UNIVERSITÀ
DEGLI STUDI DI TRIESTE

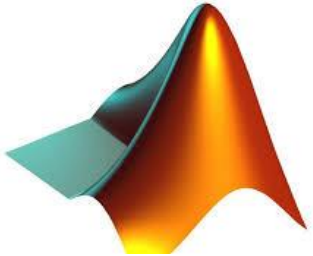
dipartimento
di ingegneria
e architettura

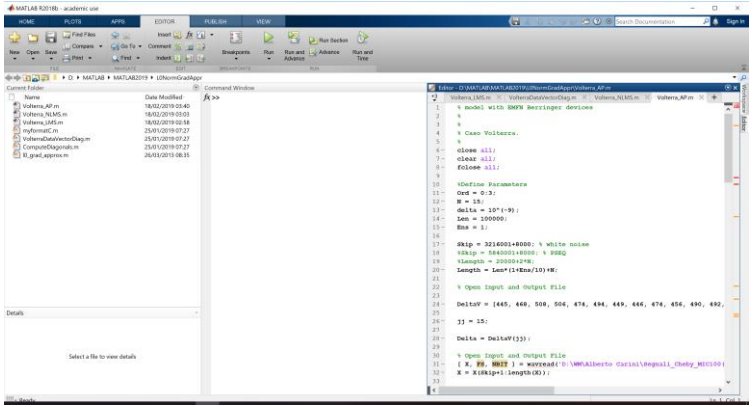
Basi di elaborazione audio

A.Carini – Elettronica per l'audio e l'acustica

I tool

- Matlab





- Audacity

Formati dati dei campioni

- Matlab → floating point (double)
- Altri tool → floating point / fixed point

- Fixed point: bassa occupazione di memoria ma possibili problemi per underflow o overflow, rumore di quantizzazione.
- Floating point: maggiore occupazione di memoria, richiede comunque conversione da/a fixed point in lettura scrittura file o in I/O su ADC/DAC.

Numero di bit e frequenza di campionamento

Table 1.1 Sampling characteristics of common applications.

Application	Sample rate, resolution	How used
Telephony	8 kHz, 8–12 bits	64 kbps A-law or μ-law
Voice conferencing	16 kHz, 14–16 bits	64 kbps SB-ADPCB
Mobile phone	8 kHz, 14–16 bits	13 kbps GSM
Private mobile radio	8 kHz, 12–16 bits	<5 kbps, e.g. TETRA
Long-play audio	32 kHz, 14–16 bits	Minidisc, DAT, MP3
CD audio	44.1 kHz, 16–24 bits	Stored on CDs
Studio audio	48 kHz, 16–24 bits	CD mastering DAT, DVD
Very high end	96 kHz, 20–24 bits	For *golden ears* listening

- Range dinamico:

$$\mathrm{DR\ (dB)} = 6.02 \times n.$$

Vedere:

- Ian Vince McLoughlin, “Speech and Audio Processing”- Cambridge Univesity Press (2016)
  - Cap. 1.3-1.4

Matlab: sound

- I campioni che leggiamo da un file hanno in genere un formato intero a 16 bit (da -32768 a 32767) e vengono scalati dal Matlab tra [-1, +1].
- Playback di un vettore di campioni scalati con freq. di campionamento 8 kHz:

```
sound(speech, 8000);
```

- Imponendo scalaggio campioni:

```
soundsc(speech, 8000);
```

=

```
sound(speech/max(abs(speech)), 8000);
```

Matlab: plot

```
plot(speech);
```

```
plot( [ 1: size(speech) ]  / 8000, speech);
```

Audio file formats

Wave: The Wave file format is usually identified by the file extension .wav, and actually can hold many different types of audio data identified by a header field at the beginning of the file. Most importantly, the sampling rate, number of channels and number of bits in each sample are also specified. This makes the format very easy to use compared with other formats that do not specify such information, and thankfully this format is recognised by MATLAB. Normally the wave file would contain pulse coded modulation (PCM) data, with a single channel (mono), and 16 bits per sample. The sample rate could vary from 8000 Hz up to 48 000 Hz. Some older PC recording hardware is limited in the sample rates supported, but 8000 Hz and 44 100 Hz are almost always supported. 16 000 Hz, 24 000 Hz, 32 000 Hz and 48 000 Hz are also reasonably common.

PCM and **RAW** hold streams of pulse coded modulation data with no headers or gaps. They are assumed to be single channel (mono) but the sample rate and number of bits per sample are not specified in the file – the audio researcher must remember what these are for each .pcm or .raw file that he or she keeps. These can be read from and written to by MATLAB, but are not supported as a distinctive audio file. However, these have historically been the formats of choice for audio researchers, probably because research software written in C, C++ and other languages can most easily handle this format.

Audio file formats

A-law and **μ-law** are logarithmically compressed audio samples in byte format. Each byte represents something like 12 bits in equivalent linear PCM format. This is commonly used in telecommunications where the sample rate is 8 kHz. Again, however, the .au file extension (which is common on UNIX machines, and supported under Linux) does not contain any information on sample rate, so the audio researcher must remember this. MATLAB does support this format natively.

Other formats include those for compressed music such as MP3 (see Infobox 2.2: *Music file formats* on page 15), MP4, specialised musical instrument formats such as MIDI (musical instrument digital interface) and several hundred different proprietary audio formats.

Audio file formats

MP3, represented by the file extension .mp3, is a standard compressed file format invented by the Fraunhofer Institute in Germany. It has taken the world by storm: there is probably more audio in this format than in any other. The success of MP3, actually MPEG (Motion Pictures Expert Group) version 1 layer 3, has spawned numerous look-alikes and copies.

One notable effort is the strangely named format **Ogg Vorbis**, which is comparable in functionality to MP3, but not compatible with it: it is solely designed to be an open replacement for MP3, presumably for anyone who does not wish to pay licence fees or royalties to the Fraunhofer Institute. As such it has enjoyed widespread adoption worldwide, but may require download of a separate audio codec for playback on computer.

- (AAC): Advance audio coding dell'MPEG, standard incluso in MPEG-2 e MPEG-4 (fino 48 canali con F_C fino a 96 kHz + fino 10 canali LFE a 120 Hz) .mp4
- (AC-3): algoritmo di compressione *lossy* della Dolby Digital (fino a 7 canali con frequenza di campionamento a 48 kHz + 1 canale LFE)
- FLAC: Free lossless audio codec
- ALAC: Apple lossless audio codec
- WMA: Windows media audio della Microsoft (base come MP3, pro multicanale, WMA lossless)

Matlab: audioread, fread

- Lettura file wav:

```
[y,fs]=audioread('myfile.wav');
```

- Lettura file raw/pcm:

```
fid=fopen('recording.pcm', 'r');
```

```
speech=fread(fid , inf , 'int16' , 0, 'ieee-le');
```

```
fclose(fid);
```

Matlab: audiowrite

- Scrittura file wav:

```
audiowrite('myfile.wav',y,fs);
```

- Scrittura e lettura file .mat:

```
save  myspeech.mat  speech speech2
```

```
load  myspeech.mat
```

Matlab: filter

- Filtraggio FIR:

```
y=filter(b, 1, x);
```

$$y(n) = b(1) \times x(n) + b(2) \times x(n-1) + b(3) \times x(n-2) + \cdots + b(m+1) \times x(n-m).$$

- `b` vettore riga contenente la risposta impulsiva

Matlab: filter

- Filtraggio IIR:

```
y=filter(b, a, x);
```

$$
\begin{aligned}
y(n) = {} & b(1) \times x(n) + b(2) \times x(n-1) + b(3) \times x(n-2) + \cdots \\
& + b(m+1) \times x(n-m) \\
& - a(2) \times y(n-1) - a(3) \times y(n-2) - \cdots \\
& - a(m+1) \times y(n-m).
\end{aligned}
$$

- `b` e `a` vettori riga contenenti numeratore e denominatore della funzione di trasferimento.
- `b` e `a` scalati per a(1) se diverso da 1.

