

Rimozione di rumore sinusoidale da un segnale audio

Progetto per il corso Elaborazione Numerica dei Segnali, 2014 - 2015

Michele Polese, 1100877

20 gennaio 2015

1 Introduzione

In questa relazione viene descritta una procedura per identificare e filtrare rumore sinusoidale da un segnale audio - in particolare modo dal parlato - e l'implementazione della stessa in MATLAB®. L'obiettivo è di rimuovere con precisione le componenti sinusoidali e operare in real-time, senza degradare la qualità della restante parte del segnale. Nella prima sezione è introdotto il metodo iterativo utilizzato per identificare sinusoidi utilizzando la *Discrete Fourier Transform* (DFT), segue poi la descrizione dei filtri notch impiegati per rimuovere il rumore. Nella terza sezione viene riportata l'implementazione del sistema real-time in MATLAB®, con alcune considerazioni sui risultati raggiunti.

2 Rilevazione delle componenti sinusoidali

Il file audio da elaborare contiene un segnale audio a frequenza di campionamento $F_s = 16$ KHz del tipo $x(nT) = s(nT) + \sum_{i=1}^n z_i(nT)$, dove $s(nT)$ è il segnale utile e $z_i(nT) = A_0 \cos(2\pi f_i nT)(u(nT) - u((n - n_0)T))$ un rumore sinusoidale finestrato. Per rendere la procedura più generale possibile non vengono fatte assunzioni preliminari sull'intervallo di frequenze del rumore, sul numero di seni presenti e sul loro istante d'attacco. Per rilevare le componenti sinusoidali in un segnale si effettua un'analisi frequenziale del segnale. Calcolando la trasformata di Fourier su un segnale sinusoidale discreto di tipo $n_i(nT) = A_0 \cos(2\pi f_i nT)$ si ottiene

$$N_i(e^{j\theta}) = \frac{A_0}{2T}(\delta(\theta - \theta_i) + \delta(2\pi - \theta_i - \theta)) \quad (1)$$

con $\theta_i = 2\pi f_i T$.

Tuttavia per via della finestratura $z_i(nT) = n(nT)(u(nT) - u((n - n_0)T))$ che viene effettuata sul segnale (sia perché le componenti in analisi hanno un inviluppo e una durata limitati, sia perché viene considerata solo una porzione del segnale per volta) si ha una convoluzione in frequenza. In particolare modo, data $w_r(nT) = u(nT) - u((n - n_0)T)$ e la sua trasformata $W(e^{j\theta}) = \text{sinc}(\theta)$ si ottiene che l'effettiva rappresentazione del segnale sinusoidale nel dominio della trasformata di Fourier è:

$$Z_i(e^{j\theta}) = W(e^{j\theta}) * N_i(e^{j\theta}) = \frac{A_0}{2T}(W(\theta - \theta_i) + W(2\pi - \theta_i - \theta)) \quad (2)$$

Pertanto dato il segnale $x(nT) = s(nT) + \sum_{i=1}^n z_i(nT)$, somma del parlato $s(nT)$ con trasformata di Fourier $S(e^{j\theta})$ e di n componenti sinusoidali finestrate $z_i(nT)$ a frequenza f_i , la trasformata di Fourier che si ottiene è:

$$X(e^{j\theta}) = S(nT) + \sum_{i=1}^n Z_i(e^{j\theta}) \quad (3)$$

In un calcolatore si utilizza la DFT del segnale, ovvero sia la versione campionata della precedente trasformata.

Tutto questo può essere evidenziato nella Figura 1, che riporta la DFT calcolata con MATLAB® di 250 ms di segnale audio con il parlato e 3 componenti sinusoidali di frequenza $f_1 = 1300$ Hz, $f_2 = 2600$ Hz, $f_3 = 3800$ Hz.

