Laboratorio per l'Elaborazione MultiMediale
Lezione 3 -
Applicazioni sui segnali audio

Prof. Michele Scarpiniti

Dipartimento di Ingegneria dell'Informazione, Elettronica e Telecomunicazioni
"Sapienza" Università di Roma

http://ispac.diet.uniroma1.it/scarpiniti/index.htm
michele.scarpiniti@uniroma1.it

Introduzione

Introduzione

Introduzione

L'obiettivo di questa lezione è di illustrare alcune applicazioni utili per l'elaborazione dei segnali audio. Ad esempio, si vedrà un cancellatore adattativo di rumore (**ANC**) il cui scopo è di ridurre l'effetto di un segnale rumoroso sull'acquisizione di un segnale utile.
Un altro esempio sarà l'implementazione di un misuratore di livello sonoro (**SLM**) con filtro di ponderatura **A**. Di questa applicazione ne daremo due versioni: la prima di tipo *off-line*, cioè analizzeremo il livello sonoro del segnale contenuto in un file, la seconda di tipo *online*, cioè valuteremo il livello sonoro del segnale raccolto da un microfono.

Ancora sulla *convoluzione*

Ricordiamo che la convoluzione tra due sequenze $x[n]$ (di lunghezza N) e $h[n]$ (di lunghezza M) è definita come

$$y[n] = x[n] * h[n] = \sum_{k=0}^{N-1} x[k]h[n-k]$$

ed è lunga $L = N + M - 1$ campioni. Arrangiando i campioni della risposta impulsiva e dell'uscita in vettori colonna definiti, rispettivamente, come

$$\mathbf{h} \triangleq \begin{bmatrix} h[0] & h[1] & \cdots & h[M-1] \end{bmatrix}^T$$
$$\mathbf{y} \triangleq \begin{bmatrix} y[0] & y[1] & \cdots & y[L-1] \end{bmatrix}^T$$

l'uscita del del sistema può essere scritta come

$$\mathbf{y} = \mathbf{X} \cdot \mathbf{h}$$

Ancora sulla *convoluzione*

La matrice **X** contiene i campioni del segnale d'ingresso arrangiati in colonne e via-via *traslati* (shiftati) di un campione in basso:

$$\mathbf{X} = \begin{bmatrix} x[0] & 0 & \cdots & 0 \\ x[1] & x[0] & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ x[M-1] & x[M-2] & \cdots & x[0] \\ x[M] & x[M-1] & \cdots & x[1] \\ \vdots & \vdots & \ddots & \\ x[N-1] & x[N-2] & \cdots & x[N-M] \\ 0 & x[N-1] & \cdots & x[N-M+1] \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & x[N-1] \end{bmatrix}$$

Ancora sulla *convoluzione*

Vediamo come posso costruire la matrice **X** in Matlab. Si noti che la matrice **X** ha tutti gli elementi sulle sopra e sotto-diagonali (principale) uguali: di conseguenza è una matrice di **Toeplitz**. Si noti però, che **non** è una matrice *simmetrica*: è quindi una matrice di Toeplitz non simmetrica.
Questa matrice può essere costruita in Matlab considerando la prima colonna **c** e la prima riga **r**.

```
>> c = [x zeros(1,length(h)-1)].';
>> r = [x(1) zeros(1,length(h)-1)];
>> X = toeplitz(c,r);
>> y = Xh.';
```

Se fossi interessato a costruire una matrice di Toeplitz *simmetrica*, userei il comando:

```
>> X = toeplitz(c);
```

Ancora sulla *convoluzione*

Riportiamo un esempio:

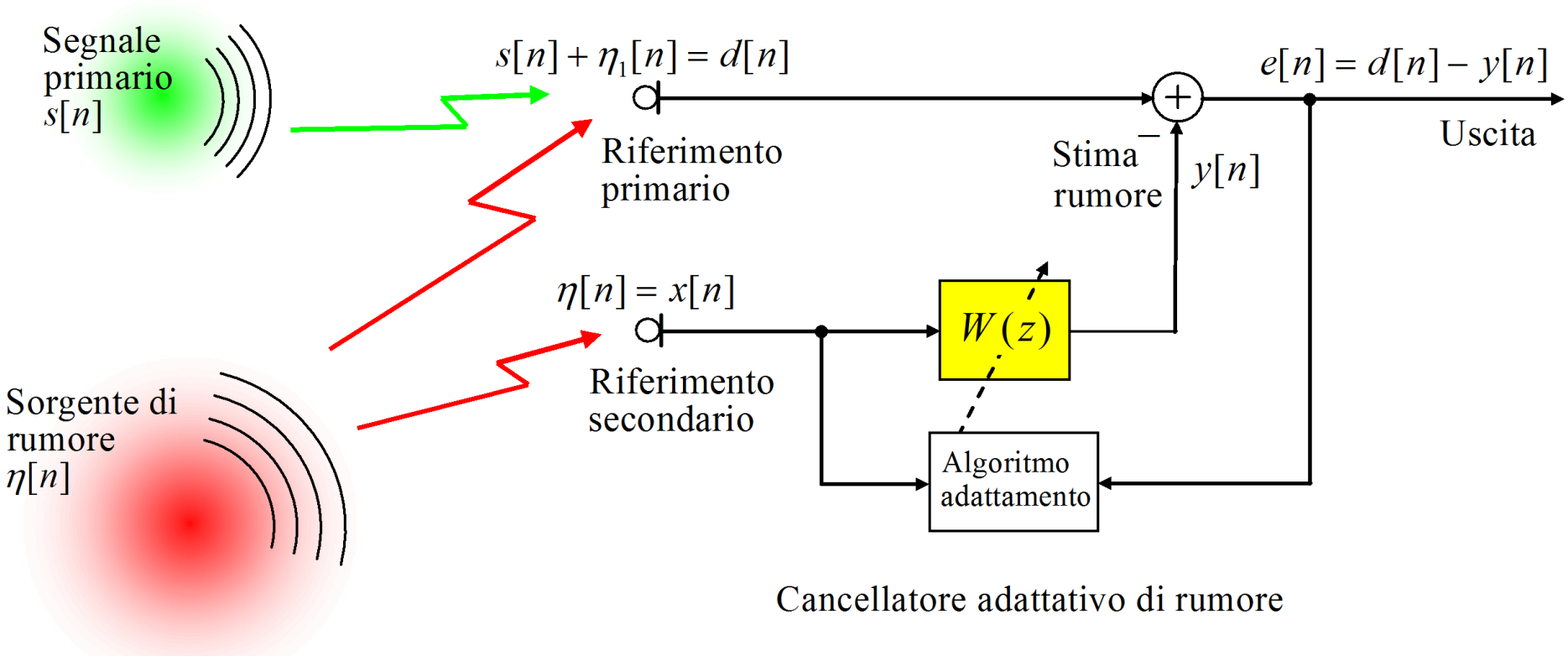
```
>> h = [1 1 1 1]/4;
>> x = randn(1,5);
>> y1 = conv(h,x)

y1 =
    -0.1081 -0.5245 -0.4932 -0.4213 -0.5998 -0.1834 -0.2147
        -0.2866

>> c = [x zeros(1,length(h)-1)].';
>> r = [x(1) zeros(1,length(h)-1)];
>> X = toeplitz(c,r);
>> y2 = X*h.'

y2 =
    -0.1081 -0.5245 -0.4932 -0.4213 -0.5998 -0.1834 -0.2147
        -0.2866
```

Il cancellatore adattativo di rumore (*ANC*)

Ci proponiamo di implementare un cancellatore adattativo di rumore o Adaptive noise canceller (**ANC**) che è un dispositivo che tenta di sottrarre a un segnale utile $s[n]$ una componente di rumore incorrelato e additivo.
Una delle principali applicazioni dell'**ANC** consiste nel miglioramento della qualità del segnale vocale o Speech Enhancement (**SE**) con la finalità di migliorare la qualità acustica percepita nei sistemi di comunicazione auditiva o per migliorare le prestazioni dei sistemi di riconoscimento automatico del parlato o Automatic Speech Recognition (**ASR**) che, infatti, degradano fortemente le loro prestazioni in presenza di rumore.
Nel caso di ambiente riverberante, uno dei problemi principali consiste nella *lunghezza* del filtro L necessaria per la modellazione del percorso acustico del rumore. Tale percorso include il ritardo tra la sorgente di rumore e la sorgente primaria e le riflessioni dovute al riverbero.

