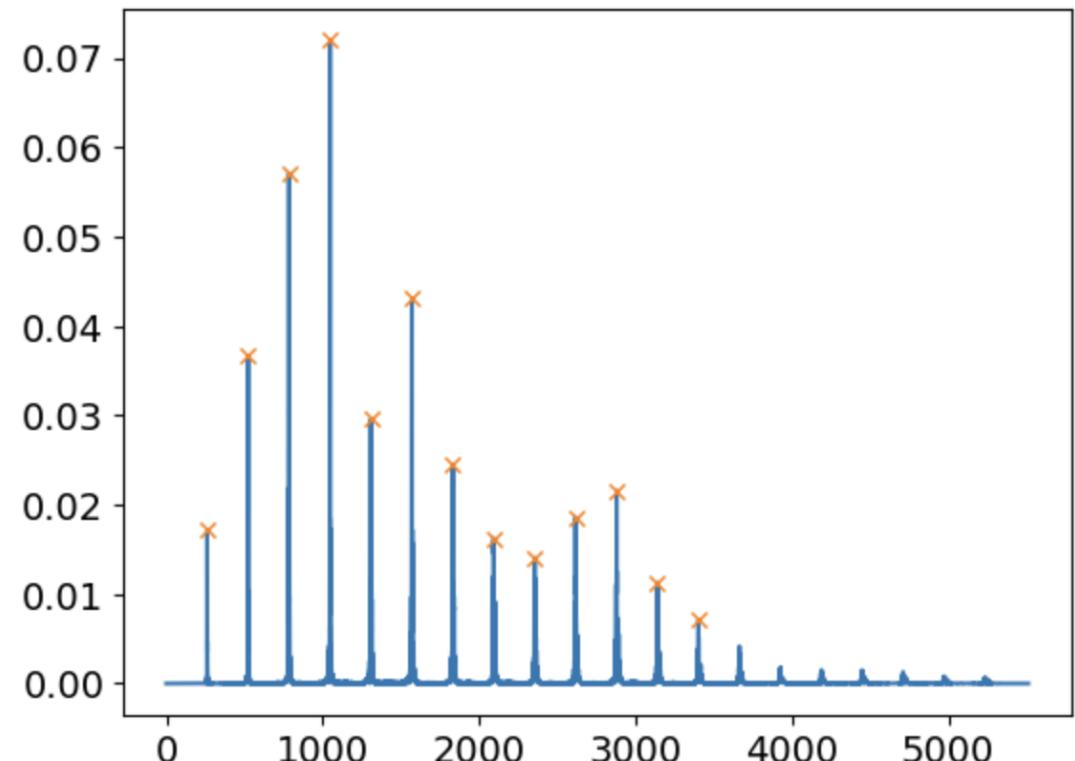
- `peaks, _ = find_peaks(spectrum, height=0.005, distance=100)`

```
y, sr = librosa.load('media/trumpet.wav', sr=None)
spectrum, freqs = dft(y, sr)
peaks, _ = find_peaks(spectrum, height=0.005, distance=100)
plt.plot(freqs, spectrum)
plt.plot(freqs[peaks], spectrum[peaks], 'x')

A0 = spectrum[peaks[0]]
F0 = freqs[peaks[0]]
print(f"F0: {F0:.1f} Hz")
for i in range(len(peaks)):
    pk = peaks[i]
    fk = freqs[pk]
    F0_ratio = np.round(freqs[pk]/F0,2)
    print(f"{i} - {F0_ratio}: {spectrum[pk]/A0:.2f}")
```