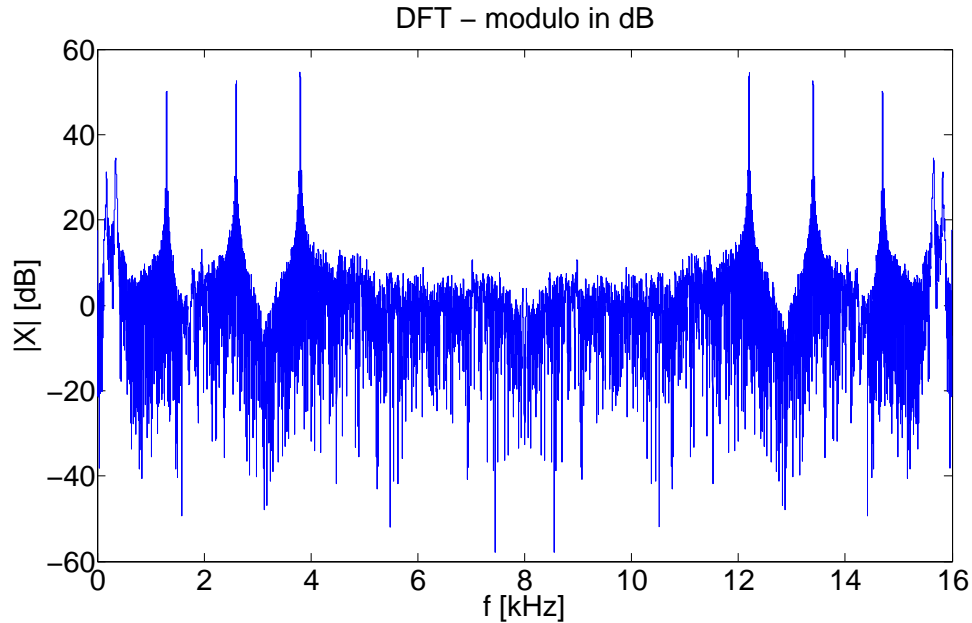


Figura 1: DFT di 250 ms di segnale con rumore a frequenza $f_1 = 1300$ Hz, $f_2 = 2600$ Hz, $f_3 = 3800$ Hz

La prominenza della rappresentazione in frequenza di un seno sulla parte utile del segnale è ciò che consente di identificare le componenti sinusoidali in modo automatico. In particolar modo, data la DFT del segnale, si possono cercare i massimi per le frequenze da 0 a $F_s/2$. Le frequenze corrispondenti saranno quelle dei seni da rimuovere. Tuttavia, nel momento in cui si opera in real-time e non sull'intero segnale già registrato non si può sapere se nell'intervallo che si sta analizzando sia presente un rumore sinusoidale o meno. La strategia adottata pertanto deve tenere conto di questo fattore. Di seguito si propone lo pseudocodice 1.

In questo modo si riescono a identificare tutti i picchi più alti della soglia R , senza assunzioni preliminari sul numero di componenti sinusoidali presenti in un dato istante.

Algorithm 1 Procedura per rilevare rumore sinusoidale

```
1: procedure
2:   dato  $x(nT)$ , segnale d'ingresso con
3:    $F_s$  frequenza di campionamento
4:   applica una finestra triangolare, ottenendo  $y(nT)$ 
5:   definisce una soglia  $R$ 
6:   calcola  $X(e^{j\theta}) = DFT(x(nT))$ 
7:   calcola  $Y(e^{j\theta}) = DFT(y(nT))$ 
8:   while  $\max_{\theta \in [0, \pi]} (Y(e^{j\theta})) > R$  do
9:      $f_j = \frac{F_s}{2\pi} \arg \max_{\theta \in [0, \pi]} (Y(e^{j\theta}))$ 
10:    filtra la componente sinusoidale a frequenza  $f_j$  in  $X(e^{j\theta})$  e  $Y(e^{j\theta})$ 
11:     $out(nT) = IDFT(X(e^{j\theta}))$ 
```

L'approccio iterativo della procedura è ispirato all'articolo [1], in cui si propone un metodo per isolare componenti sinusoidali da un rumore bianco. In questa versione tuttavia lo scopo è eliminare le componenti sinusoidali rumorose dal parlato, che non ha una caratterizzazione statistica come il rumore bianco, pertanto non è stata ricavata una formulazione analitica per la soglia R . Questa è infatti costante e viene impostata applicando l'algoritmo su diversi campioni audio (solo parlato, parlato e componenti sinusoidali) e identificando quale valore consente di discriminare le sinusoidi del rumore dalle fondamentali della voce nella maggiorparte dei casi. Queste ultime sono infatti sinusoidi, se osservate in un intervallo sufficientemente stretto, ma di ampiezza inferiore rispetto a quella del rumore da eliminare, che tende a saturare e quindi raggiungere il massimo valore rappresentabile.