Matlab: fft

- Implementa un algoritmo veloce per il calcolo della DFT:

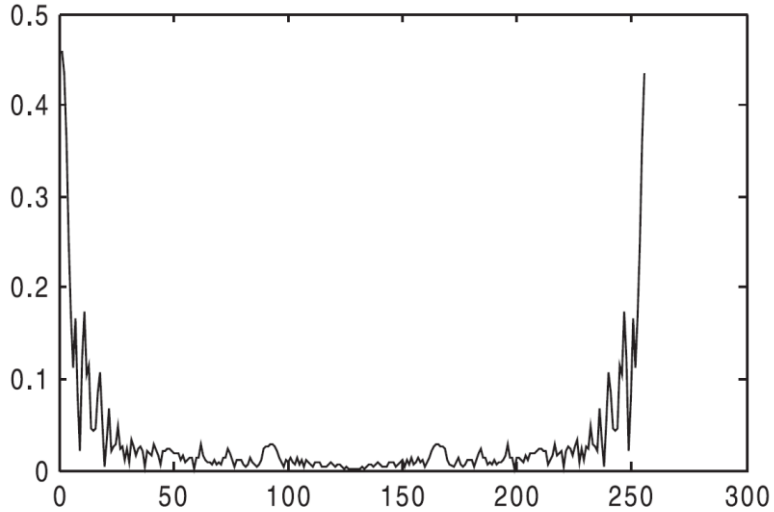
$$\mathcal{S}(k) = \sum_{n=0}^{N-1} x(n) e^{-2\pi jkn/N} \quad \text{for} \quad k = 0, 1, \ldots, \{N-1\}.$$

```
a_spec=fft(a_vector);
```

```
a_spec=fft(a_vector, 256);
```

Matlab: spettro di ampiezza

```
plot(abs(a_spec))
```



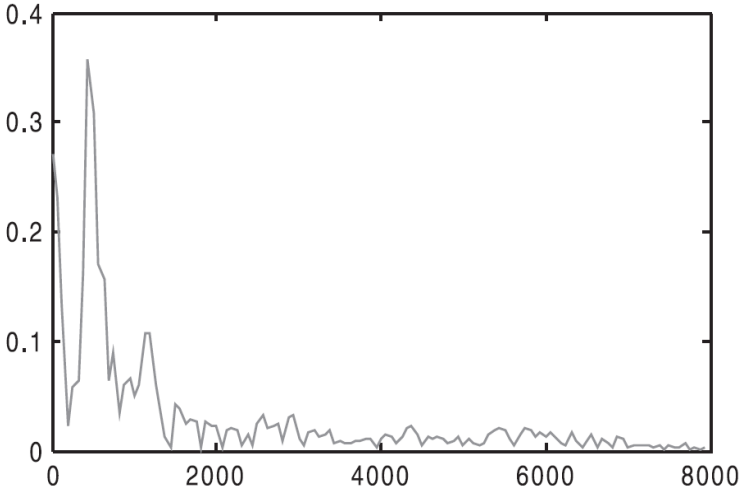

```
plot(abs(fftshift(a_spec)))
```

Matlab: spettro di ampiezza

```
plot([1:2*Fs/Ns:Fs], abs(a_spec(1:Ns/2)),'r')
```

NOOO!!!!!!

```
plot([0:Ns/2]*Fs/Ns, abs(a_spec(1:Ns/2+1)), 'r')
```



/2

Segmentazione: motivazioni

- Per l'elaborazione di segnali continui
- Per l'analisi di segnali tempo-varianti in cui le caratteristiche nel breve periodo sono importanti (fotografia istantanea)
- Se la complessità computazionale scala non-linearmente
- Se lo spazio di memoria è limitato
- Se si vuole distribuire l'elaborazione nel tempo invece che concentrarla alla fine
- Se bisogna limitare la latenza del sistema

Segmentazione con overlap

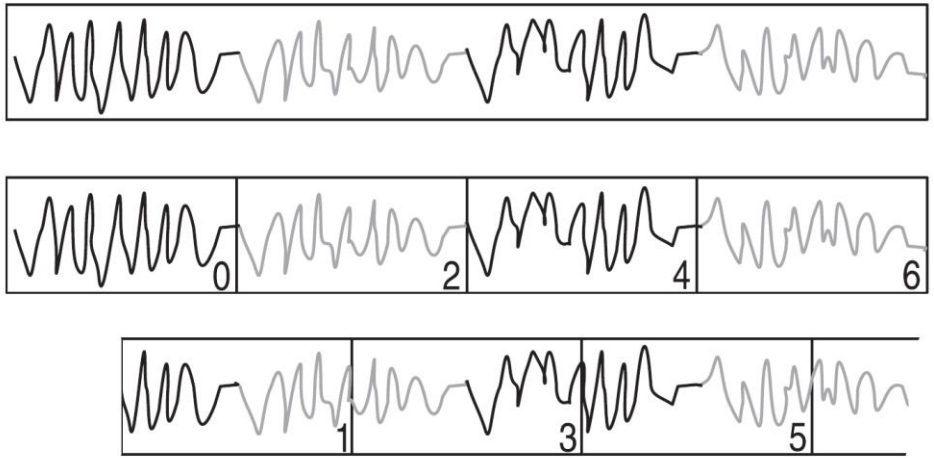


Figure 2.4 An audio recording (the upper waveform) is analysed by being divided into a sequence of seven frames (the two lower rows of waveforms) with 50% overlap. No short-term auditory feature is cut at the boundary between analysis frames. Instead it will appear unbroken in at least one analysis frame.