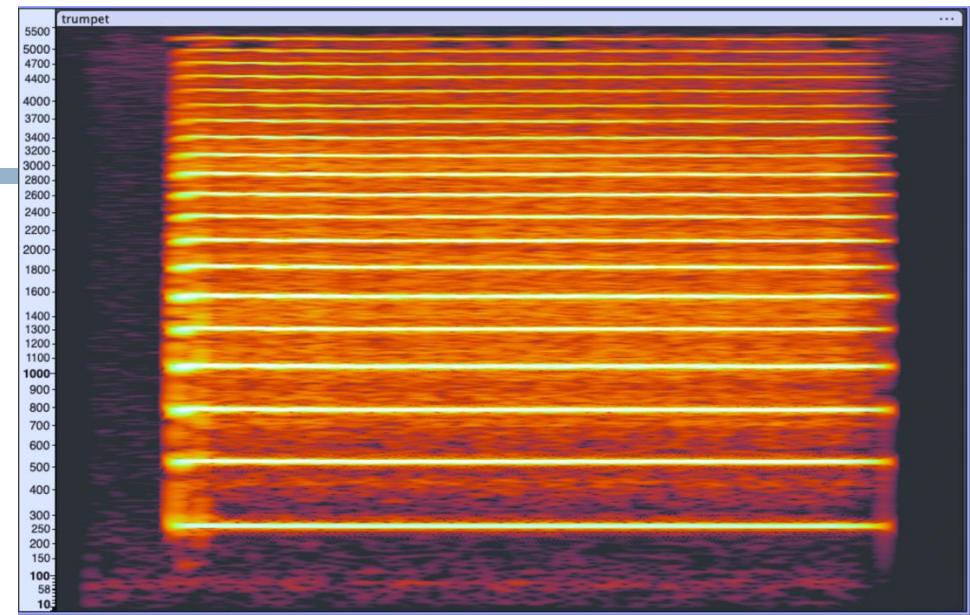
F0: 261.8 Hz  
0 - 1.0: 1.00 - 0  
1 - 2.0: 2.13 - 0  
2 - 3.0: 3.32 - 0  
3 - 4.0: 4.19 - 0  
4 - 5.0: 1.72 - 0  
5 - 6.0: 2.51 - 0  
6 - 7.0: 1.43 - 0  
7 - 8.0: 0.95 - 0  
8 - 9.0: 0.82 - 0  
9 - 10.0: 1.08 - 0  
10 - 11.0: 1.26 - 0  
11 - 12.0: 0.66 - 0  
12 - 13.0: 0.42 - 0

trumpet-orig.wav

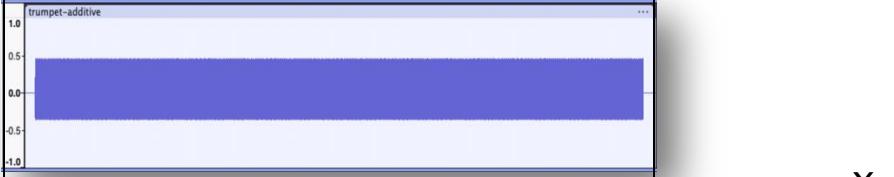


# Semplificazioni e limiti

- Si è sintetizzato un suono statico
- Il suono invece cambia nel tempo
- Ogni componente spettrale dovrebbe avere una curva che modella la sua variazione di intensità nel tempo

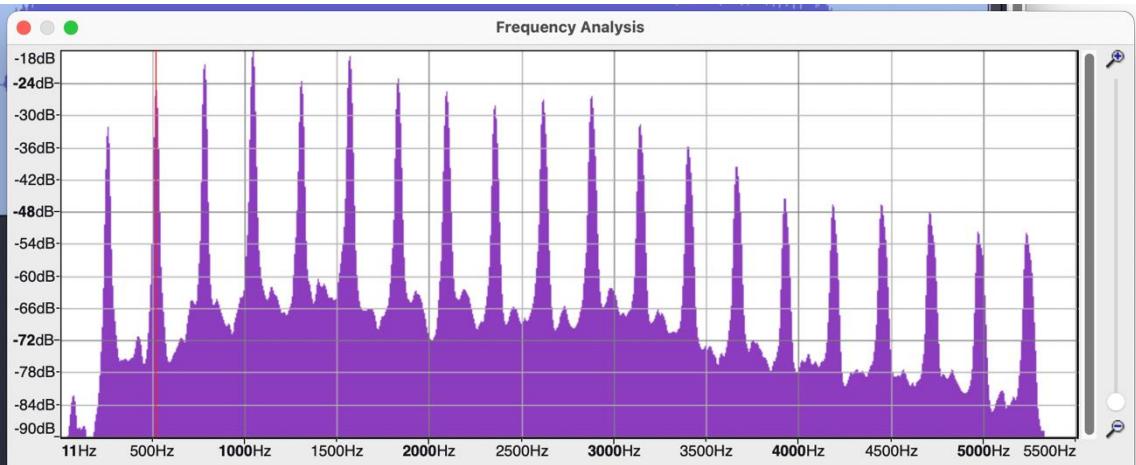


*Abbiamo sintetizzato "la media"  
più o meno corrisponde  
alla parte stazionaria più lunga*

