MATLAB

Esercizi (ETSI 2017-18)

(ver. 14/06/2018)

1. Introduzione

ETSI - Lezione 12/04/2018 (Appunti MartiPi)

Matrici

clc = cancella shell
clear = cancella anche la workspace
array → vettori → parentesi [ ]

array [, ,]
↑ o spazi
vettore [; ; ;] → vettore colonna
C = [1 2 3; 4 5 6; 7 8 9] → matrice

Inizializzazione matrici

C = zeros (M, N) → matrice m × n composta da zeri
C = ones (M, N) → " " " " uno
Se la voglio di 6 invece che 1
C = C + 5 posso fare anche (1 rinco)
Concatenare matrici C = [C, C] → concatena 6 due matrici
C = 3 × 3
C = 3 × 6

Accedere agli elementi

Accedere agli elementi c(2, 3) ex.
Indice primo elemento = 1, MatLab è 1 based
Comandi per ottenere righe o colonne (operatori)

: = mettere in riga
; = " " colonna
: = identifica tutti gli elementi

V = C(4, :)
riga 4 tutte le colonne
2 : end → tutte le colonne dalla 2 all'ultima
V = C(3:4, 2:3)
colonne dalla 2 alla 3
voglio dalla riga 3 alla 4

Dimensioni

size(C) → numero righe e numero colonne
C = C' ⇒ mi trasporre la matrice
C trasposta
: fa a passi di uno
t = 1 : 100 ; vettore riga } crea un vettore da 100 e va da 1 a 100
t = t' ; → vettore colonna } elementi 1 = 1, 2 = 2, 100 = 100
t = 1 : 0,5 : 100 ⇒ Va a passi di 0,5 così!

Grafici

Rette

ex: $y = 2x + 3$

↓ vettore per x. Come lo faccio?
↓ per y. Devo calcolarla per ogni valore della x. Come faccio? Scrivo semplicemente l'equazione

per MatLab non fa differenza, rip

```
x = -50:50;
```

↑ devo definire un range

```
y = 2*x + 3
```

`plot(y)` → sull'asse delle x mette solo i campioni che ha preso della retta (# campioni di y)

`plot(x,y)` → mi mette anche x

→ grafico da -50 a 50 a passi di 1. Perché non è a pintini?

MatLab interpola per default

`plot(x,y,'-x')` stesso grafico con * nei punti in cui ha calcolato la funzione

Se non voglio che interpoli:

`plot(x,y,'x')`

↑ mette solo i marcatori

INSERIRE FINESTRE

```
x label('Asse x');
y label('Asse y')
title('Retta')
```

Tracciare più funzioni

$y_1 = -2*x + 7$ Voglio y_1 e y nello stesso grafico

- apro finestra in cui aprire i grafici (ancora vuota)

→ `figure;` → apri finestra vuota

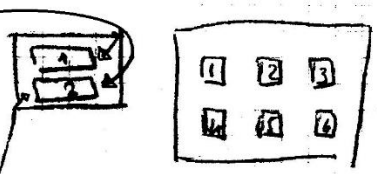
```
hold on   → metti in attesa
plot(x,y)
plot(x,y1)
```

Cambiare colore di una retta `plot(x,y,'colore')`

```
hold off
subplot(2,1,1)
plot(x,y)
subplot(2,1,2)
plot(x,y1,'k')
```

quadretti

righe colonne



ETSI - Lezione 19/04/2018 (Appunti MartiPi)

Array e Stringhe

Array

Se voglio concatenare vettori di dimensione diversa non posso fare una matrice, ma devo fare un ***cell array*** → metto le parentesi {} mentre concateno vettori.

Come accedere agli elementi dell'array: posso farlo con

- parentesi *graffe*: [ex. `M{a,b}`] danno accesso all'elemento contenuto nella cella (sostanzialmente mi restituisce il vettore) → per accedere ai valori
- parentesi *tonde*: [ex. `M(1,1)`] ottengo l'elemento cella (quello che avevo nella prima cella) ↳ per accedere all'elemento della cella

Per **accedere al singolo valore**: `M{a,b}(#elemento che voglio)`.

"`M(1,1)(2)`" mi dà errore perché non esiste l'elemento due.