Ricostruzione: problema delle discontinuità ai bordi

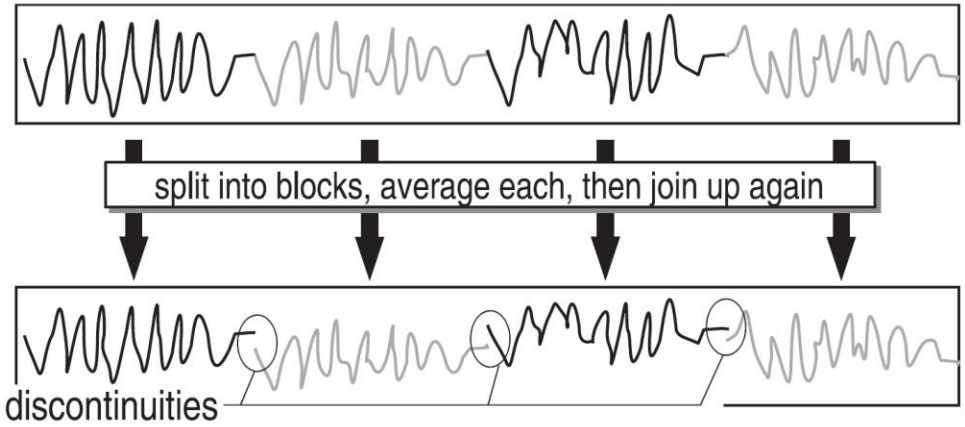
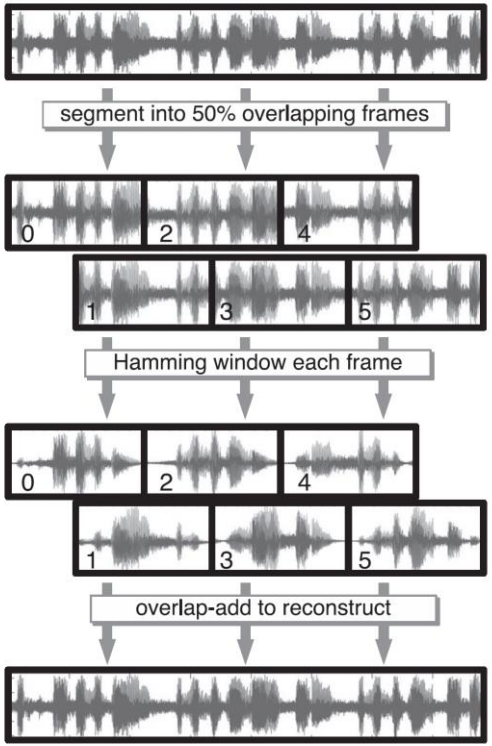
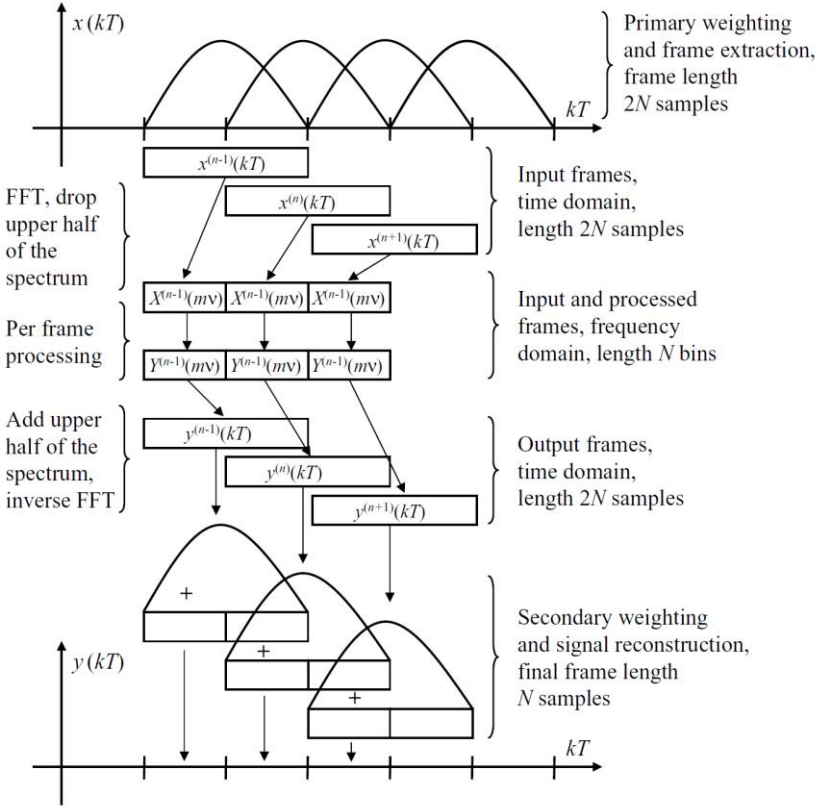


Figure 2.5 An audio recording (upper waveform) is split into equal-length analysis frames without overlap. Each frame is then normalised before being rejoined to create the lower waveform. The rejoined waveform exhibits obvious discontinuities at frame boundaries which would result in audible distortion.

Ricostruzione: problema delle discontinuità ai bordi



Funzione finestra e filtraggio



Analisi in frequenza: uso della funzione finestra

- Per calcolare lo spettro di un segnale abbiamo bisogno di tutti i campioni da -∞ a +∞.
- In pratica, osserviamo solo una finestra di *T L* secondi, *T = 1 / Fs, L* numero dei campioni raccolti.
- Limitare la finestra di osservazione limita la risoluzione in frequenza a $\frac{1}{T\ L}$ Hz
- Diviene impossibile distinguere due frequenze con separazione < $\frac{1}{T\ L}$ Hz

Analisi in frequenza: uso della funzione finestra

$$\hat{x}(n) = x(n) \cdot w(n)$$

$$w(n) = \begin{cases} 1 & 0 \leq n \leq L-1 \\ 0 & \text{altrimenti.} \end{cases}$$

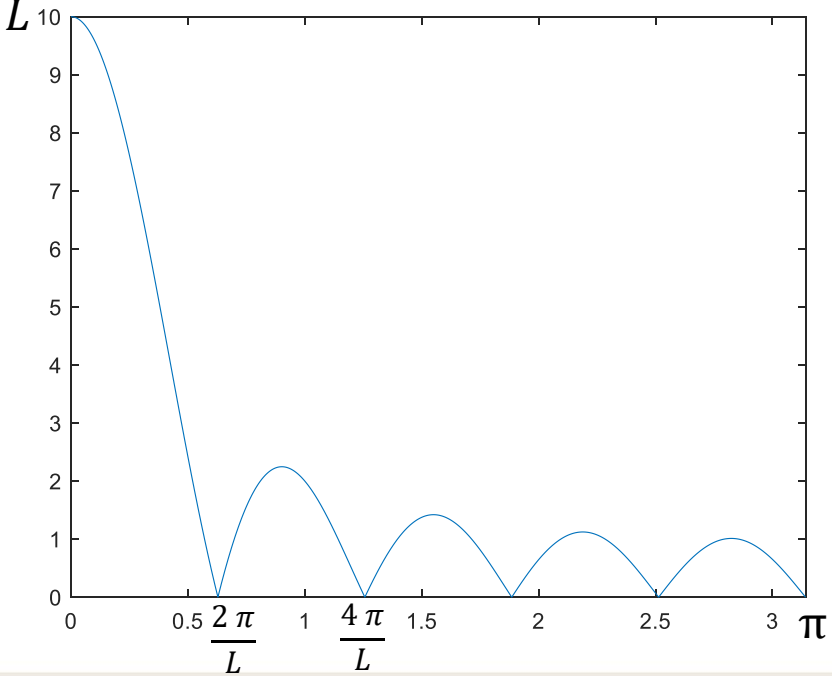
$$\hat{X}(e^{j\omega}) = \frac{1}{2\pi} \int_{-\pi}^{+\pi} X(e^{j\theta}) W(e^{j(\omega-\theta)}) \mathrm{d}\theta$$