Il cancellatore adattativo di rumore (*ANC*)

Lo schema generale di un cancellatore adattativo di rumore (**ANC**) è riportato nella seguente figura



Cancellatore adattativo di rumore

Lo scopo dell'architettura riportata è di determinare un filtro $W(z)$ che possa ricostruire un *buona* stima del rumore che poi viene sottratta dal segnale di riferimento $d[n]$.

Il cancellatore adattativo di rumore (*ANC*)

L'uscita del cancellatore di rumore consiste nel segnale di errore

$$e[n] = d[n] - y[n] = s[n] + \eta_1[n] - \mathbf{w}^T\boldsymbol{\eta}$$

mentre la funzione costo $J(\mathbf{w})$ è

$$J(\mathbf{w}) = E\left\{e^2[n]\right\} = E\left\{s^2[n]\right\} + E\left\{(\eta_1[n] - \mathbf{w}^T\boldsymbol{\eta})^2\right\} + \\ +2E\left\{s[n](\eta_1[n] - \mathbf{w}^T\boldsymbol{\eta})\right\}$$

Se conoscessi la *statistica del segnale*, potrei subito ottenere (filtro di **Wiener**): $\partial J(\mathbf{w})/\partial\mathbf{w} \to 0 \Rightarrow\ -2\mathbf{R}_{\eta_1\eta} + 2\mathbf{R}_{\eta\eta}\mathbf{w} = 0$ cioè

$$\mathbf{w}_{opt} = \mathbf{R}_{\eta\eta}^{-1}\mathbf{R}_{\eta_1\eta}$$

Ovvero, in frequenza:

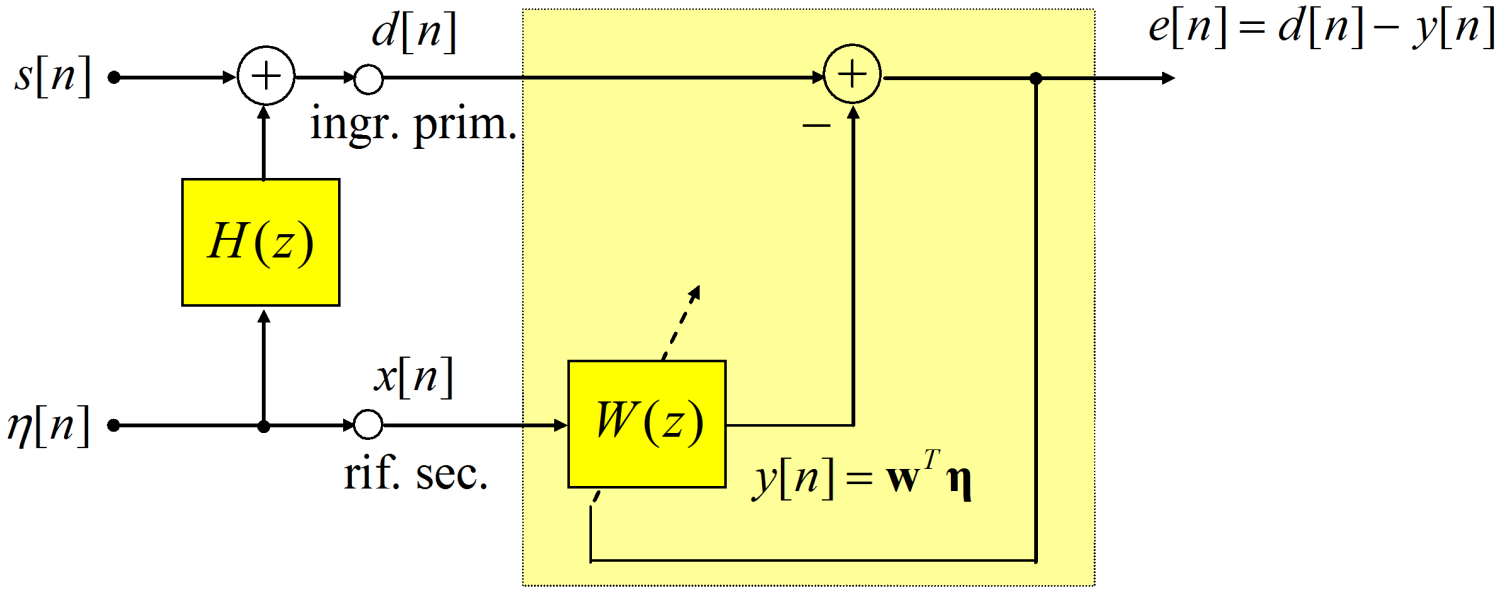
$$W_{opt}(e^{j\omega}) = \frac{R_{\eta_1\eta}(e^{j\omega})}{R_{\eta\eta}(e^{j\omega})}$$

Il cancellatore adattativo di rumore (*ANC*)

Questo risultato implica che, ricordando $R_{\eta_1\eta}(e^{j\omega}) = H(e^{j\omega})R_{\eta\eta}(e^{j\omega})$, dove il filtro $h[n]$ rappresenta la risposta impulsiva del percorso tra la sorgente primaria e quella secondaria o **acoustic phat**

$$W(e^{j\omega}) = H(e^{j\omega})$$

quindi il filtro **ottimo** dell'**ANC** consiste nella replica del filtro $h[n]$ che modella l'acoustic phat, come messo in evidenza dalla seguente figura



Il cancellatore adattativo di rumore (*ANC*)

Se **non** conosco la statistica dei segnali in gioco, posso adoperare delle stime *online*, e quindi lavorare sul singolo campione, in questo caso, ricordando che $e[n] = d[n] - y[n] = d[n] - \mathbf{w}^T\mathbf{x}$, ottengo:

$$\begin{aligned}\frac{\partial J}{\partial \mathbf{w}} &= \frac{\partial E\left\{e^2[n]\right\}}{\partial \mathbf{w}} \approx \frac{\partial e^2[n]}{\partial \mathbf{w}} = 2e[n]\frac{\partial e[n]}{\partial \mathbf{w}} = \\ &= 2e[n]\frac{\partial}{\partial \mathbf{w}}\left(d[n] - \mathbf{w}^T\mathbf{x}\right) = -2e[n]\mathbf{x}\end{aligned}$$

Questo è il classico algoritmo Least Mean Squares (o **LMS**). Usando quindi la tecnica della discesa del gradiente, il *filtro ottimo* è valutato secondo la seguente ricorsione:

$$\mathbf{w}_{k+1} = \mathbf{w}_k + \mu e[k]\mathbf{x}_k$$

Il cancellatore adattativo di rumore (*ANC*)

In Matlab potrei, quindi, caricare i segnali per prima cosa

```
>> mu = 0.0005;
>> M = 128;
>> w = zeros(1,M);
>> [s,Fs] = audioread('radio.wav');
>> [n,Fs] = audioread('noise.wav');
>> load hs
>> load hn
>> n1 = fftfilt(hn(1:128),n);
>> s1 = fftfilt(hs(1:128),s);
>> d = s1 + 0.5*n1;
```

Mi costruisco successivamente i due vettori **x** e **y**

```
>> x = [zeros(M-1,1); n];
>> y = zeros(1,length(n));
```

Il cancellatore adattativo di rumore (*ANC*)

La parte centrale dell'algoritmo è quindi scritta come:

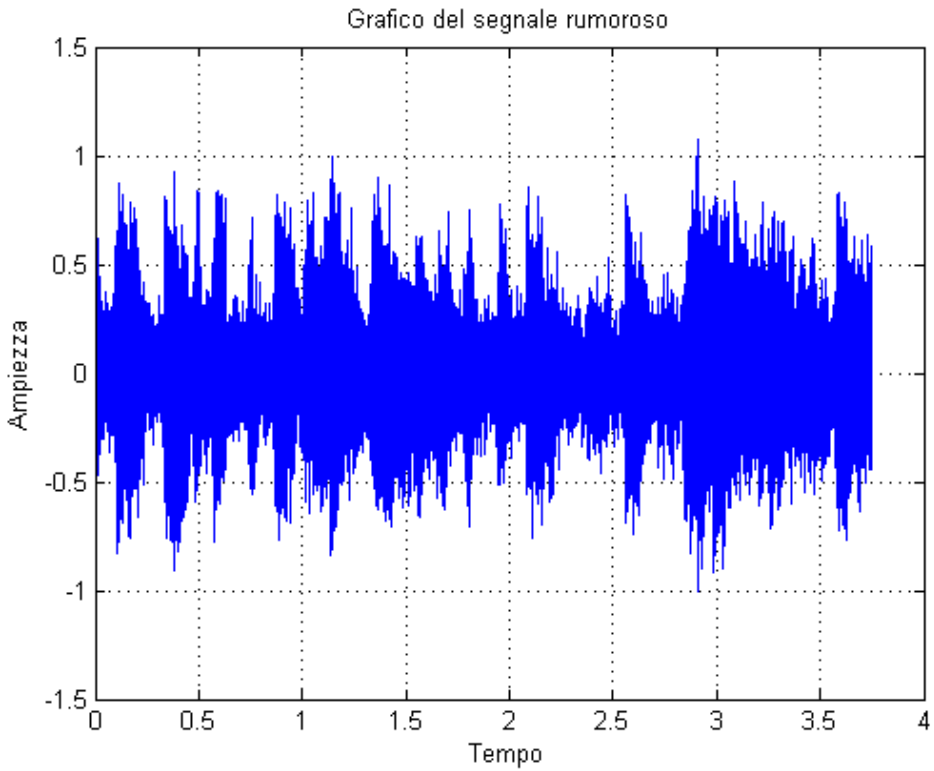
```
>> for k=1:length(n),                    % per ogni campione
    y(k) = w*x(k+M-1:-1:k);              % mi calcolo l'uscita del
        filtro
    e = d(k)-y(k);                       % mi calcolo l'errore
    w = w + mu*e*x(k+M-1:-1:k).';        % aggiorno i pesi secondo l'
        LMS
end
```

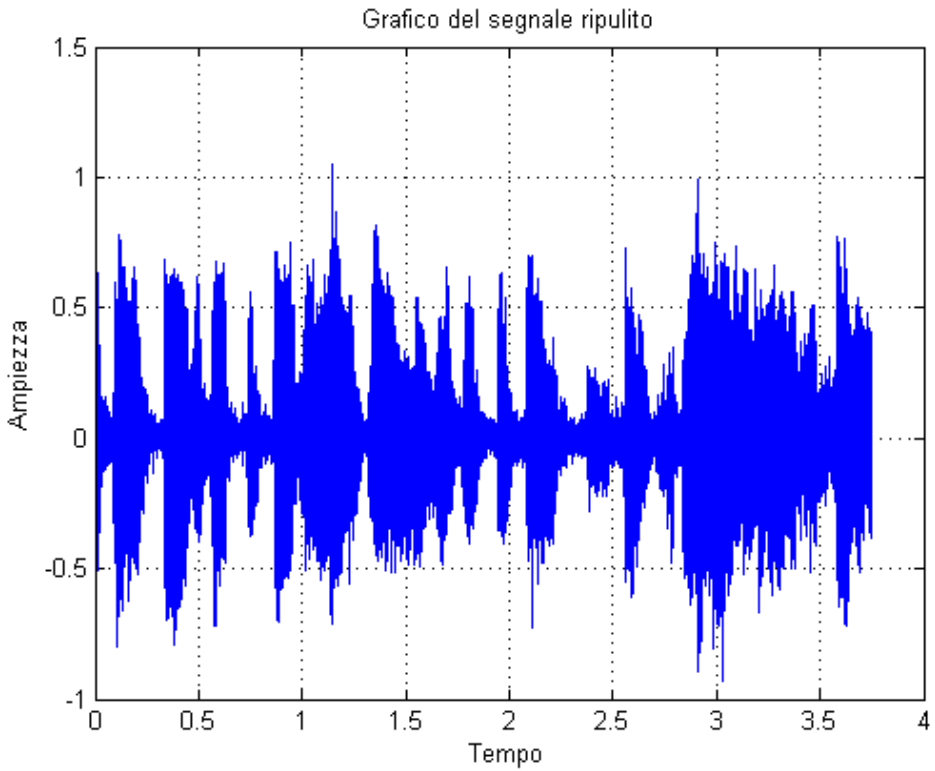
Infine mi ricostruisco il segnale in uscita al filtro ed il segnale errore:

```
>> y = fftfilt(w,n);
>> e = d - y;
>> soundsc(e,Fs);
```

Il cancellatore adattativo di rumore (*ANC*)

Riportiamo un esempio dell'effetto del cancellatore adattativo di rumore sul file `'radio.wav'` a cui è stato aggiunto del rumore:





Misuratore del Livello Sonoro

Misuratore del Livello Sonoro

Misuratore del Livello Sonoro (**SLM**)

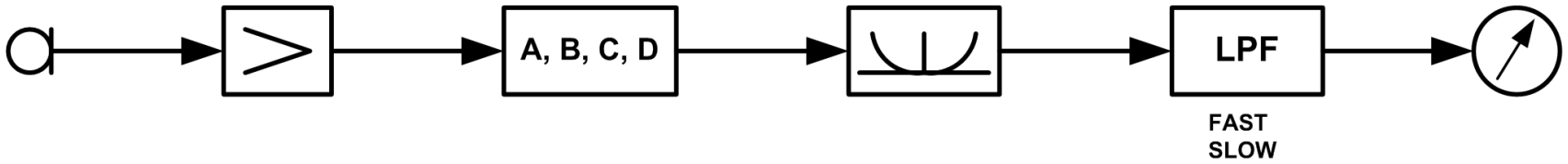
In molti contesti applicativi siamo interessati alla misura del livello sonoro: un tipico esempio applicativo è la misura del rumore ambientale in luoghi di *lavoro* al fine di valutare il rispetto della normativa vigente in materia.
Ricordiamo che come unità di livello sonoro è stata assunto il decibel (dB), secondo l'espressione:

$$L = 10 \log \frac{W}{W_0}$$

dove W rappresenta la potenza del segnale sonoro e W_0 un valore di W assunto come riferimento. Di solito nelle misure acustiche si prende come pressione di riferimento il valore $P_0 = 20\ \mu Pa$ a cui corrisponde una $W_0 = 10^{-12}\ W$.

Misuratore del Livello Sonoro (**SLM**)

Il misuratore di *livello sonoro* o **fonometro** è uno strumento in grado di effettuare una misura del livello sonoro. Esso è generalmente costituito da un microfono, uno squadratore, un filtro passa-basso ed un indicatore, secondo lo schema riportato in figura.



Oltre ai componenti citati, è presente un insieme di quattro reti filtranti denominate **A**, **B**, **C** e **D**.
L'impiego di questi filtri dovrebbe (almeno secondo le intenzioni originarie) fornire una indicazione almeno approssimata del livello di intensità **soggettiva** determinato da un suono o rumore. Comunque l'uso in tal senso è valido solo per toni puri, quindi le *scale di ponderazione* **A**, **B**, **C** e **D** vengono utilizzate in modo differente.

Misuratore del Livello Sonoro (**SLM**)