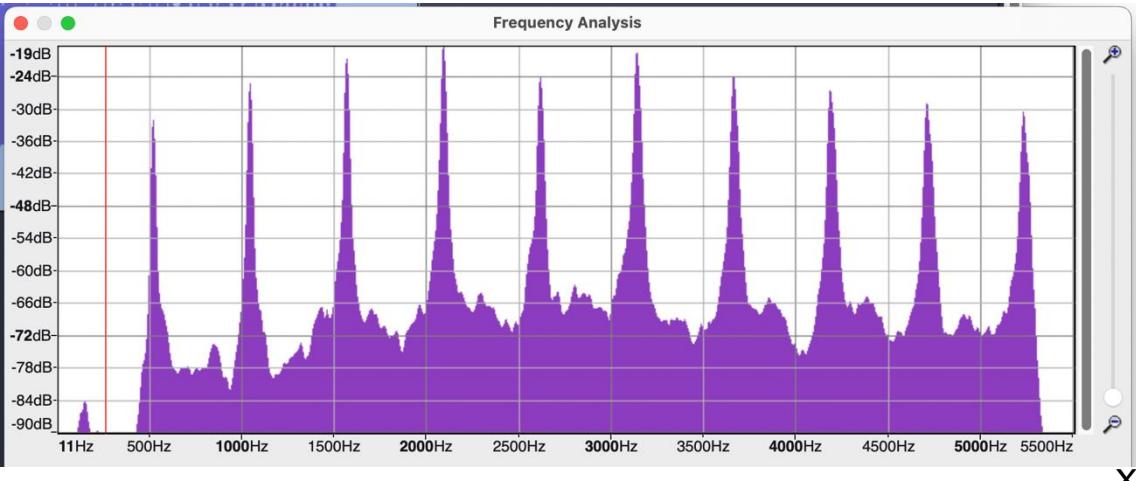
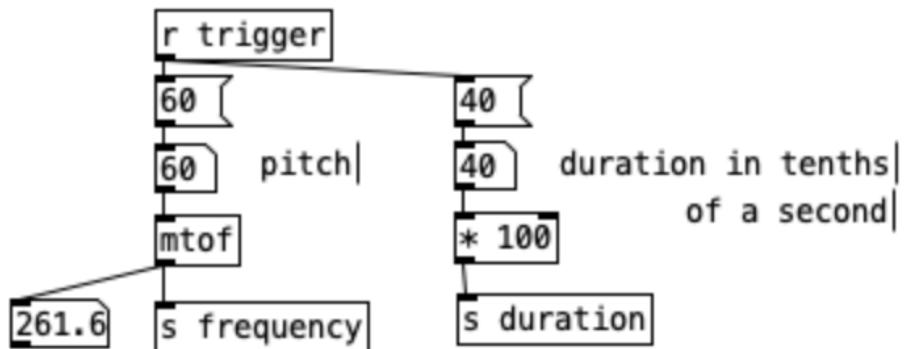


# Variazione dell'altezza (pitch)

- La sintesi additiva permette di modificare facilmente l'altezza di un suono traslando le sue armoniche senza alterarne la durata