Analisi in frequenza: uso della funzione finestra

$$\text{DTFT}\,[w(n)] = \sum_{n=0}^{L-1} e^{-j\omega n} = \frac{1-e^{-j\omega L}}{1-e^{-j\omega}} = \Big/ \cdot \frac{e^{j\omega\frac{L}{2}}e^{-j\omega\frac{L}{2}}}{e^{j\frac{\omega}{2}}e^{-j\frac{\omega}{2}}}$$

$$= \frac{e^{j\omega\frac{L}{2}} - e^{-j\omega\frac{L}{2}}}{e^{j\frac{\omega}{2}} - e^{j\frac{\omega}{2}}} \cdot e^{-j\omega\frac{L-1}{2}} =$$

$$= \frac{j\sin\left(\omega\frac{L}{2}\right)}{j\sin\left(\frac{\omega}{2}\right)} \cdot e^{-j\omega\frac{L-1}{2}}$$



Analisi in frequenza: uso della funzione finestra

$$x(n) = \cos(\omega_0 n) = \frac{1}{2}\left[e^{-j\omega_0 n} + e^{j\omega_0 n}\right]$$

$$\hat{X}(e^{j\omega}) = \frac{1}{2}\left[W(e^{j(\omega+\omega_0)}) + W(e^{j(\omega-\omega_0)})\right]$$

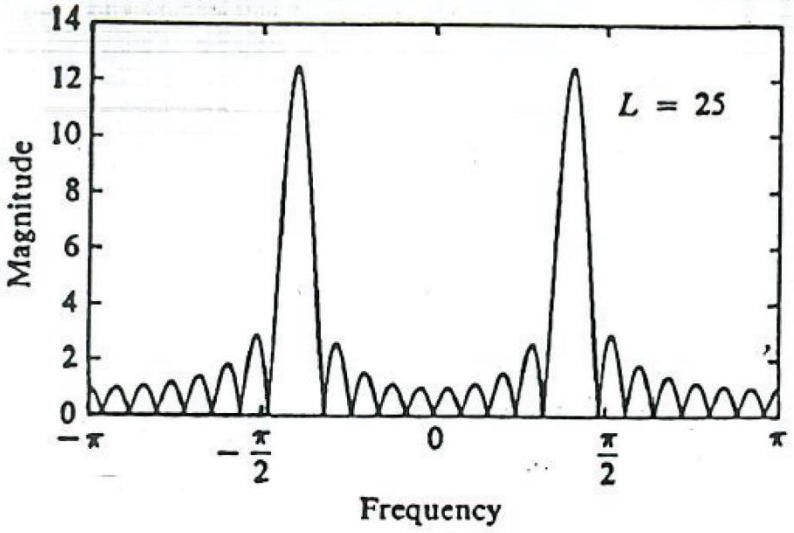


Figure 5.12 Magnitude spectrum for $L = 25$ and $n = 2048$, illustrating the occurrence of leakage.

Analisi in frequenza: uso della funzione finestra

- Consideriamo la somma di due sinusoidi con frequenze vicine:

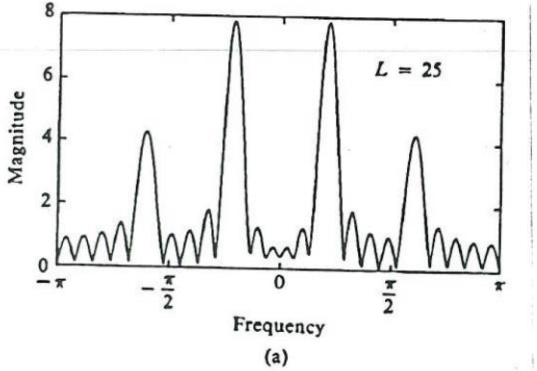
$$x(n) = \cos(\omega_1 n) + \cos(\omega_2 n)$$

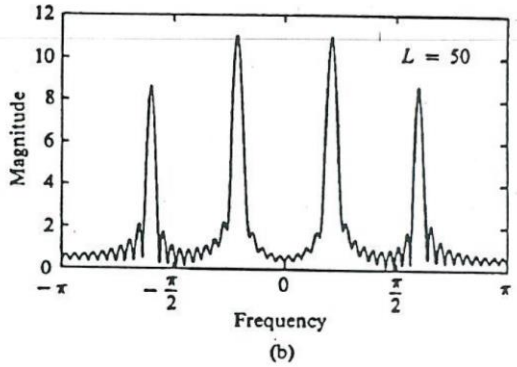
$$\hat{X}(e^{j\omega}) = \frac{1}{2}\left[W(e^{j(\omega+\omega_1)}) + W(e^{j(\omega+\omega_2)}) + W(e^{j(\omega-\omega_1)}) + W(e^{j(\omega-\omega_2)})\right]$$

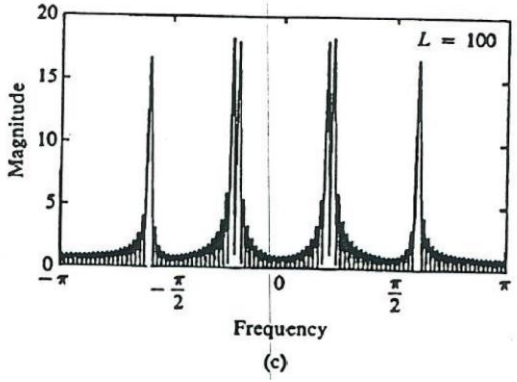
- Vedremo due lobi distinti solo quando:

$$|\omega_1 - \omega_2| \geq \frac{2\pi}{L}$$

Analisi in frequenza: uso della funzione finestra







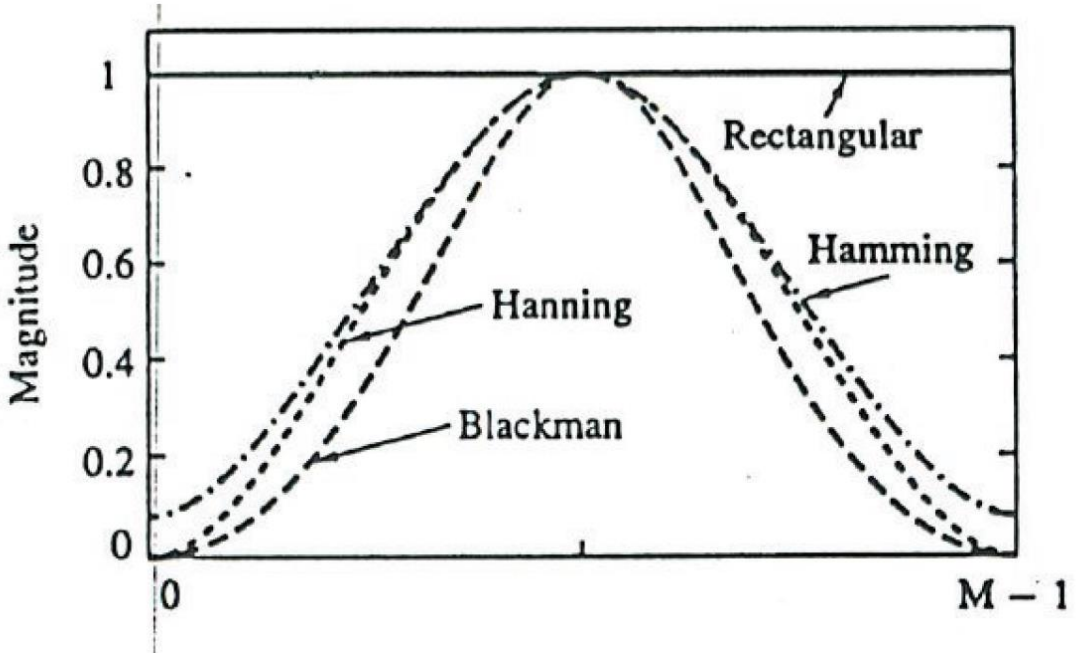
$$x(n) = \cos(\omega_0 n) + \cos(\omega_1 n) + \cos(\omega_2 n)$$