L'utilizzo di una finestra triangolare consente di abbassare i picchi laterali dei seni, in questo modo lo *spectral leakage* diminuisce e aumenta la differenza in ampiezza tra i picchi che rappresentano un seno e quelli che rappresentano le fondamentali della voce. Si possono quindi usare soglie R più basse, identificando il rumore sinusoidale anche quando si sta sviluppando e non è in saturazione. Inoltre l'utilizzo della finestra triangolare consente di distinguere eventuali seni con frequenze vicine.

3 Filtri Notch IIR del second'ordine

Per rimuovere le componenti sinusoidali dal segnale utile sono stati utilizzati dei filtri notch IIR del second'ordine. Ciascun filtro ha risposta in frequenza

$$H_j(e^{j\theta}) = b_0 \frac{1 - 2 \cos \theta_0 e^{-j\theta} + e^{-j2\theta}}{1 - 2r \cos \theta_0 e^{-j\theta} + r^2 e^{-j2\theta}} \quad (4)$$

dove $\theta_0 = 2\pi f_j / F_s$, $r = (1 - \Delta\theta_{3dB}/2)$, con f_j frequenza da filtrare e F_s frequenza di campionamento, $b_0 = 1$

Nel campione da analizzare le frequenze f_j a cui si sviluppano le componenti sinusoidali sono riportate in Figura 1. Modulo e fase dei singoli filtri applicati su ciascuna componente si possono vedere nelle Figure 2, 3, 4, mentre modulo e fase della cascata dei 3 filtri sono nelle Figure 5 e 6. Per ciascuno $\Delta\theta_{3dB} = 200$ Hz, $F_s = 16$ KHz.

La fase di questi filtri non è lineare, ma non è un requisito necessario per il filtraggio di

segnali audio.