Stringhe

Si definiscono col singolo apice: `s1 = '...'` → **è un array di char** (viene scritto 1x#caratteri)

Per accedere ad un carattere: `s1(#carattere)`

Se voglio accedere ad un certo numero di caratteri `s1(#primocaratterechevoglio:#numeroultimo)`

Ci sono diverse funzioni di libreria per lavorare con le stringhe. Iniziano con "str" (es: `strcat(s1, s2)`)

Le parole sono vettori di caratteri → posso concatenarle come facevo per i vettori di numeri: `[s1 s2]`

File

Due tipi di file:

- *Script* → non prevedono ingresso
- *Funzioni* → prevedono un ingresso da parte dell'utente

Script

- Inizio con `edit` + nome dello script (ci apre una finestra)
- Salva il codice in un file .m e posso eseguirlo lanciando il file.
- Quando definisco una variabile quindi non la vedo nel workspace.

Esempio di script semplice

- "`%`" = commento. "`%%`" = commento in grassetto + Matlab mi divide il codice in funzioni. Posso scrivere delle righe a caso, poi le seleziono e con CTRL+R le commento, con 'CTRL+T le decommento'.
- Se scrivo `help` + nome del file sulla *command window* ottengo tutti i commenti.
- Se premo **RUN** (freccia verde) il codice dentro allo script compare sulla *workspace*. Stessa cosa succede se sulla *command window* scrivo il nome del file (che devo aver salvato prima ovviamente).
- Gli **script generano variabili globali.**

Funzione Di Debug

Cliccare col mouse una delle righe di codice con la righetta accanto (sono quelle che hanno delle funzioni) → metto un **breakpoint.** Con **step** proseguo il codice passo passo a partire dal breakpoint. Anche se il codice è ancora nella fase di debug posso accedere (tramite la command window ovviamente) ai vettori e vari che sono presenti nelle righe di codice prima del punto in cui ho fermato il codice (non per forza il breakpoint, guardo la freccina verde).

Esercizio 1: script

(Voglio fare uno script che chieda numero di righe e colonne all'utente e riempia una matrice)

```
clc;
disp('Fill the matrix from keyboard');
nRows = input('How many rows?\n');

%voglio che lo script mi segnali gli errori, tipo se mettessi un numero
%di righe < 0
if (nRows <= 0)
    error('Number of rows must be greater than zero');
%else, se volessi aggiungercelo
end %tutto quello che si apre si chiude con la keyword end

nCols = input('How many columns?\n');

if (nCols <= 0)
    error('Number of columns must be greater than zero');
end

M = zeros(nRows, nCols); %come inizializzare la matrice

for i=1:nRows
    %1:qui definisco l'incremento, sennò è 1 di default:nRows
    for j = 1:nCols
       el = input(['Insert the element {' num2str(i) ',' num2str(j) '}\n']);
                                        %metto le parentesi quadre per concatenare
       M(i,j) = el;
    end
end

M %per stampare a video la matrice
```

Funzioni

Si crea nello stesso modo di uno script. Le **funzioni generano variabili locali**.

Esercizio 2: funzione

```
%%My first function
%inizia con la parola chiave function
%nella parentesi quadra ci sono gli output che vogliamo non importa il tipo
%il nome della funzione DEVE COINCIDERE col file.m in cui la salviamo
%tra parentesi tonde metto gli input non importa il tipo
%la funzione si chiude con la parola chiave end

%supponiamo di prendere in ingresso e farne una somma elemtento per
%elemento
%e poi elevare elemento per elemento per il corrispondente elemento nella
%matrice 2

function [sumM, expM] = myFirstFunction(M1, M2);

sumM = M1 + M2; %gestisce già elemento per elemento, non occorre il ciclo for

expM = M1 .^ M2; %con operatori come elemento e prodotto bisogna specificare che
sia elemento per elemento
%se non lo faccio, il prodotto per esempio fa prodotto di matrici
%metto un punto prima dell'operatore e me lo fa elemento per elemento .^
%per il resto fa tutto in maniera automatica

%vogliamo vedere cosa succede alle matrici

subplot(2,1,1);
plot(sumM(1, :)); %voglio stampare la riga 1
subplot(2,1,2);
plot(expM(1,:));

end

%non posso far partire il programma senza definire le matrici M1 e M2 nella
%command window
```

Nella Command window devo scrivere:

```
>> Matrix1 = [1 2 3; 4 5 6];
>> Matrix2 = [1 2 3; 8 9 10];
>>
>> [sommaM, esponenzialeM] = myFirstFunction(Matrix1, Matrix2);
```