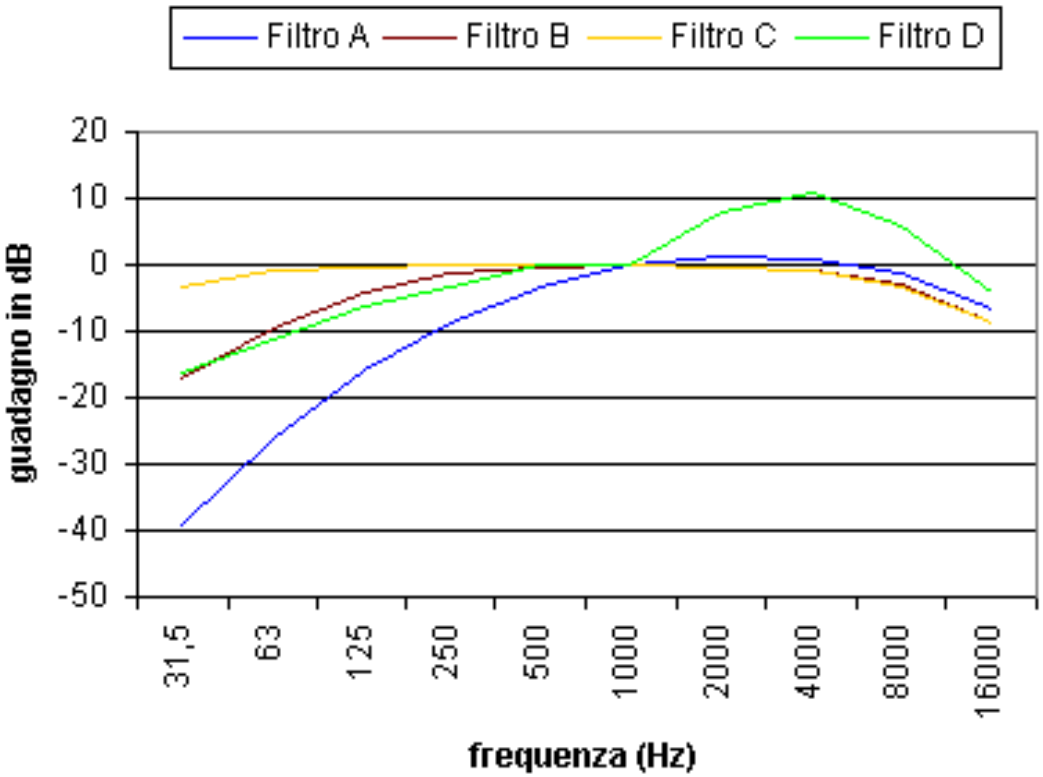
Il significato delle *scale di ponderazione* **A**, **B**, **C** e **D** è il seguente:

- La curva **A** è la più utilizzata: è adoperata quando più che l'intensità oggettiva interessa valutare gli effetti di un disturbo che un rumore può provocare sui soggetti ad esso esposto. Questa curva fornisce una classificazione abbastanza attendibile dei rumori secondo la loro crescente fastidiosità o pericolosità.
- La curva **B** è praticamente caduta in disuso.
- La curva **C** viene utilizzata nelle misure di livello sonoro per limitare l'intervallo di frequenze nel campo compreso fra 50 Hz e 10 kHz.
- La curva **D** è di uso molto specifico, per misure di rumore dovuto al traffico aereo.

Le intensità misurate con queste scale, vengono denotate con le unità dBA, dBB, dBC e dBD.

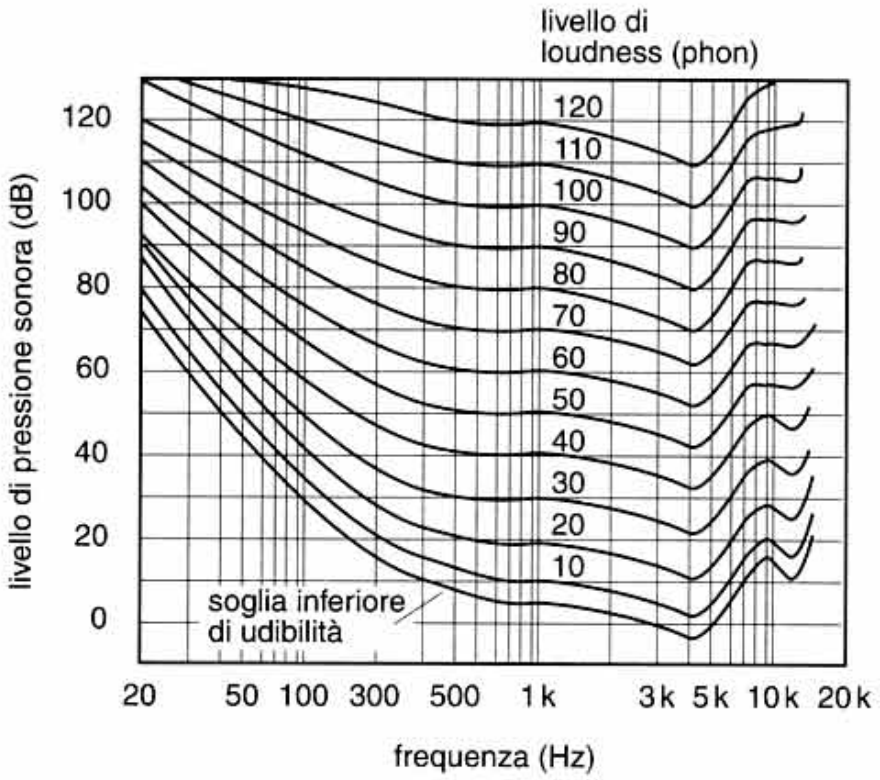
Misuratore del Livello Sonoro (**SLM**)

Le curve di ponderazione **A**, **B**, **C** e **D** sono riportate nella seguente figura



Misuratore del Livello Sonoro (**SLM**)

Si noti come la pesatura **A** tenga in conto della "*sordità*" dell'orecchio umano alle basse frequenze e della "media *sordità*" alle alte frequenze dello spettro udibile, come del resto è descritto nelle curve dell'audiogramma normale di seguito riportate.



Ciascuna curva (denominata isofonica) è caratterizzata dalla *medesima sensazione uditiva* (di livello) al variare delle frequenze e chiamato livello di loudness. Questo livello soggettivo è misurato in **phon**.

Misuratore del Livello Sonoro (**SLM**)

Il filtro passa-basso (*integratore*) in uscita dello strumento comprende due possibili costanti di tempo:

1. una rapida di valore pari a 125 ms (denominata con "**Fast**"), che permette di seguire sullo strumento indicatore le variazioni relativamente rapide del segnale acustico;
2. una lenta pari a 1000 ms (denominata "**Slow**"), da impiegare quando non sia possibile effettuare una lettura agevole per la presenza di un segnale con fluttuazioni troppo ampie, o non si voglia seguire l'andamento del segnale ma ottenere solo una misura *media*.

Gli strumenti più moderni hanno una terza costante di tempo di valore pari a 35 ms in salita (denominata "**Impulse**") e 3 s in discesa, nonché una memoria per la memorizzazione dei valori di picco raggiunti.

Implementazione di un misuratore di Livello Sonoro (**SLM**)

Vogliamo ora implementare in Matlab un algoritmo per la misura del livello sonoro con pesatura **A**. Vedremo due varianti: un algoritmo *off-line*, che consente di esaminare il livello sonoro contenuto in un file, e un algoritmo *online*, che visualizza sul monitor il livello sonoro del segnale catturato dal microfono del PC.

Come primo esempio implementiamo l'algoritmo *off-line*, dopo aver scelto i parametri, ovvero la costante di tempo dello strumento (`'slow'` o `'fast'`) e la costante per la calibrazione (`C`). Leggiamo quindi un file e la sua frequenza di campionamento:

```
>> responseType = 'fast';
>> C = 72;
>> if strcmp(responseType,'slow')
       durata = 1.0;
   else
       durata = 0.125;
   end
>> [x,Fs] = audioread('radio.wav');
```

Implementazione di un misuratore di Livello Sonoro (**SLM**)

Quindi si determina l'asse dei tempi ed il numero di punti per l'FFT (come la potenza di due più vicina al numero di campioni all'interno della finestra di analisi):

```
>> t = (1/Fs)*[0:(length(x)-1)];
>> N = ceil(durata*Fs);
>> N = 2^nextpow2(N);
```

Mi costruisco quindi la finestra di analisi, memorizzando in un vettore gli istanti iniziali di ciascuna finestra:

```
>> windowStart = [1:N:(length(x)-N)];
>> dBA = zeros(length(windowStart),1);
>> windowTime = t(windowStart+round((N-1)/2));
```

Infine per ogni finestra richiamo una funzione (`stimaLivello`) che mi calcola il livello sonoro in **dBA**:

```
>> for i = [1:length(windowStart)]
     [X,dBA(i)] = stimaLivello(x(windowStart(i)-1+[1:N]),Fs,C);
   end
```

Implementazione di un misuratore di Livello Sonoro (**SLM**)

Quindi si può graficare il segnale audio contenuto nel file e il livello sonoro in **dBA**:

```
% Grafico del segnale di ingresso.
figure(2); clf;
plot(t,x);
title('Segnale di ingresso');
xlabel('Tempo (sec.)');
ylabel('Ampiezza Normalizzata');
xlim([t(1) t(end)]);
grid on;

% Grafico del livello sonoro in dBA.
figure(3); clf;
plot(windowTime,dBA,'LineWidth',2);
title('Livello sonoro ponderato con scala A');
xlabel('Tempo (sec.)');
ylabel('Livello sonoro (dBA)');
xlim([t(1) t(end)]);
grid on;
```