**CAVEAT**



# Effetto chipmunk

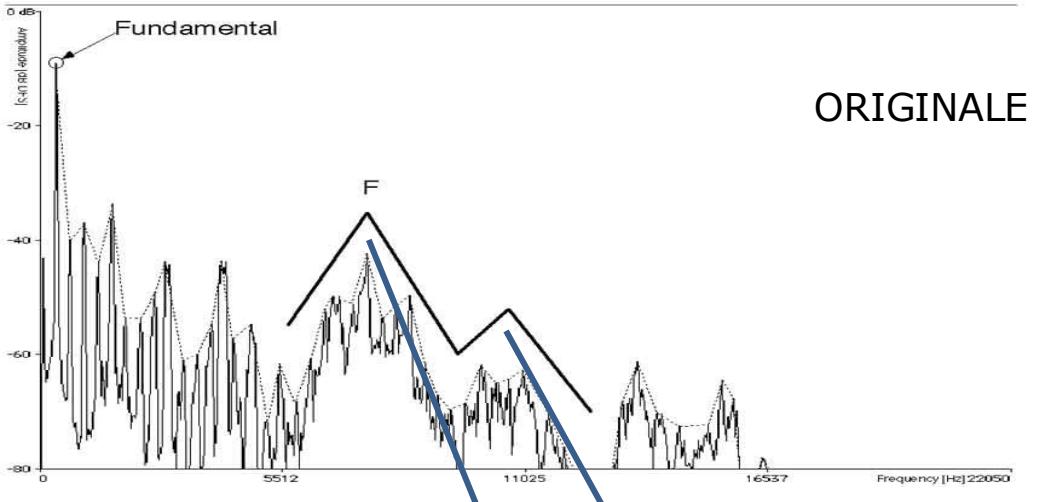
---

- La semplice traslazione di un suono armonico di una ottava: intesa come la semplice moltiplicazione x2 delle frequenze delle sue armoniche
  - ✓ Es. 200, 400, 600, 800 => 400, 800, 1200, 1600
- Alza la fondamentale (da 200 a 400) ma non ne preserva il timbro
  - ✓ Crea il cosiddetto effetto "chipmunk"
- Il timbro è dato dall'inviluppo spettrale che, traslando le armoniche, non viene preservato
  - ✓ Le formanti (picchi dell'inviluppo) si spostano anche esse x2

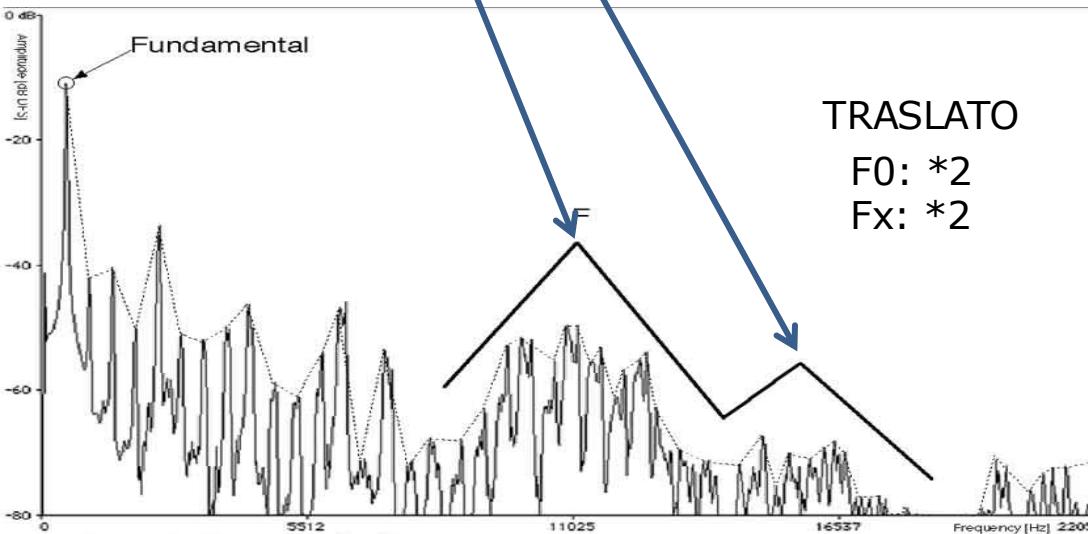
---

[1] <https://blogs.zynaptiq.com/bernsee/formants-pitch-shifting/> - orig2.wav, shifted2.wav, shftfrm2.wav