2. Script e Funzioni

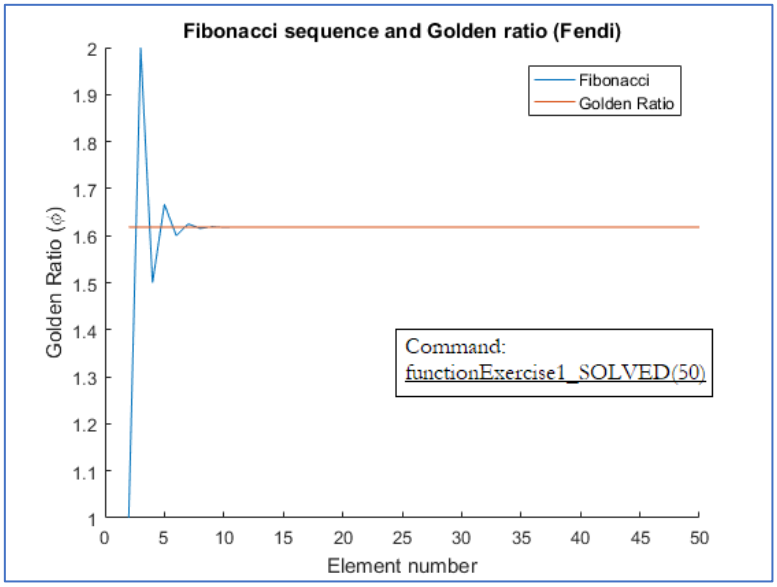
File: scriptExercise1_SOLVED.m

```
%% 1. Generate a random matrix 100x100 by using the command rand
    % Use help rand for instructions

clear;
clc;

M = rand (100, 100);

%% 2. Using a for loop, extract its main diagonal and assign it to a vector v1
[rows, cols] = size (M);

v1 = zeros(1, rows);
for i=1:rows
        v1(1,i) = M(i,i);
end
v1

%% 3. Using a for loop, extract its anti diagonal and assign it to a vector v2
v2 = zeros(1, rows);
for i=1:rows
        v2(1,i) = M(rows - (i-1), i);
end
v2
```

File: functionExercise1_SOLVED.m

```
%% Creare una funzione che dato in ingresso un numero n^N , produce la serie di
Fibonacci all'n-esimo termine.
% Mostrare il limite al GOLDEN RATIO
function nNumber = functionExercise1_SOLVED(inputN)

clc

nNumber(1) = 1;
nNumber(2) = 1;
%inserisco i primi due numeri della sequenza in un array: [ 1 1 ]

%Devo realizzare questo ciclo scritto qui in pseudocodice:
% k = 3;
% while k <= inputN
%     nNumber(k) = [nNumber(k-2) + nNumber(k-1)];
%     k = k + 1;
% end

for k=3:inputN
    nNumber(k) = (nNumber(k-2) + nNumber(k-1));
end

%creo un array che indicizzi dal 2° all'ultimo numero della serie
n = 2:inputN;
%con l'indice scorro la serie nNumber=[...] e creo un array ratio
ratio = nNumber(n)./nNumber(n-1);
% ./ is the element-per-element right division.
```