$$\omega_0 = 0.2\pi,\ \omega_1 = 0.22\pi,\ \text{e}\ \omega_2 = 0.6\pi$$

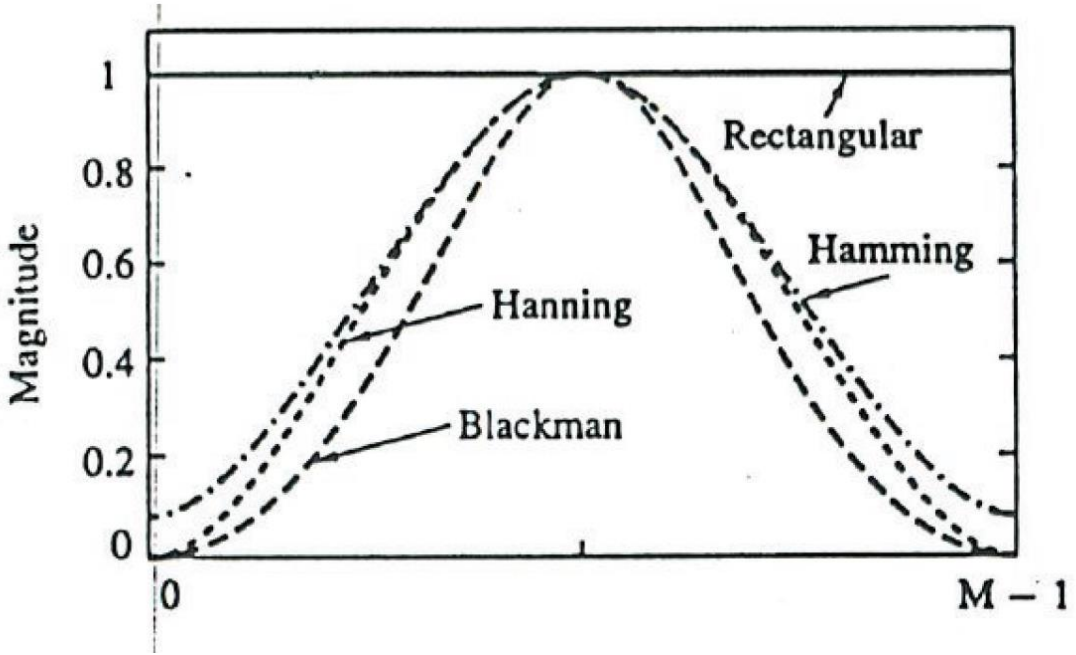
Analisi in frequenza: uso della funzione finestra

Name of window	Time-domain sequence, $h(n), 0 \le n \le M-1$
Bartlett (triangular)	$1 - \dfrac{2\left\lvert n - \dfrac{M-1}{2} \right\rvert}{M-1}$
Blackman	$0.42 - 0.5\cos\dfrac{2\pi n}{M-1} + 0.08\cos\dfrac{4\pi n}{M-1}$
Hamming	$0.54 - 0.46\cos\dfrac{2\pi n}{M-1}$
Hanning	$\dfrac{1}{2}\left(1 - \cos\dfrac{2\pi n}{M-1}\right)$

Analisi in frequenza: uso della funzione finestra



Analisi in frequenza: uso della funzione finestra



Analisi in frequenza: uso della funzione finestra

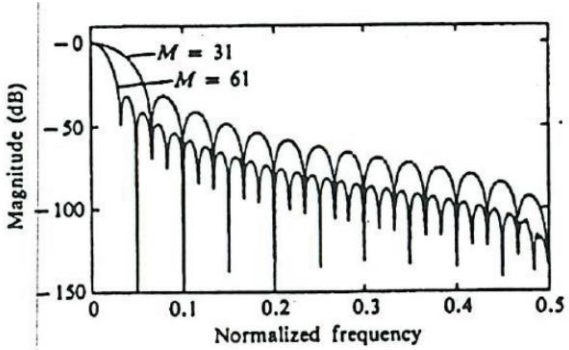


Figure 8.6 Frequency responses of Hanning window for (a) $M = 31$ and (b) $M = 61$.

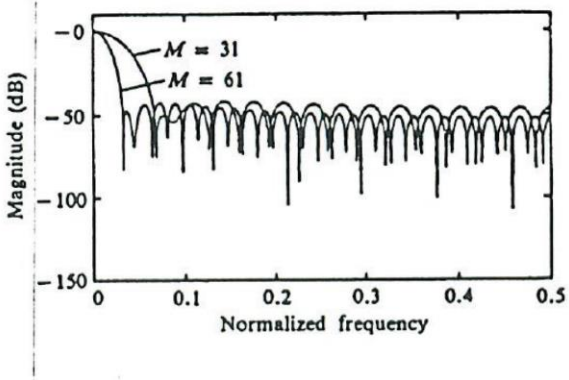


Figure 8.7 Frequency responses for Hamming window for (a) $M = 31$ and (b) $M = 61$.

Analisi in frequenza: uso della funzione finestra

TABLE 8.2 IMPORTANT FREQUENCY-DOMAIN CHARACTERISTICS OF SOME WINDOW FUNCTIONS

Type of window	Approximate transition width of main lobe	Peak sidelobe (dB)
Rectangular	$4\pi/M$	−13
Bartlett	$8\pi/M$	−27
Hanning	$8\pi/M$	−32
Hamming	$8\pi/M$	−43
Blackman	$12\pi/M$	−58

Short-time Fourier Transform (STFT) e spettrogramma

- La STFT è una trasformata di Fourier calcolata su una finestra limitata e applicata sequenzialmente sul segnale da analizzare.
- Esegue un'analisi nel tempo e nella frequenza del nostro segnale.
- Otteniamo una sequenza di spettri che possono diagrammati in funzione del tempo con una rappresentazione tridimensionale o uno *spettrogramma:*

$$\text{spectrogram } x(x) = |X(\tau, \omega)|^2.$$

```
subplot(3,1,3)
spectrogram(sr(2500:12500,1),128,0,128,24000,'yaxis')
```