Implementazione di un misuratore di Livello Sonoro (**SLM**)

Analizziamo come può essere implementata la funzione `stimaLivello` che calcola il livello sonoro con pesatura **A**. Si passerà alla funzione la *finestra di segnale*, la *frequenza di campionamento* e la *costante per la calibrazione*; essa mi restituisce lo spettro del segnale in **dB** e il livello in **dBA** del segnale nella finestra analizzata.

```
function [X,dBA] = stimaLivello(x,Fs,C)
```

Per prima cosa calcolo la trasformata del segnale utilizzando la FFT, eliminando gli eventuali valori nulli (si evita quindi log(0)) e considero solo mezzo spettro (da 0 a $F_s/2$, per la *realtà* del segnale.

```
>> X = abs(fft(x));
>> X(find(X == 0)) = 1e-17;
>> f = (Fs/length(X))*[0:(length(X)-1)];
>> ind = find(f<Fs/2);
>> f = f(ind);
>> X = X(ind);
```

Implementazione di un misuratore di Livello Sonoro (**SLM**)

Mi costruisco, per ogni frequenza f, il filtro A (tramite la funzione `filterA`) e filtro lo spettro:

```
>> A = filterA(f);
>> X = A'.*X;
```

Non resta che calcolare l'*energia media* usando la relazione di Parseval:

$$\varepsilon_x = \sum_{n=0}^{N-1} |x[n]|^2 = \frac{1}{N} \sum_{k=0}^{N-1} |X[k]|^2$$

```
>> EnergiaTotale = sum(X.^2)/length(X);
>> EnergiaMedia = EnergiaTotale/((1/Fs)*length(x));
>> dBA = 10*log10(EnergiaMedia) + C;
>> X = 20*log10(X);
```

Il livello in **dBA** è sempre riferito ad un valore di riferimento (calibrazione):

$$L_{dBA} = 10 \log \left(\frac{\varepsilon_x}{\varepsilon_{ref}} \right) = 10 \log(\varepsilon_x) + C$$

Implementazione di un misuratore di Livello Sonoro (**SLM**)

Analizziamo ora come può essere implementata la funzione `filterA` che calcola il filtro per la pesatura **A**. Al link `http://www.beis.de/Elektronik/AudioMeasure/WeightingFilters.html#A-Weighting` è disponibile per tale filtro, una espressione in forma chiusa

$$\alpha_A(f) = \frac{c_1 f^8}{(c_2 + f^2)^2 (c_3 + f^2)(c_4 + f^2)(c_5 + f^2)^2}$$

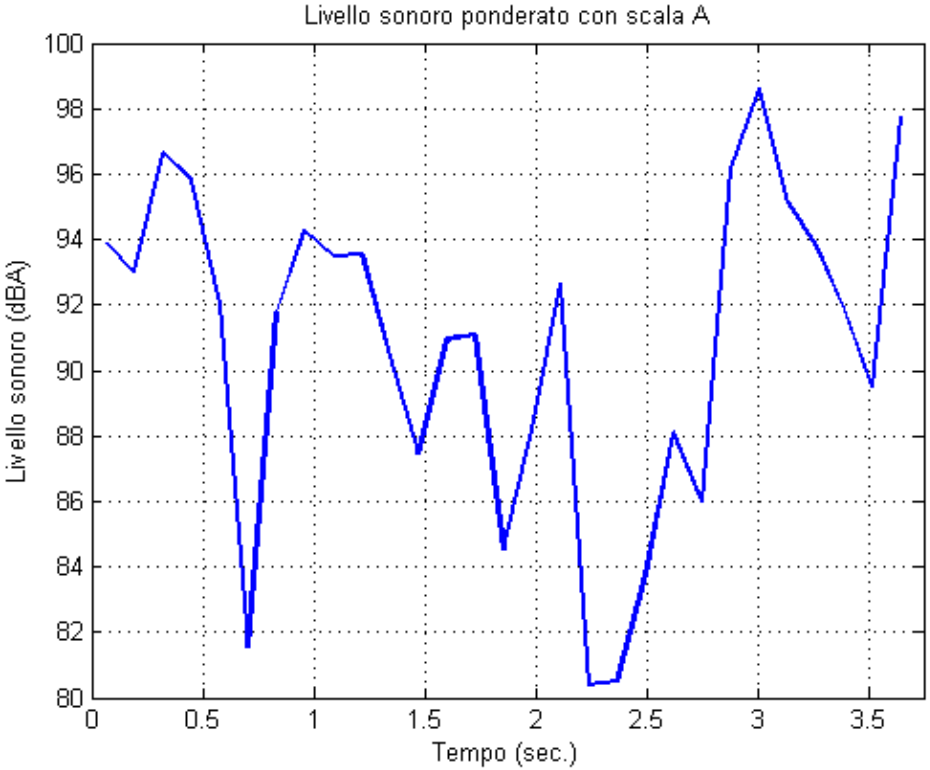
Quindi il segnale, dopo il filtraggio con pesatura **A**, può essere espresso come:

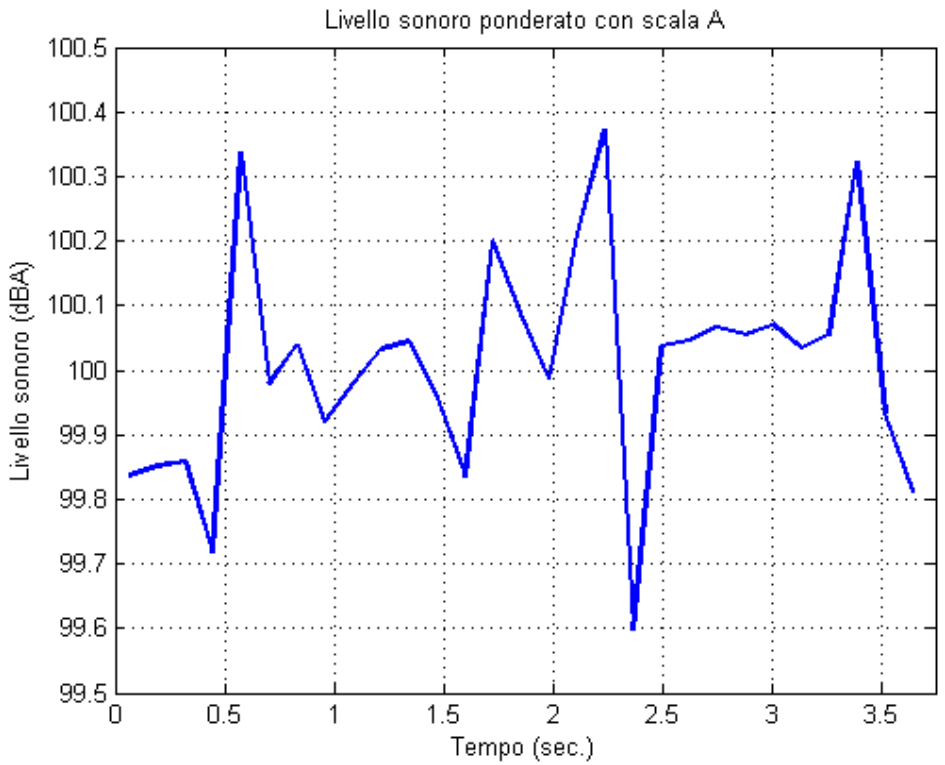
$$X_A[k] = \alpha_A(f_k) X[k] \quad \text{per} \quad f_k = k\Delta f$$

```
>> c1 = 3.5041384e16;
>> c2 = 20.598997^2;
>> c3 = 107.65265^2;
>> c4 = 737.86223^2;
>> c5 = 12194.217^2;
>> f(find(f == 0)) = 1e-17;  % Elimino gli zeri
>> f = f.^2;
>> num = c1*f.^4;
>> den = ((c2+f).^2) .* (c3+f) .* (c4+f) .* ((c5+f).^2);
>> A = num./den;
```

Implementazione di un misuratore di Livello Sonoro (**SLM**)