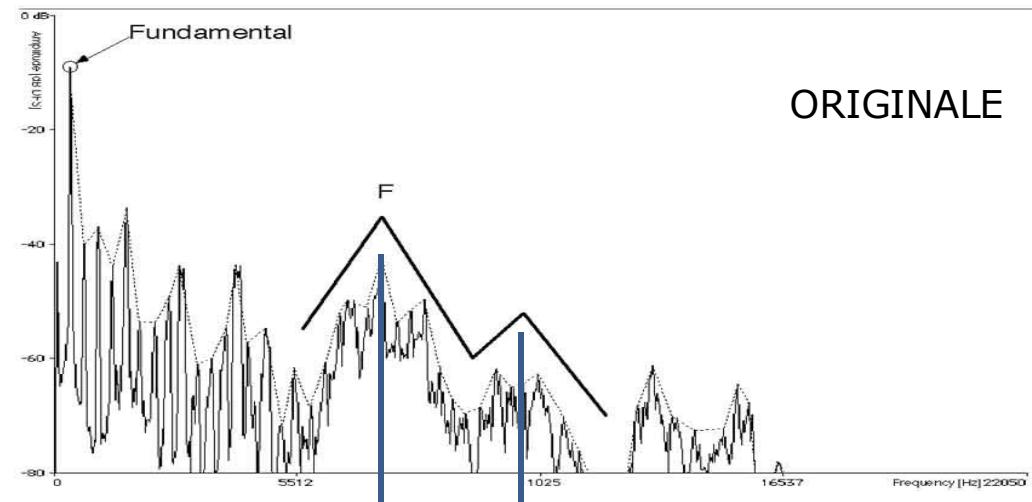
# Esempio - chipmunk



ORIGINALE



TRASLATO  
F0: \*2  
Fx: \*2

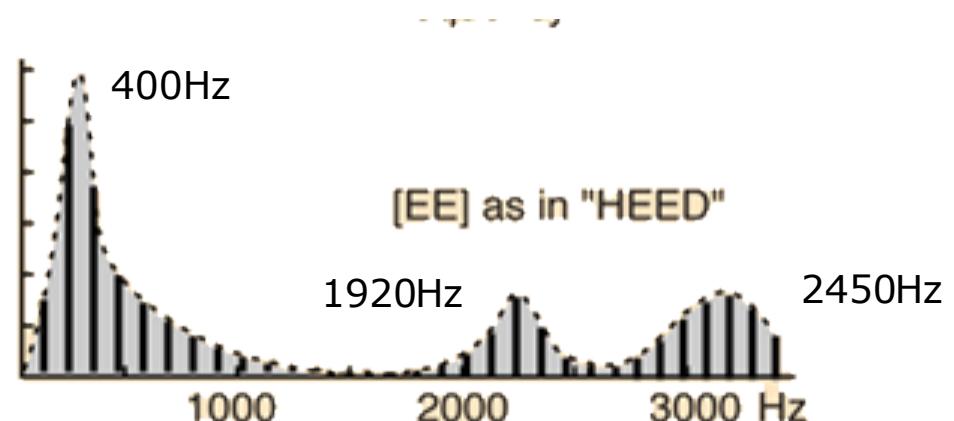
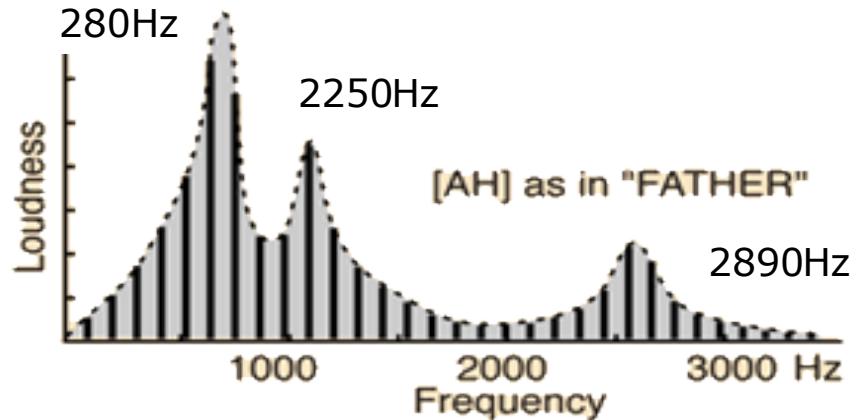