Segnali e relativo spettrogramma

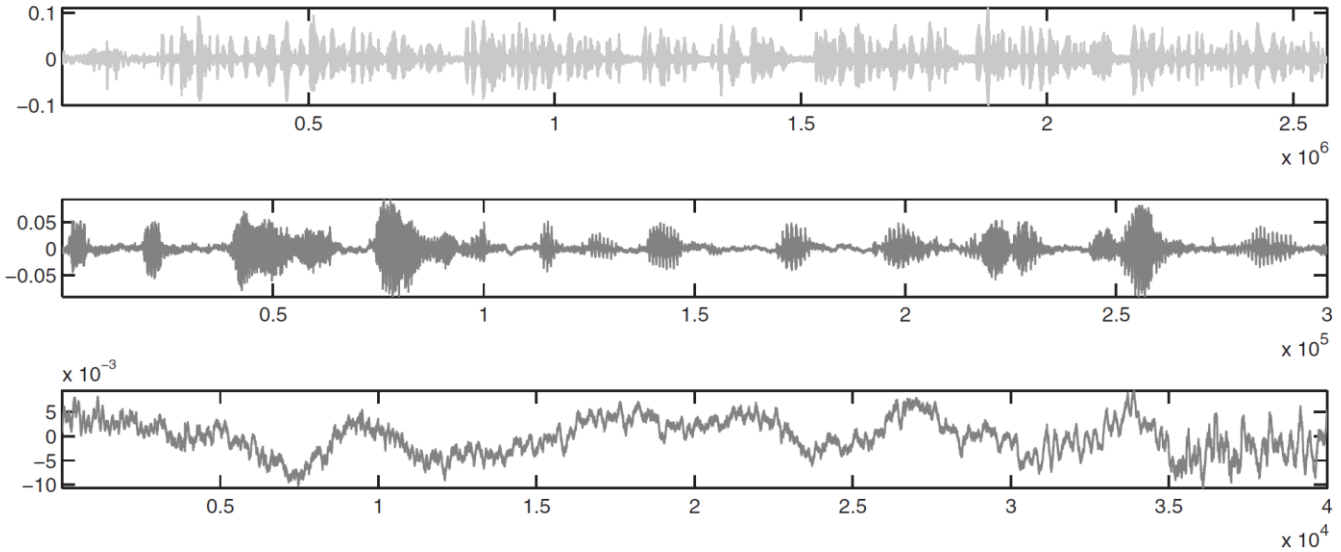
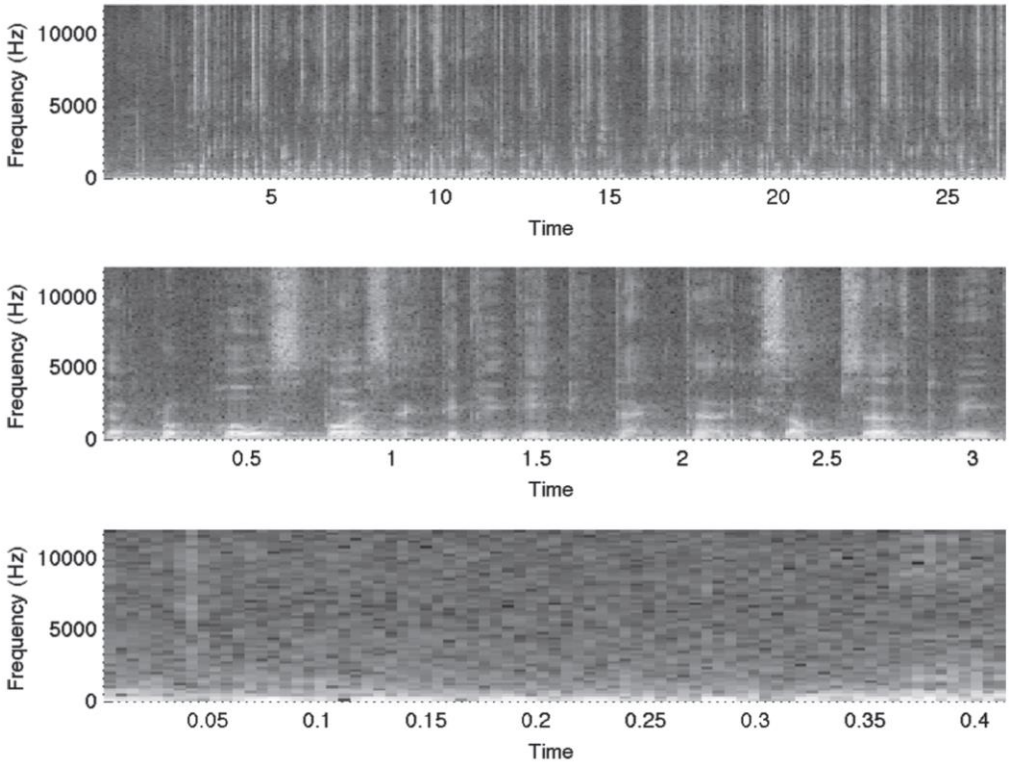


Figure 2.8 Amplitude against time plots of the same speech recording at three different time scales.

Segnali e relativo spettrogramma



Spettrogramma

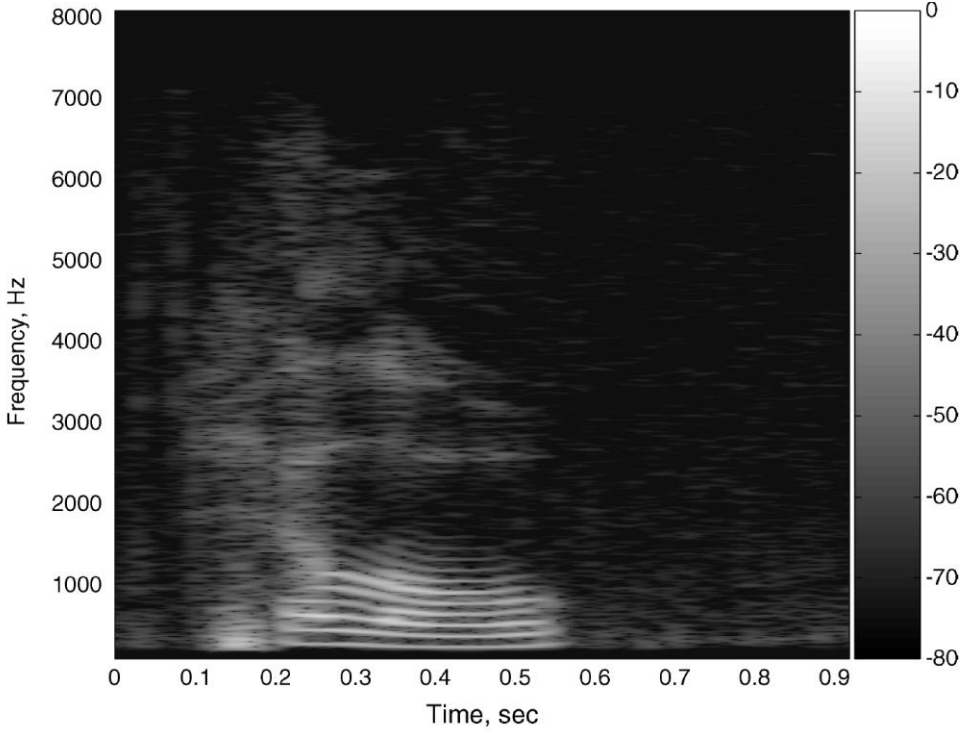


Figure 2.16 Spectrogram of English word "hello"