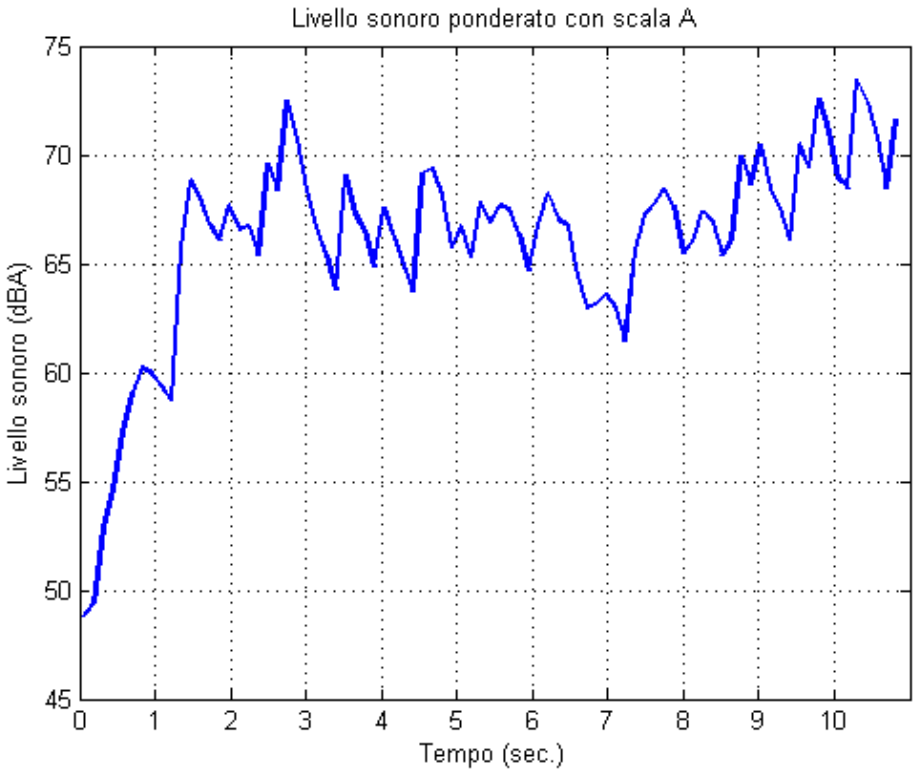
Riportiamo l'andamento del livello di energia sonora con pesatura **A** dei file `'radio.wav'` e `'noise.wav'`

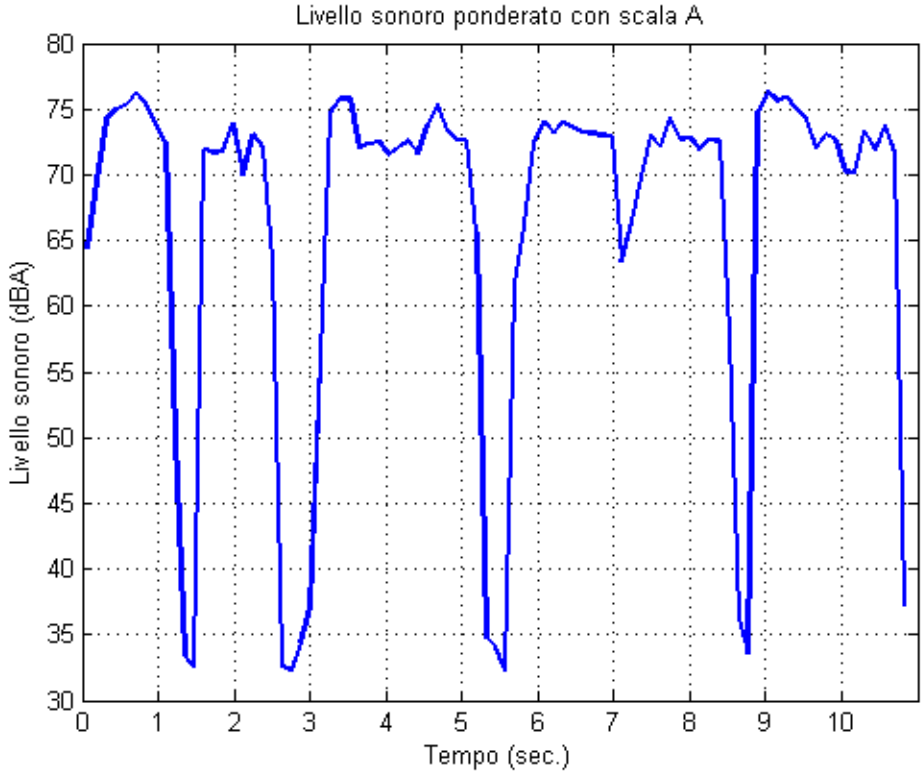




Implementazione di un misuratore di Livello Sonoro (**SLM**)

Riportiamo inoltre l'andamento del livello di energia sonora con pesatura **A** dei file `'musica1.wav'` e `'musica2.wav'`





Registrazione di un file audio

Interrompiamo la descrizione del misuratore per parlare di come sia possibile registrare un file audio da Matlab®.
In particolare Matlab mette a disposizione i seguenti comandi che creano un oggetto (il cui *handle* è `y`) di tipo `audiorecorder`:

```
>> y = audiorecorder;
>> y = audiorecorder(Fs, nbits, nchans);
>> y = audiorecorder(Fs, nbits, nchans, id);
```

Se non specificate, per default è assunto $F_s = 8000$ Hz, *nbits* $= 16$ bit e *nchans* $= 1$ (monofonico). *nchans* può essere 1 (monofonico) o 2 (stereofonico).
Il parametro `'id'` è l'identificatore della scheda audio: per default è $\mathrm{id} = -1$, cioè uso il device audio di defalut (questa opzione vale solo per sistemi *Windows*).

Registrazione di un file audio

Di seguito sono riportati alcuni dei metodi messi a disposizione per questo oggetto:

- `record(y)`: inizia la registrazione;
- `record(y,length)`: registra per un numero di secondi pari a `length`;
- `recordblocking(y,length)`: come `record`, ma restituisce il controllo solo dopo `length` secondi;
- `stop(y)`: ferma la registrazione;
- `pause(y)`: mette in pausa la registrazione;
- `resume(y)`: ricomincia la registrazione interrompendo la pausa;
- `isrecording(y)`: indica lo stato della registrazione: 1 se è in esecuzione, 0 se non è in esecuzione;
- `play(y)`: suona la registrazione;
- `getaudiodata(y)`: restituisce i campioni (in formato `double`) della registrazione;
- `getaudiodata(y,'type')`: restituisce i campioni della registrazione nel formato specificato da `'type'`;
- `display(y)`: mostra tutte le proprietà della registrazione.

Registrazione di un file audio

Di seguito sono riportati alcune delle proprietà messe a disposizione per questo oggetto:

- `Type`: nome dell'oggetto;
- `SampleRate`: frequenza di campionamento in Hz;
- `BitsPerSample`: numero di bit per campione;
- `NumberOfChannels`: numero di canali;
- `TotalSamples`: lunghezza totale (in campioni) della registrazione;
- `Running`: stato della registrazione (`'on'` o `'off'`);
- `CurrentSample`: campione corrente che viene registrato.