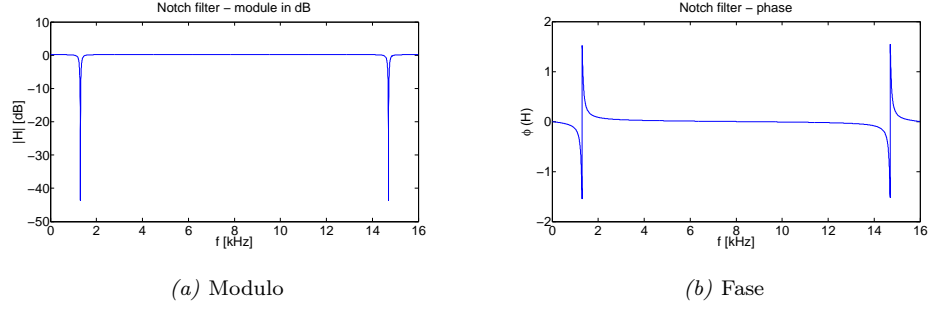


Figura 2: Filtro notch per $f = 1300$ Hz

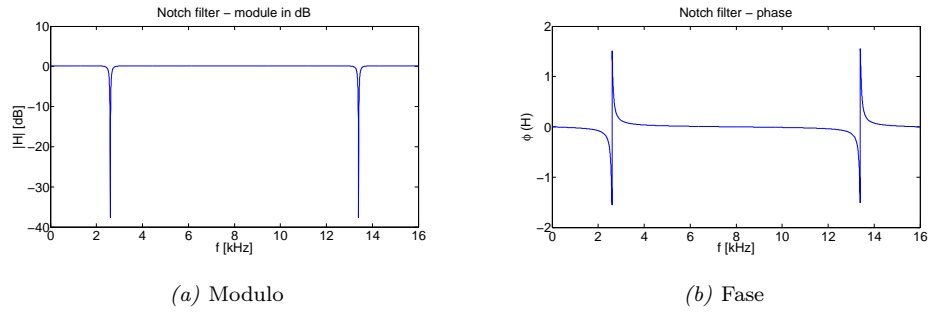


Figura 3: Filtro notch per $f = 2600$ Hz

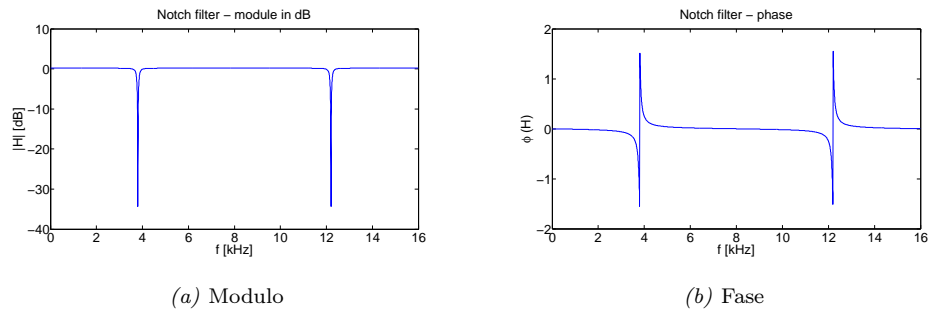


Figura 4: Filtro notch per $f = 3800$ Hz

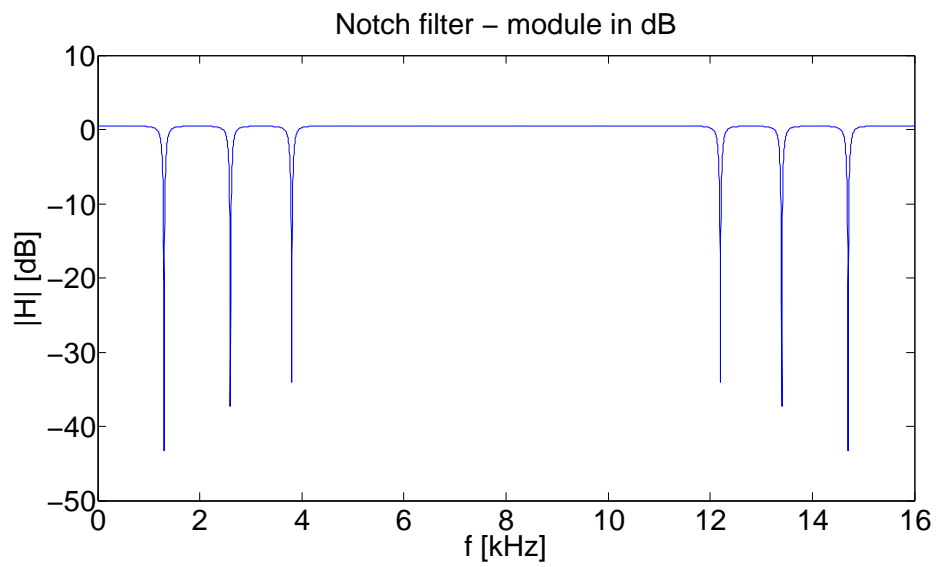


Figura 5: Modulo della cascata dei 3 filtri notch

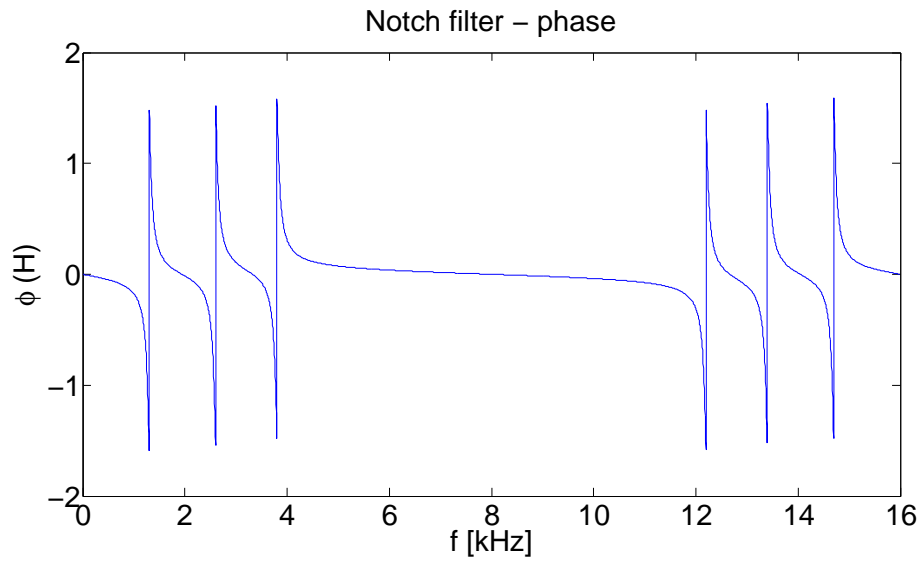


Figura 6: Fase della cascata dei 3 filtri notch

4 Implementazione in MATLAB®

Il campione audio da analizzare presenta rumore sinusoidale con una componente da 3 s a 5 s a frequenza 1300 Hz, la precedente e un seno di frequenza 2600 Hz da 5 s a 6 s e infine si aggiunge una sinusoide a frequenza 3800 Hz fino alla fine del segnale (10 s). In Figura 7 si può trovare l'andamento del numero di seni nel tempo.