ORIGINALE

TRASPOSTO  
F0: \*2  
Fx: stabili

# Formanti

- Le formanti non sono "singole frequenze", ma regioni dello spettro dove l'energia del suono è più alta
- Le formanti modellano l'inviluppo spettrale

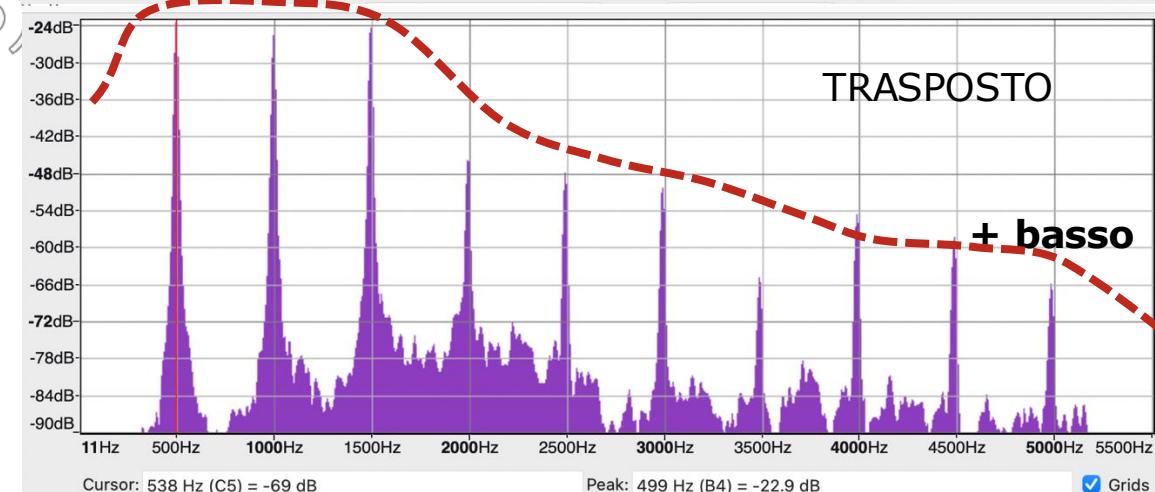
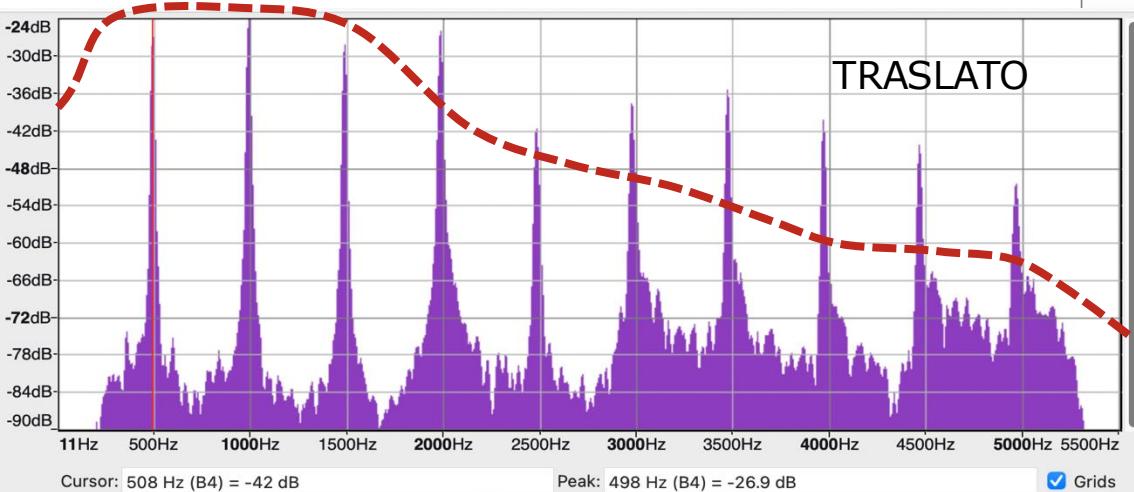
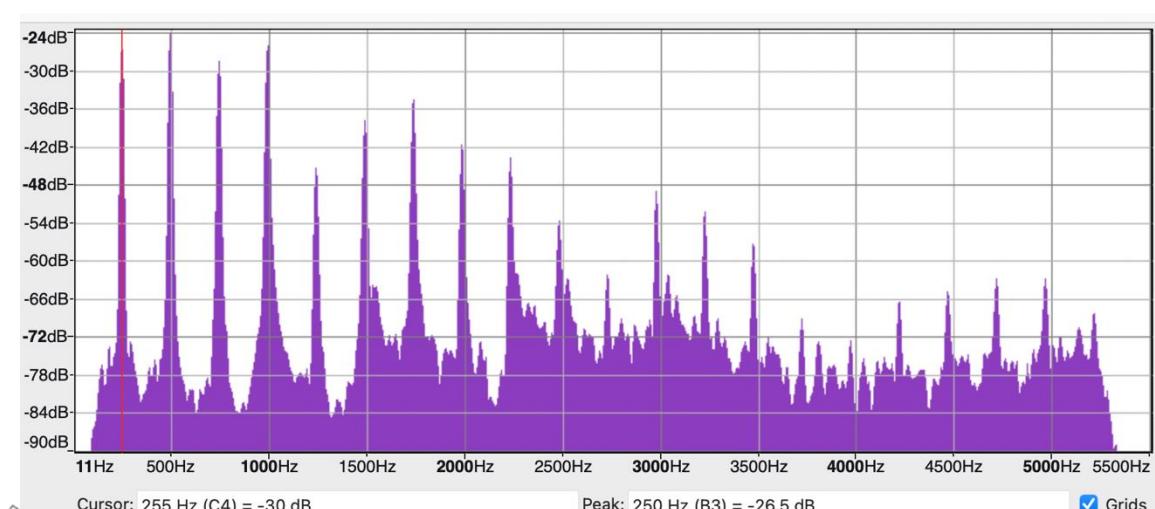
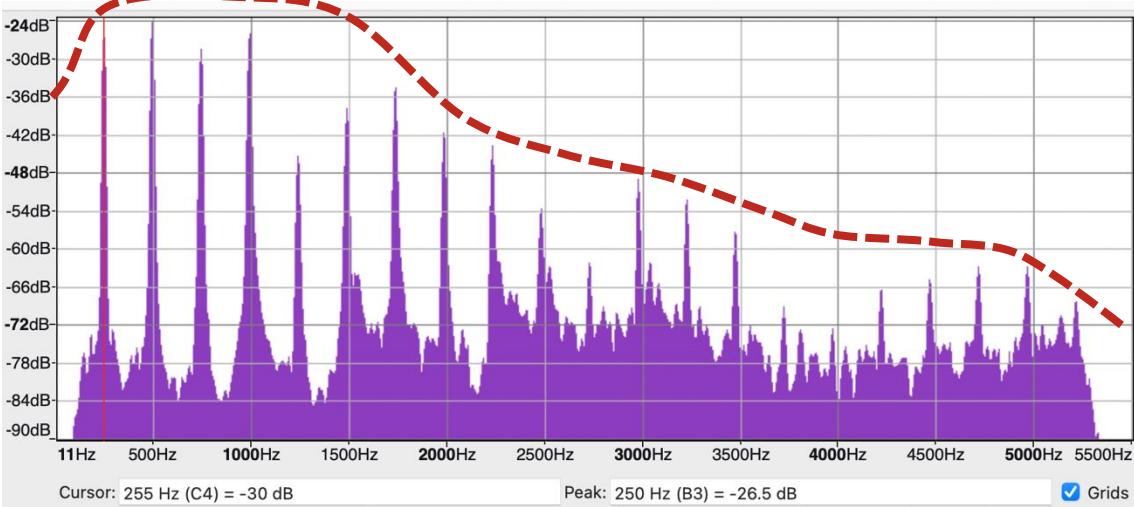