Per "leggere" una proprietà dell'oggetto, si usa il comando

```
>> get(y,'property')
```

Per "settare" una o più proprietà dell'oggetto, si usa il comando

```
>> set(y,'property1',value,'property2',value,...)
```

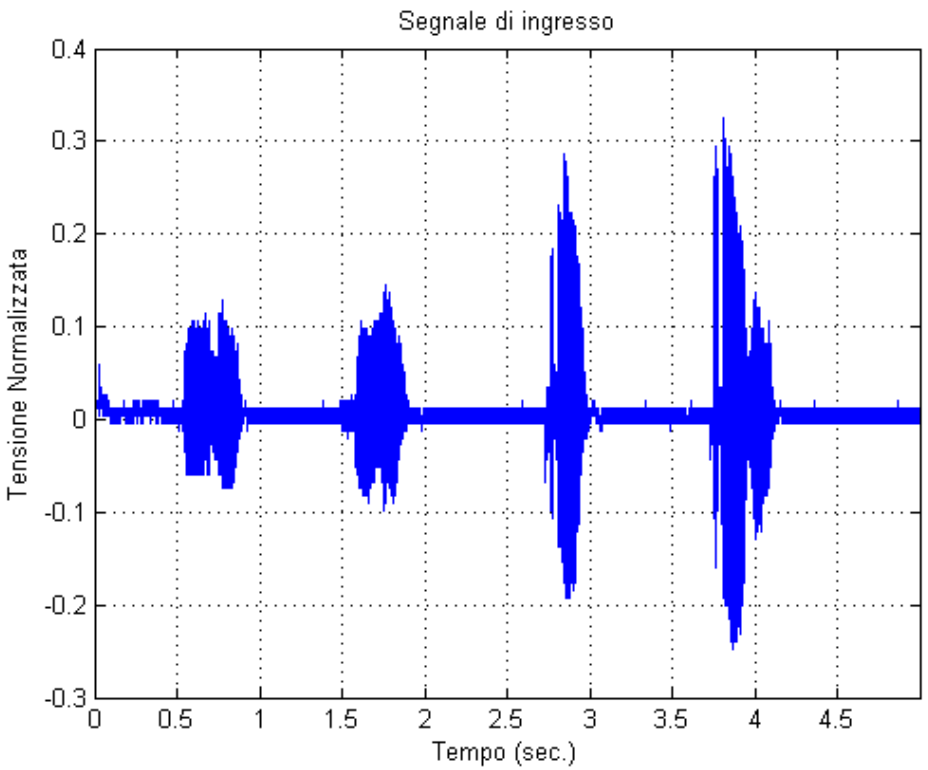
Registrazione di un file audio

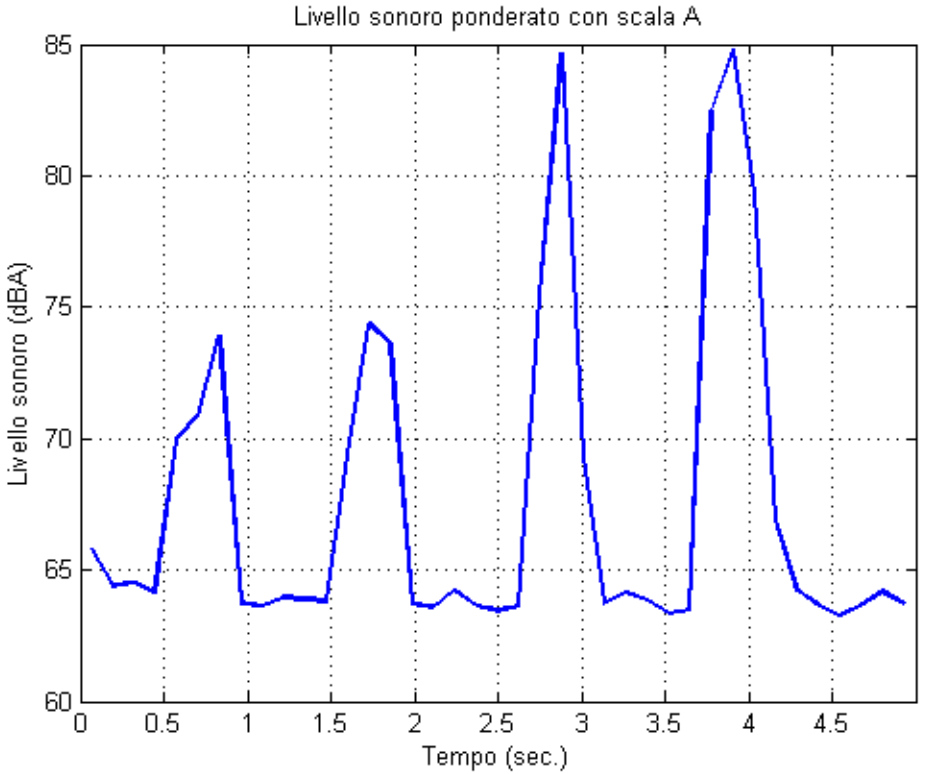
Riportiamo un esempio:

```
>> y = audiorecorder(44100, 16, 1);  % registro 1 canale,
                                     % a 44,1 kHz 16 bit
>> record(y);       % inizio la registrazione
>> pause(y);        % metto in pausa
>> play(y);         % ascolto la registrazione
>> resume(y);       % riprendo la registrazione
>> stop(y);         % fermo la registrazione
>> play(y);         % ascolto la registrazione
>> x = getaudiodata(y);   % ottengo in x i campioni
>> wavwrite(x,44100,16,'prova-rec');   % salvo il file
```

Registrazione di un file audio

Eseguiamo allora una registrazione della frase "*uno*, *due*, *tre*, *prova*" in un file mono canale, con $F_s = 8000$ Hz e 16 bit. Eseguiamo poi l'analisi del livello sonoro del file registrato, che riportiamo di seguito:





Implementazione di un misuratore di Livello Sonoro (**SLM**)

Riprendiamo il discorso sul misuratore di livello sonoro. Cercheremo ora di implementare un algoritmo che funzioni *online*. A tal proposito basta lanciare lo sript `'SlmOnLine.m'`, cioè basta scrivere sulla *Command Window*

```
>> SlmOnLine
```

Per implementare un tale tipo di algoritmo è necessario leggere i dati dalla scheda audio: bisogna quindi approfondire questo argomento.
Per prima cosa si deve *inizializzare* la scheda audio. Questa operazione verrà implementata nella funzione

```
>> [AI,Fs,N] = initSoundCard(Fs,responseType);
```

Analizziamo, pertanto, questa funzione in dettaglio.

Implementazione di un misuratore di Livello Sonoro (**SLM**)

Per creare un oggetto di input analogico della scheda audio, si usano i comandi

```
>> AI = analoginput('adaptor');
>> AI = analoginput('adaptor',ID);
```

In cui:

- `'adaptor'`: è il nome dell'adattatore per il driver hardware. E' supportato: `advantech`, `hpe1432`, `keithley`, `mcc`, `nidaq`, e `winsound` (la scheda audio standard di Windows).
- `ID`: l'identificatore del device hardware (è opzionale e di default vale 0).
- `AI`: l'oggetto per l'ingresso analogico.

Nel nostro caso:

```
>> AI = analoginput('winsound');
```

Implementazione di un misuratore di Livello Sonoro (**SLM**)

Successivamente aggiungo al device audio i canali, utilizzando i comandi

```
>> chans = addchannel(obj,hwch);
>> chans = addchannel(obj,hwch,index);
>> chans = addchannel(obj,hwch,'names');
>> chans = addchannel(obj,hwch,index,'names');
```

In cui:

- `obj`: è l'oggetto per l'ingresso analogico.
- `hwch`: specifica l'identificativo del canale audio aggiunto.
- `index`: un indice adoperato per associare il canale.
- `'names'`: nome descrittivo del canale.
- `chans`: vettore colonna dei canali di stessa lunghezza di `hwch`.

Nel nostro caso:

```
>> channel = addchannel(AI,1);
```

Implementazione di un misuratore di Livello Sonoro (**SLM**)

Imposto quindi la frequenza di campionamento. Non tutte le frequenze sono supportate: se scelgo una frequenza non supportata, Matlab si imposta a quella più vicina al valore desiderato:

```
>> set(AI,'SampleRate',Fs);
>> Fs = get(AI,'SampleRate');
```

Allo stesso modo setto il Trigger per la sincronizzazione della scheda:

```
>> set(AI,'SamplesPerTrigger',N);
>> N = get(AI,'SamplesPerTrigger');
```

Infine imposto le altre proprietà:

```
>> set(AI,'TriggerType','Manual');
>> set(AI,'TriggerRepeat',1);
>> set(AI,'TimerPeriod',durata/4);
```

L'ultima impostazione serve a richiamare la funzione di *callback* ogni *durata*/4 secondi, al fine di rendere l'acquisizione in real-time.

Implementazione di un misuratore di Livello Sonoro (**SLM**)

L'acquisizione viene fatta partire con i comandi:

```
>> start(AI);
>> trigger(AI);
```

In cui:

- `start(obj)`: fa partire il device descritto dall'oggetto `obj`.
- `trigger(obj)`: fa partire il *trigger* sull'oggetto `obj`.