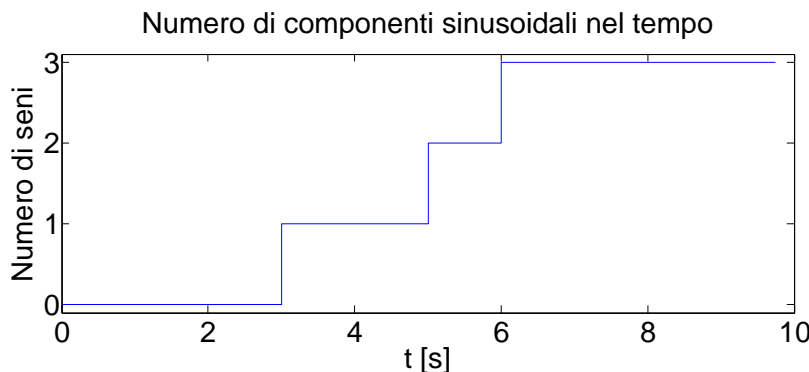


Figura 7: Numero di componenti sinusoidali nel tempo

L'implementazione dell'algoritmo 1 è stata realizzata in real-time grazie all'utilizzo del package `dsp` di MATLAB®. L'approccio seguito è quello di filtrare n samples del segnale prima e riprodurre gli stessi in seguito, mentre si filtrano gli n successivi. Quindi in realtà è presente una latenza che è pari a n/F_s , con F_s frequenza di campionamento. La DFT è calcolata usando la funzione `fft`.

Di seguito il codice utilizzato, con `systobjdemo` che ha il compito di acquisire il segnale, filtrarlo e riprodurlo n samples per volta. Il rilevamento dei rumori sinusoidali e l'effettivo filtraggio è svolto dal System Object `deleteSins`.

Durante lo sviluppo del codice sono state effettuate alcune osservazioni. Innanzitutto c'è un trade-off tra la qualità del segnale filtrato, le risorse computazionali richieste e la latenza ammessa. Con la procedura 1 si riescono ad isolare e filtrare le componenti sinusoidali, senza filtrare anche fondamentali della voce, con una latenza minima di 125 ms, ovvero sia è necessario utilizzare almeno 2000 campioni del segnale. Al di sotto di questa soglia il sistema funziona ma riconosce come seno anche qualche vocale, in particolar modo le "i".

Inoltre per identificare con precisione la frequenza delle sinusoidi è necessario introdurre uno *zero padding* nella FFT, ovvero sia aumentare la risoluzione frequenziale a cui questa è calcolata. Identificare correttamente la frequenza del seno è fondamentale per poter applicare correttamente un filtro a banda stretta, che quindi non penalizzi il parlato, ma che consenta di rimuovere completamente il rumore. Errori nell'ordine di qualche Hertz lasciano un rumore di fondo distinguibile.

```

% Questo codice acquisisce, elabora riproduce un file .wav in tempo reale.

% Inizializzazione
clear all;
5 clc;
close all;

fileName = 'segnale_107.wav';
saveTo = 'Polese_Michele.wav';
10 % Numero di campioni che vengono acquisiti
FrameSize = 2000;
% Inizializza l'oggetto che importa i campioni del file .wav
AR = dsp.AudioFileReader('Filename', fileName, 'SamplesPerFrame', FrameSize);
Fs = AR.SampleRate; % frequenza di campionamento

15 % Inizializza l'oggetto che riproduce i campioni
AP = dsp.AudioPlayer('SampleRate', Fs);

% Inizializza l'oggetto che riconosce le componenti sinusoidali e filtra il
segnale
20 % I parametri sono commentati a parte nella relazione
iterativeSinsDel = deleteSins('Threshold', 120, 'Fs', Fs, 'SamplesPerFrame',
FrameSize, 'notch_band', 200);

% Inizializza l'oggetto che registra il file audio su file
toFile = dsp.AudioFileWriter('Filename', saveTo, 'SampleRate', Fs);
25 % Vettore che registra il numero di componenti sinusoidali presenti
% contemporaneamente
sinStartTimes = ones(1, Fs/FrameSize); % Fs/FrameSize il numero massimo di
iterazioni
% indice
30 count = 1;

while ~isDone(AR)
    % acquisizione di FrameSize campioni del segnale
    audioIn = step(AR);
    35 % filtraggio
    [audioOut, nSins] = step(iterativeSinsDel, audioIn);
    sinStartTimes(count) = nSins;
    % salva su file
    step(toFile, audioOut);
    40 % riproduzione
    step(AP, audioOut);

    count = count + 1;
end
45 % rilascia le risorse
release(AR);
release(AP);
release(toFile);
50 release(iterativeSinsDel);