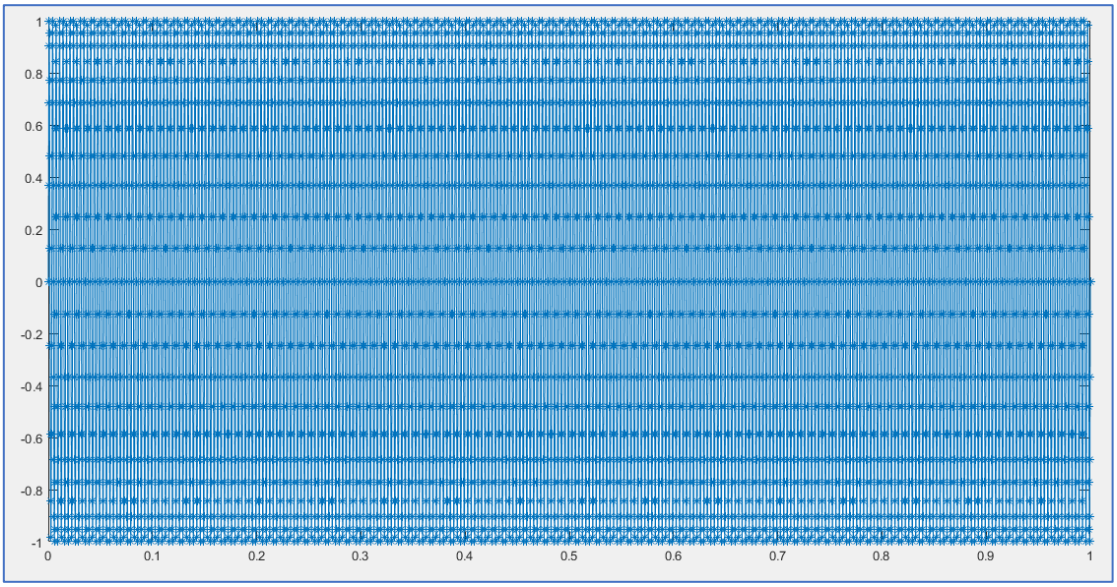
```
     % A./B is the matrix with elements A(i,j)/B(i,j).
     % A and B must have the same size, unless one of them is a scalar.
%ratio mi darà il rapporto tra numeri successivi

%mi definisco la Golden Ratio (numero aureo)
gr = (1 + sqrt(5) ) / 2;
grVector = gr * ones(1, length(nNumber) - 1);
%ones(M,N) is an M-by-N matrix of ones.


hold on
%confronto come grafici la "Fibonacci Ratio" e la Golden Ratio
plot(n,ratio);
plot(n,grVector);
%faccio il sofisticato sul grafico e lo rendo Fendi
title('Fibonacci sequence and Golden ratio (Fendi)');
xlabel('Element number');
ylabel('Golden Ratio (\phi)');
legend ('Fibonacci','Golden Ratio');
```



3. Sampling & Aliasing

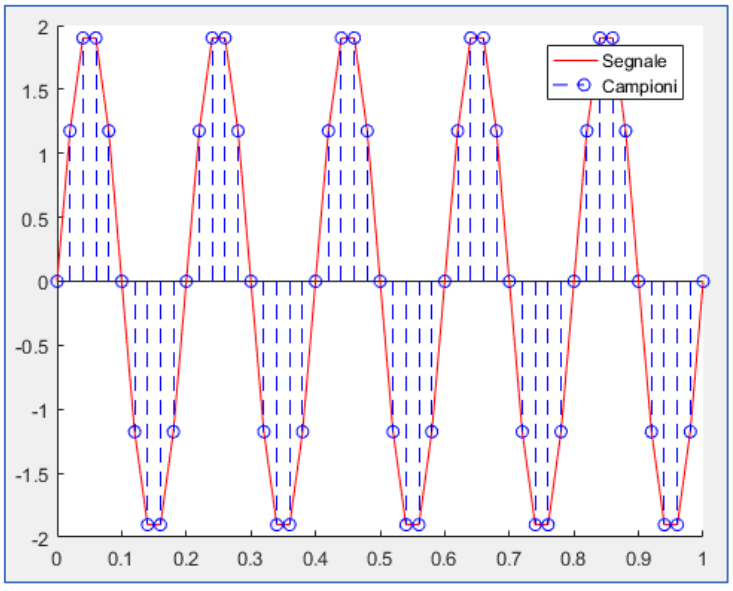
File: ex_tone.m

```
% Vogliamo creare un segnale y con determinate caratteristiche
clear all;
close all;

%Caratteristiche del segnale:
fy = 300; %frequency
Ty = 1 / fy; %period
wy = 2 * pi * fy; %omega = 2pi * frequency

%Caratteristiche del campionamento
fs = 5000; %samplig frequency
Ts = 1/fs; %sampling period

%Numero di periodi da riprodurre
numPeriod = 300;

Ttot = numPeriod * Ty; %Tempo totale (Numero periodi * singolo periodo)
t = 0 : Ts : Ttot; %Indice temporale da 0 al Tot a passi di Ts

%Costruiamo il segnale y
y = sin(wy * t);

%Riproduciamo il segnale y con la sua f di sampling
sound(y, fs);

%Tracciamo il segnale
plot(t, y, '*-');
```