Correlogramma

- Il correlogramma è un diagramma dell'autocorrelazione di un segnale.
- Cross-correlazione tra due segnali:

$$c_{x,y}[t] = \sum_k x[k]y[t-k].$$

```
[c,lags]=xcorr(x,y);
```

- Autocorrelazione usata per trovare periodicità nel segnale (ad esempio per la stima del pitch di un suono vocalizzato)
- La cross-correlazione viene usata per trovare i tempi di ritardo tra segnali

Cepstrum

- Inizialmente definito come la trasformata di Fourier del logaritmo complesso della trasformata di Fourier di un segnale.
- Ceps-trum da spec-trum con l'inversione della prima parte
- Similmente le nuove frequenze vengono dette «quefrency»
- Oggi normalmente si usa l'IDTFT del logaritmo complesso della DFT del segnale:

$$x_{cc}(\theta) = \frac{1}{2\pi} \int_{-\pi}^{\pi} \ln(X(e^{j\omega}))e^{j\omega\theta} \mathrm{d}\omega,$$

- Cepstrum reale:

$$x_{cc}(\theta) = \frac{1}{2\pi} \int_{-\pi}^{\pi} \ln(|X(e^{j\omega})|)e^{j\omega\theta} \mathrm{d}\omega.$$

Cepstrum

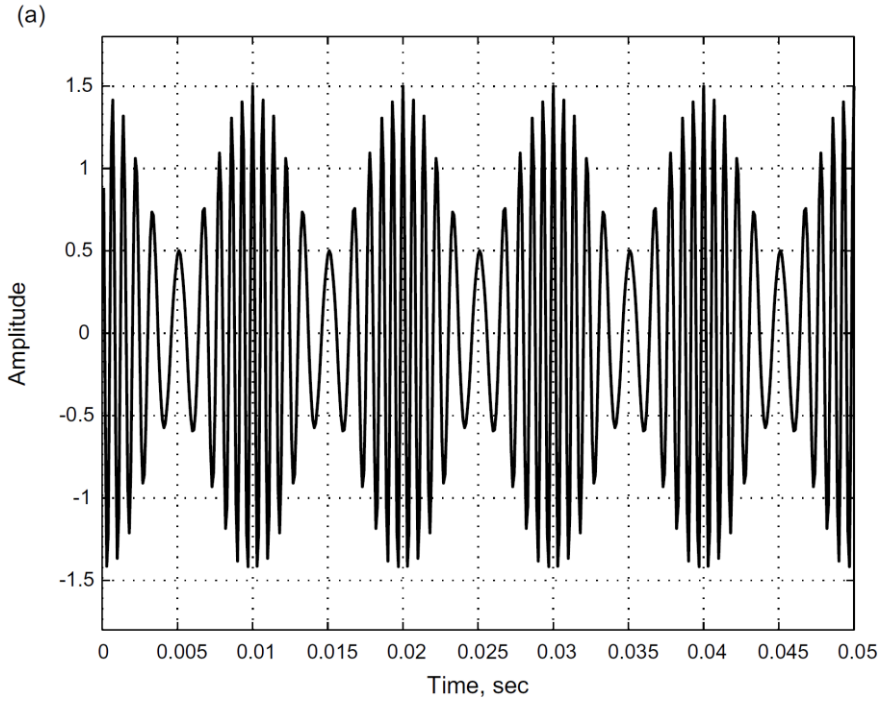
- Molto usato nell'analisi di segnali audio
- Dà informazioni su quanto rapidamente cambia lo spettro di un segnale.
- Se abbiamo un picco a una certa quefrency, lo spettro cambia con quel periodo.
- Principale vantaggio è la separazione di segnali convoluti:

$$y(t) = x(t) * h(t)$$

$$Y(f) = X(f) \cdot H(f)$$

$$y_{cc}(\theta) = x_{cc}(\theta) + h_{cc}(\theta)$$

Cepstrum


(a)
Amplitude
1.5
1
0.5
0
-0.5
-1
-1.5
0
0.005
0.01
0.015
0.02
0.025
0.03
0.035
0.04
0.045
0.05
Time, sec

Cepstrum

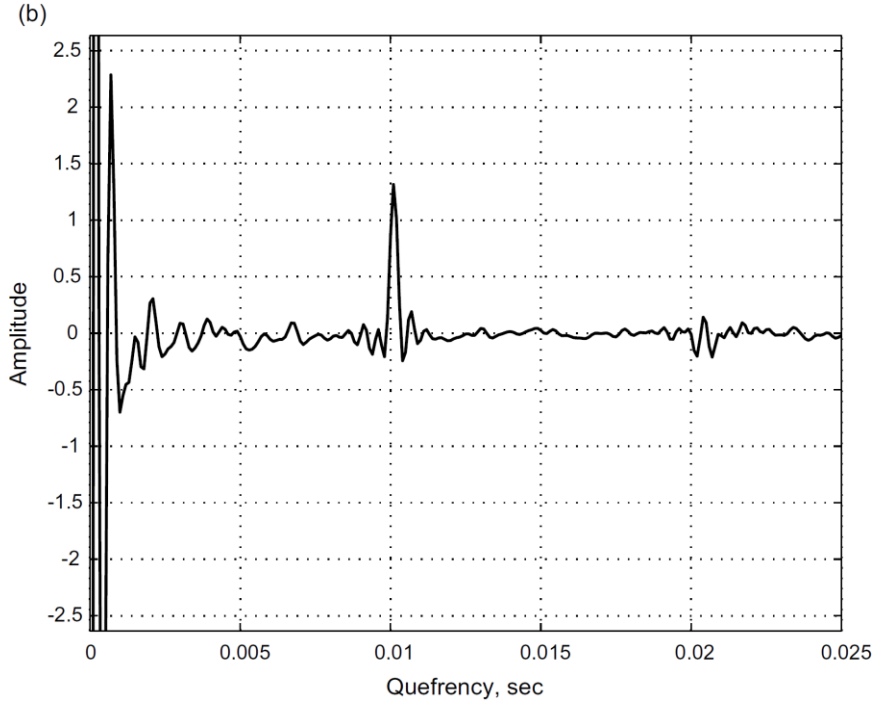


Figure 2.17 (a) FM and AM modulated signal, (b) its cepstrum, and (c) its spectrum

Generazione di toni puri

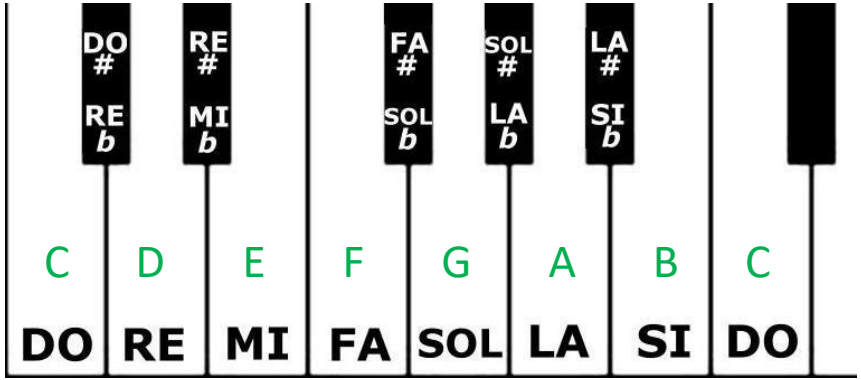
```
function [s]=tonegen(Ft, Fs, Td)
  s=sin([1:Fs*Td]*2*pi*Ft/Fs);
```

```
note=tonegen(440, 16000, 2);
soundsc(note, 16000);
```

La sopra il do centrale

Le note

- Nella scala musicale occidentale ogni ottava viene suddivisa in 12 semitoni.
- Il rapporto tra la frequenza di un semitono e quello immediatamente inferiore è $\sqrt[12]{2} = 2^{1/12}$



- Tra una nota e la stessa una ottava superiore raddoppia la frequenza.
- Il diesis porta ad un aumento di un semitono, il bemolle alla riduzione di un semitono.