Quindi ritorno nella funzione `main`, e acquisisco i primi campioni tramite:

```
>> x = getdata(AI);
```

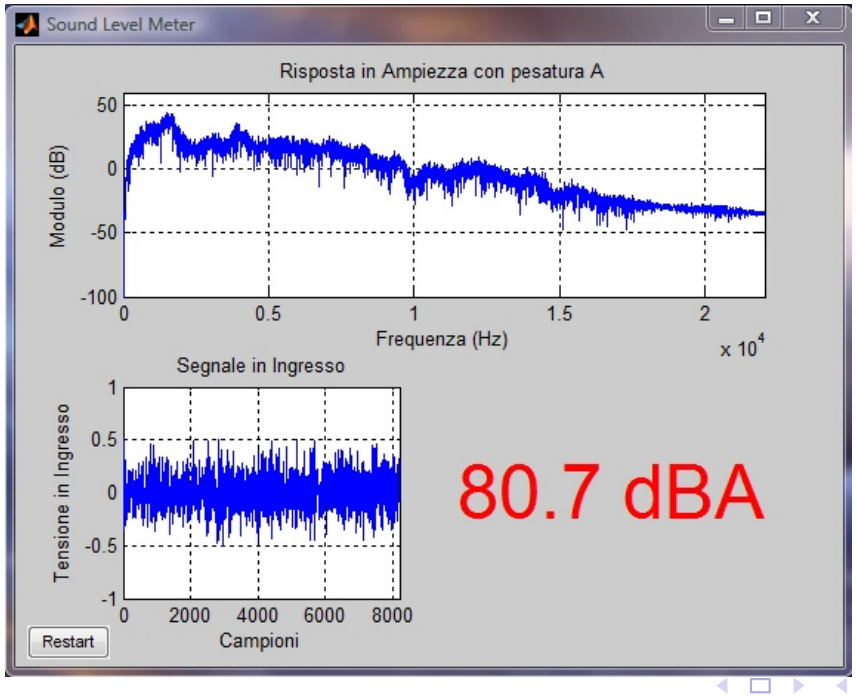
Tale funzione `getdata(obj)` estrae un numero di campioni specificato da `'SamplesPerTrigger'`. Oppure si poteva usare la funzione `getdata(obj,samples)` che restituisce un numero di campioni pari a `samples`.

Implementazione di un misuratore di Livello Sonoro (**SLM**)

A questo punto la stima del livello sonoro è fatta allo stesso modo del caso off-line, cioè vengono utilizzate le due funzioni:

```
>> [X,dBA] = stimaLivello(x,Fs,C);
>> A = filterA(f);
```

Il passo successivo è creare ed inizializzare una finestra in cui mostrare l'andamento del livello sonoro, l'andamento del segnale acquisito e il livello in dBA, come è illustrato dalla seguente figura.



Implementazione di un misuratore di Livello Sonoro (**SLM**)

Utilizzo, per creare tale finestra, la funzione

```
>> initDisplay(AI,x,X,dBA,Fs,C);
```

Essa è così costituita. Per prima cosa creo la finestra, con il proprio nome

```
>> figWindow = figure(1); clf;
>> set(gcf,'Name','Misuratore del Livello Sonoro (SLM)');
>> set(gcf,'NumberTitle','off','MenuBar','none');
```

Quindi creo l'asse delle frequenze:

```
>> f = (Fs/length(x))*[0:(length(X)-1)];
```

Implementazione di un misuratore di Livello Sonoro (**SLM**)

Disegno il modulo dello spettro del segnale

```
>> subplot(2,1,1);
>> fftPlot = plot(f,X);
>> title('Risposta in Ampiezza con pesatura A');
>> xlabel('Frequenza (Hz)');
>> ylabel('Modulo (dB)');
>> xlim([0 Fs/2]); ylim([-100 60]);
>> grid on;
```

E disegno il segnale (nel tempo)

```
>> subplot(2,2,3);
>> samplePlot = plot([1:length(x)],x);
>> title('Segnale in Ingresso');
>> xlabel('Campioni');
>> ylabel('Ampiezza in Ingresso');
>> axis([0 length(x) -1 1]);
>> grid on;
```

Implementazione di un misuratore di Livello Sonoro (**SLM**)

Nel terzo grafico scriverò il valore del livello sonoro convertito in stringa da:

```
>> dBA_str = sprintf('%5.1f%s',dBA,' dBA');
```

Disegno quindi la stima dei **dBA**

```
>> subplot(2,2,4); axis off;
>> dBA_text = text(1.0,0.5,dBA_str,...
      'FontSize',38,'HorizontalAlignment','Right','Color','r');
```

Inoltre, creo un pulsante di start/stop:

```
>> uiButton = uicontrol('Style','pushbutton',...
      'Units', 'normalized',...
      'Position',[0.0150 0.0111 0.1 0.0556],...
      'Value',1,'String','Stop',...
      'Callback',@stopSoundCard);
```

Implementazione di un misuratore di Livello Sonoro (**SLM**)

Infine memorizzo tutte le variabili utilizzate per la gestione delle figure in un campo dati 'UserData', per poterle utilizzare in altre funzioni:

```
>> figData.figureWindow = figWindow;
>> figData.uiButton     = uiButton;
>> figData.samplePlot   = samplePlot;
>> figData.fftPlot      = fftPlot;
>> figData.dBA_text     = dBA_text;
>> figData.AI           = AI;
>> figData.C            = C;
>> set(gcf,'UserData',figData);
>> set(AI,'UserData',figData);
```

Implementazione di un misuratore di Livello Sonoro (**SLM**)

Quando è stato creato il pulsante di start/stop, si è scritto come argomento della funzione del pulsante `'Callback'`, `@stopSoundCard`, questo significa che al click viene eseguita la funzione `stopSoundCard`, così implementata:

```
>> figData = get(gcbf,'UserData');
>> AI = figData.AI;
>> if isrunning(AI)
       stop(AI);
       set(figData.uiButton,'string','Restart');
   else
       delete(AI);
       SlmOnLine;
   end
```

Cioè, se l'oggetto che legge dalla scheda audio è in esecuzione, viene arrestata e sul pulsante si scrive `'Restart'`; altrimenti, se l'oggetto è già stato arrestato, viene distrutto e viene lanciato di nuovo l'algoritmo.

Implementazione di un misuratore di Livello Sonoro (**SLM**)

Infine l'acquisizione dati diviene un flusso real-time attraverso l'utilizzo del comando:

```
>> set(AI,'TimerFcn',@updateDisplay);
```

questo significa che trascorso il tempo `'TimerFcn'` viene eseguito la callback della funzione `updateDisplay`. Quindi leggo tutti i dati riguardanti la figura che avevo salvato:

```
>> figData = obj.userData;
```

Successivamente leggo i dati correnti dalla scheda audio:

```
>> x = peekdata(obj,obj.SamplesPerTrigger);
```

La funzione `peekdata(obj,samples)` legge dall'oggetto `obj` del device audio un numero di campioni pari a `samples`.

Implementazione di un misuratore di Livello Sonoro (**SLM**)

Quindi viene effettuata nuovamente la stima del livello sonoro

```
>> [X,dBA] = stimaLivello(x,obj.SampleRate,figData.C);
```

e infine riaggiornata la figura

```
>> dBA_str = sprintf('%5.1f%s',dBA,' dBA');
>> set(figData.samplePlot,'YData',x);
>> set(figData.fftPlot,'YData',X);
>> set(get(figData.fftPlot,'Parent'),'YLim',[-100 60])
>> set(figData.dBA_text,'string',dBA_str);
```

e così via fino a che non si arresta definitivamente l'applicazione.

Bibliografia

Matlab

MATLAB: Getting Started Guide.
Available on-line: http://www.mathworks.com/access/helpdesk/help/pdf_doc/matlab/getstart.pdf

Matlab

Signal Processing Toolbox 6: User's Guide.
Available on-line: http://www.mathworks.com/access/helpdesk/help/pdf_doc/signal/signal_tb.pdf

A.V. Oppenheim, R.W. Schafer.

Discrete-Time Signal Processing
2-nd Edition, Prentice Hall, 1999.

T.A. Davis.

MATLAB Primer.
8-th Edition, CRC Press, 2010..

D.M. Smith.

Engineering Computation with MATLAB.
2-nd Edition, Addison-Wesley, 2010.

A. Knight.

Basics of MATLAB and Beyond.
CRC Press, 1999.

A.D. Poularikas.

Signals and Systems Primer with MATLAB.
CRC Press, 2006.