```

```

classdef deleteSins < matlab.System & matlab.system.mixin.Propagates ...
    & matlab.system.mixin.CustomIcon
    % deleteSins Questo System Object usato per rilevare componenti
    % sinusoidali
    % in un segnale, con una procedura iterativa.
5
    properties (PositiveInteger, Nontunable)
        %Threshold
        % Specifica una soglia: se max(segnale) > threshold c' un seno
        Threshold = 200
10
        %SampleRate
        % Frequenza di campionamento
        Fs = 16000

        %SamplesPerFrame
        % Campioni analizzati ad ogni iterazione
        SamplesPerFrame = 4000

        %Notch band
        % Banda del filtro ai 3 dB
        notch_band = 200;
20
    end

    properties (Access = private)
25        % Alcune variabili che saranno inizializzate in seguito
        NFFT = 0;
        f;
        window;
        numbSins;
        r;
30
    end

    properties (DiscreteState)
        % Stato dell'oggetto, sar inizializzato in seguito
35        State;
    end

    methods
        % Costruttore
40        function obj = deleteSins(varargin)
            setProperties(obj, nargin, varargin{:});
        end
    end

    methods (Access = protected)
45        %% Funzioni comuni
        function setupImpl(obj, ~)
            % Azioni che vengono compiute una sola volta, quando l'oggetto
            % viene inizializzato
50            obj.State = zeros(obj.SamplesPerFrame, 1);
            obj.NFFT = 16*2^nextpow2(obj.SamplesPerFrame); % zero padding
            obj.f = obj.Fs*linspace(0, 1, obj.NFFT); % vettore delle
                frequenze
            obj.window = triang(obj.SamplesPerFrame); % finestra triangolare
            % calcola i parametri del notch che non dipendono dalla
            % frequenza da eliminare
55            delta = pi*obj.notch_band/(2*obj.Fs);
            obj.r=1-delta;
        end
    end

```



```

60     function [out, nSins] = stepImpl(obj,x)
        % Funzione che viene eseguita ad ogni iterazione. Riconosce le
        % componenti sinusoidali e le filtra.

        y = obj.window.*x; % applica la finestra triangolare
65     % calcola la FFT del segnale originale e di quello finestrato
        X = fft(x, obj.NFFT);
        Y = fft(y, obj.NFFT);

        % Calcola il massimo dello spettro del segnale
70     [R, index] = max(abs(Y(1:length(Y)/2)));
        obj.numbSins = 0;
        while(R > obj.Threshold) % se supera la soglia data
            % trova la frequenza corrispondente
            fm = obj.f(index);
75     obj.numbSins = obj.numbSins + 1;

            % calcola i parametri del filtro notch
            b1 = -2*cos(2*pi*fm/obj.Fs);
            b = [1 b1 1];
80     a2=obj.r^2; a1 = obj.r*b1;
            a= [1 a1 a2];
            % e il filtro notch
            [N1, ~]=freqz(b, a, obj.NFFT, 'whole', obj.Fs);

85     % filtra X, Y
            X = N1 .* X;
            Y = N1 .* Y;

            % ricalcola R
90     [R, index] = max(abs(Y(1:length(Y)/2)));
        end

        % ottieni il segnale filtrato con ifft
        out_compl = ifft(X);
95     % e aggiungi il residuo dell'iterazione precedente (dovuto a
        % zero packing e finestratura)
        out = out_compl(1:length(x)) + obj.State;
        % salva l'attuale residuo nel vettore di stato
        obj.State = out_compl(length(x) + 1:2*length(x));
100    % e ritorna il numero di seni trovati
        nSins = obj.numbSins;
    end

    function resetImpl(obj)
105    % Resetta lo stato
        obj.State = zeros(obj.SamplesPerFrame, 1);
    end

    % vengono di seguito omesse funzioni ereditate dal generico
110    % System Object di MATLAB che non sono state modificate
end
end