Le note

- Nascono attorno all'anno mille ad opera del monaco Guido d'Arezzo.
- I nomi derivano dalle iniziali di un inno a San Giovanni,

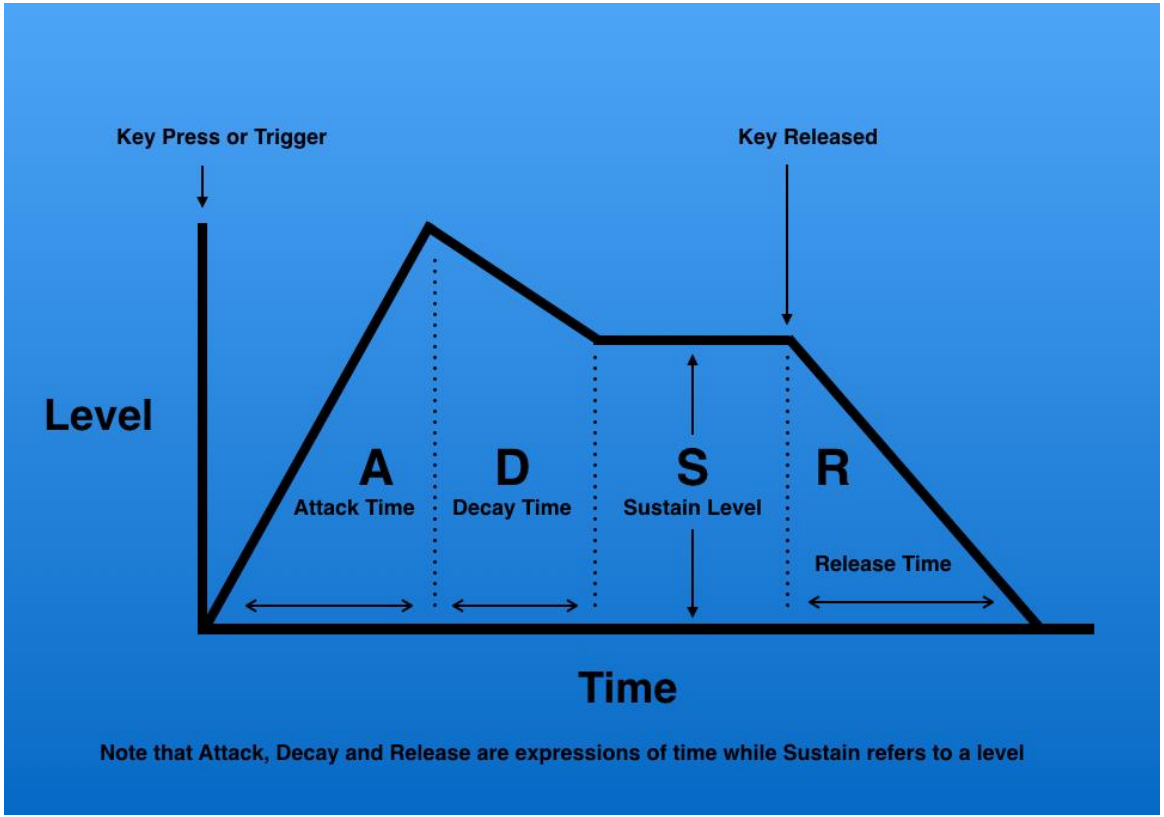
Ut queant laxis
Resonare fibris
Mira gestorum
Famuli tuorum
Solve polluti
Labii reatum
Sancte Iohannes

- L'Ut venne poi chiamato Do a partire dal '600 da Gian Battista Doni (teorico della musica), prendendola dalla prima parte del cognome.

Le frequenze delle note

C_1	32.70	C_2	65.41	C_3	130.81	C_4	261.63	C_5	523.25
$C\sharp_1$	34.65	$C\sharp_2$	69.30	$C\sharp_3$	138.59	$C\sharp_4$	277.18	$C\sharp_5$	554.37
D_1	36.71	D_2	73.42	D_3	146.83	D_4	293.66	D_5	587.33
$D\sharp_1$	38.89	$D\sharp_2$	77.78	$D\sharp_3$	155.56	$D\sharp_4$	311.13	$D\sharp_5$	622.25
E_1	41.20	E_2	82.41	E_3	164.81	E_4	329.63	E_5	659.26
F_1	43.65	F_2	87.31	F_3	174.61	F_4	349.23	F_5	698.46
$F\sharp_1$	46.25	$F\sharp_2$	92.50	$F\sharp_3$	185.00	$F\sharp_4$	369.99	$F\sharp_5$	739.99
G_1	49.00	G_2	98.00	G_3	196.00	G_4	392.00	G_5	783.99
$G\sharp_1$	51.91	$G\sharp_2$	103.83	$G\sharp_3$	207.65	$G\sharp_4$	415.30	$G\sharp_5$	830.61
A_1	55.00	A_2	110.00	A_3	220.00	A_4	440.00	A_5	880.00
$A\sharp_1$	58.27	$A\sharp_2$	116.54	$A\sharp_3$	233.08	$A\sharp_4$	466.16	$A\sharp_5$	932.33
B_1	61.74	B_2	123.47	B_3	246.94	B_4	493.88	B_5	987.77

Le frequenze delle note



From: https://theproaudiofiles.com/synthesis-101-envelope-parameters-uses/

Chirp

- Segnali la cui frequenza aumenta (o diminuisce) monotonicamente nel tempo.

```
bird=chirp([0:1/8000:1], 1000, 1, 4000);
soundsc(bird,8000)
```

Generazione di toni variabili con continuità

```
function [snd]=freqgen(frc, Fs)
  th=0;
  fr=frc*2*pi/Fs;
  for si=1:length(fr)
    th=th+fr(si);
    snd(si)=sin(th);
    th=unwrap(th);
  end
```

```
Fs=8000;        %define sample frequency
Tt=[0:1/Fs:4]; %create array of sample times
ModF=1000+200*sin(Tt*2*pi);
```

```
gensnd2=freqgen(ModF, Fs);

soundsc(gensnd2,Fs);  % listen, then view it
spectrogram(gensnd2,128,0,128,Fs,'yaxis');
```

Vedere:

- Ian Vince McLoughlin, “Speech and Audio Processing”- Cambridge Univesity Press (2016)
  - Cap. 1.3-1.4
  - Cap. 2
- Ivan Tashev “Sound Capture and Processing”, John Wiley & Sons, 2009
  - Cap. 2.5.6 (Spectrogram)