- Le formanti
  - ✓ *non dipendono dalla frequenza di vibrazione*
  - ✓ **ma dipendono dalle risonanze (del corpo dello strumento)**

[1] <http://hyperphysics.phy-astr.gsu.edu/hbase/Music/vowel.html>

# Esempio - Flute

Traslato in Audacity:  
cambio rate e resample



Flute.nonvib.ff.B3.mono.wav    Flute.nonvib.ff.B4.mono.wav    Flute.nonvib.ff.B4-chipmunk.mono.wav

# Analisi e ri-sintesi

---

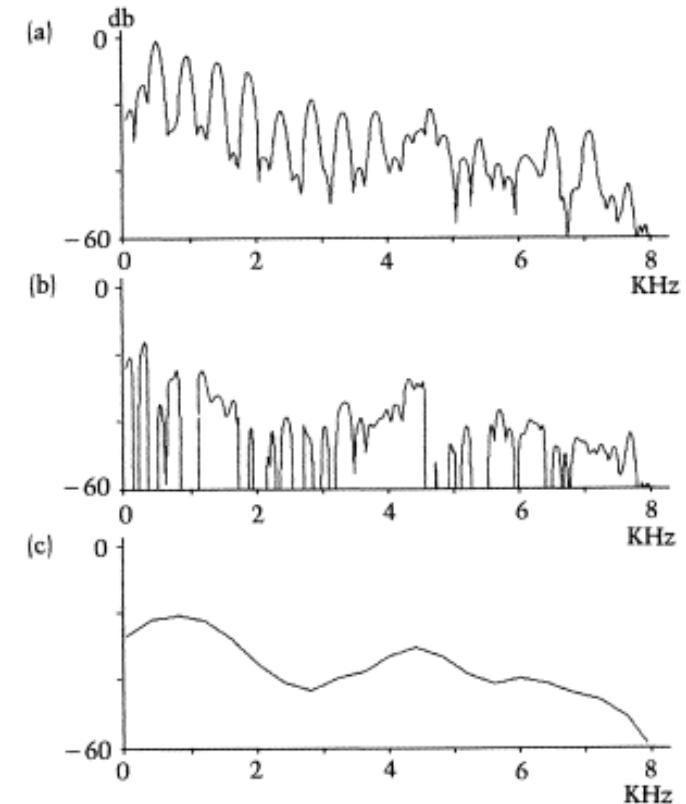
- Tramite l'analisi dello spettro di un suono è possibile scomporlo nelle sue componenti in frequenza per poi ricostruirlo
- Uno dei modelli di "analisi e sintesi" è detto "sinusoid plus noise"
- Assume che lo spettro sia composto da
  - ✓ Una parte tonale: le armoniche
  - ✓ Una parte non tonale: residuo stocastico (rumore)
- Può quindi essere ricostruito ad una intonazione diversa
  - ✓ Le armoniche vengono distanziate alla nuova F0 ma occorre preservarne l'inviluppo spettrale
  - ✓ La parte non tonale può rimanere identica

Reference: X. Serra, "Spectral Modeling Synthesis:  
A Sound Analysis/Synthesis System Based on Deterministic plus Stochastic Decomposition"

---

# Analisi e ri-sintesi

- Nel modello spettrale "sinusoid plus noise" la parte non tonale (b) è derivata sottraendo allo spettro originale (a) lo spettro armonico ricostruito
- Non essendo necessario preservarne il dettaglio, viene modellata semplicemente con il suo inviluppo



Reference: X. Serra, "Spectral Modeling Synthesis: A Sound Analysis/Synthesis System Based on Deterministic plus Stochastic Decomposition"

---

Sintesi sottrattiva

# Sintesi sottrattiva

Verrà trattata nell'unità di  
"elaborazione audio"

- Generazione a partire da un **segnale ricco di armoniche, o rumore, o entrambi**, a cui si sottraggono delle parti di spettro usando filtri per modellare il timbro
  - ✓ Si ispira al comportamento fisico degli strumenti acustici (e dell'apparato di fonazione umano)