```

5 Risultati

Sono state effettuate numerose prove per tarare al meglio i parametri del System Object `deleteSins`. Alla fine si è raggiunto un compromesso tra risorse computazionali allocate, latenza e qualità del segnale filtrato che consente di rimuovere il rumore sinusoidale pur operando con una latenza ridotta. A titolo esemplificativo, il risultato del filtraggio sul campione di Figura 1 è presente in Figura 8. Come si può osservare le componenti sinusoidali sono state ridotte di circa 40 dB. Resta un rumore di fondo, dovuto al fatto che si è scelta una larghezza di banda ai 3 dB $\Delta\theta_{3dB} = 200$ Hz, per non penalizzare eccessivamente il segnale utile, e che si opera in real-time, quindi si ha a disposizione un numero di campioni limitato per effettuare l'analisi in frequenza.

Si rimanda al file .wav allegato per i risultati del filtraggio sul segnale completo.

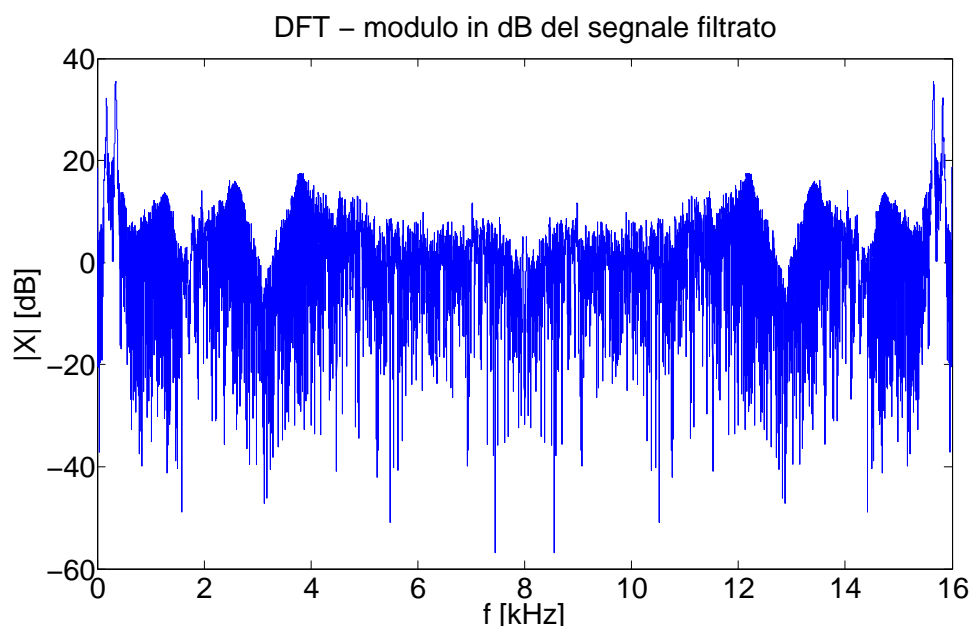


Figura 8: DFT di 250 ms di segnale filtrato

6 Conclusioni

In questa relazione si introduce una procedura per riconoscere e filtrare in tempo reale un rumore sinusoidale da un segnale utile che rappresenta del parlato. Viene descritto l'algoritmo, il tipo di filtro utilizzato e l'implementazione in MATLAB®. Infine si mostra come la rimozione delle componenti sinusoidali sia efficace, pur mantenendo la latenza ridotta.

Riferimenti bibliografici

- 1] G.T. Zhou, M.Z. Ikram, *Unsupervised detection and parameter estimation of multi-component sinusoidal signals in noise*, Signals, Systems and Computers, Conference Record of the Thirty-Fourth Asilomar Conference on, Vol 2, pagine 842 - 846, ottobre 2000