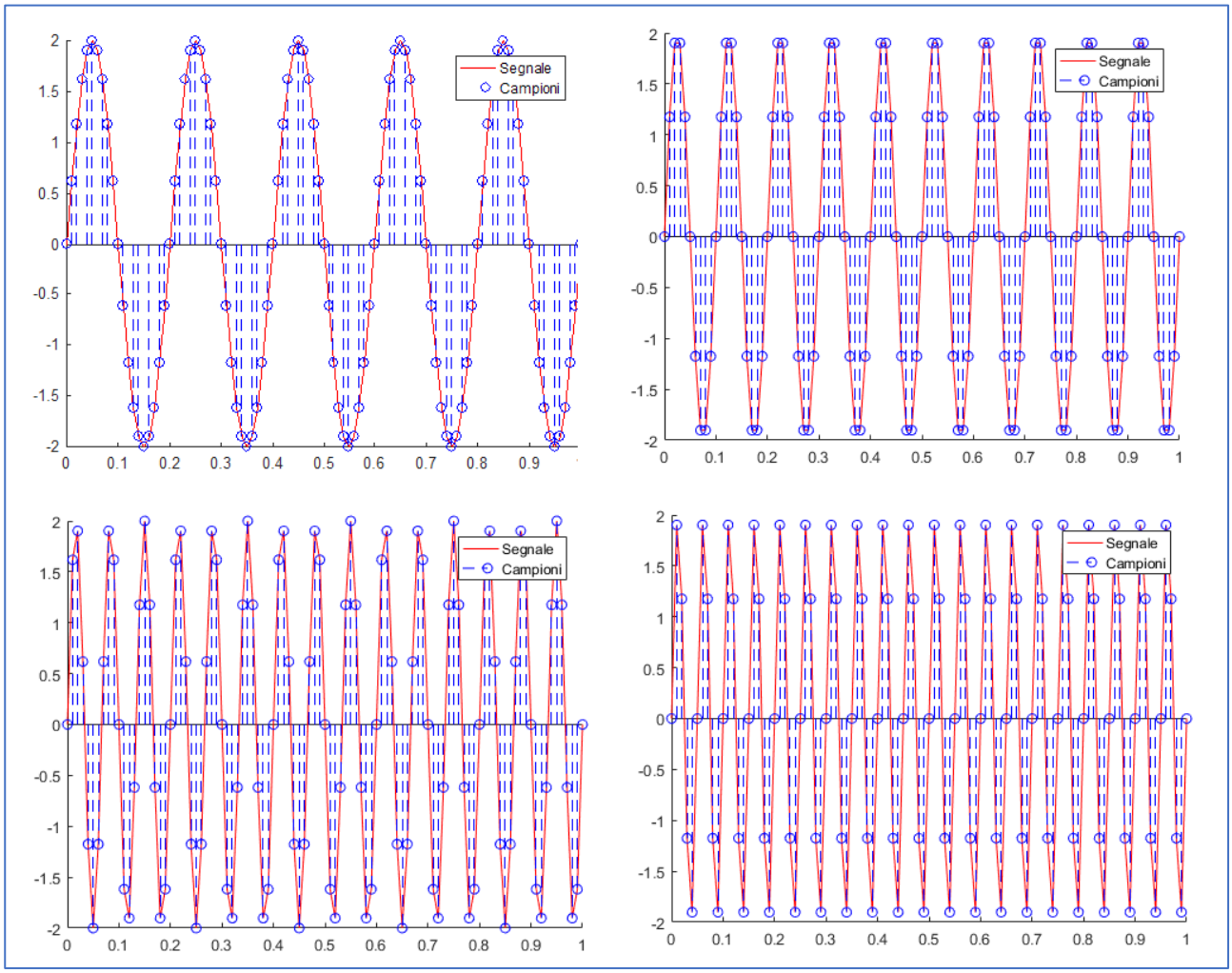
File: ex_tone2.m

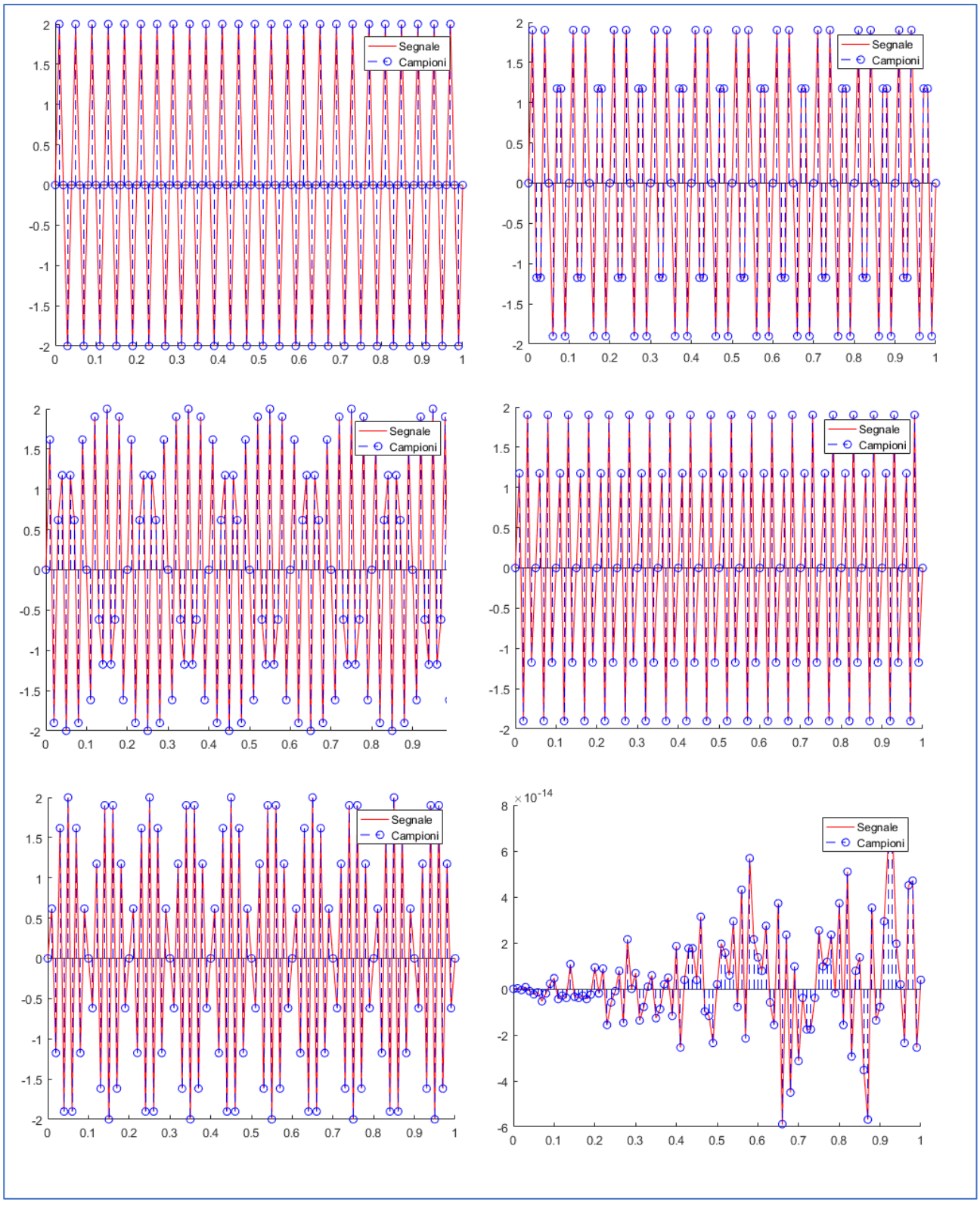
```
%%Definiamo una funzione che visualizzi il grafico ed i campioni
% dati in input Ampiezza, frequenza del segnale ed f. di campionamento
function ex_tone2(A1, f1, fs)

% Nel caso non si volessero usare gli input della funzione:
    % close all;
    % clear all;
    % f1 = 5;
    % fs = 50
    % A1 = 2;

%Calcoliamo il periodo di campionamento
Ts = 1/fs;
%Creiamo indice sui campioni di tempo
t = 0:Ts:1;

% Generiamo il segnale discreto 1 (uno)
y1 = A1*sin(2*pi*f1*t);

% Nel caso volessimo generare un segnale discreto additivo 2
    % f2 = 20;
    % A2 = 3.5;
    % y2 = A2*sin(2*pi*f2*t);
    % y = y1 + y2;

%creiamo ora una versione "campionata" del segnale
tc = t(1:1:end);
yc = y1(1:1:end);

figure;
hold on
plot(t,y1,'r'); %Visualizzo il segnale y1
stem(tc, yc,'b--'); %Visualizzo i campioni yc
      %P.S: stem(X,Y) plots the data sequence Y at the values specified in X.
legend ('Segnale','Campioni');

end
```



File: ex_tone3.m

```
%Vogliamo visualizzare ex_tone2 a frequenze via via maggiori
%Notando così l'aliasing all'aumentare della frequenza
clear all
close all

%Definisco il range di frequenze che voglio visualizzare
f = 5:5:50;

%Ed imposto la frequenza di campionamento
fs = 100;

%Determino quindi l'Ampiezza del segnale
A = 2;

%ed eseguo iterativamente la funzione
for i=1:length(f)
    ex_tone2(A, f(i), fs);
end
```

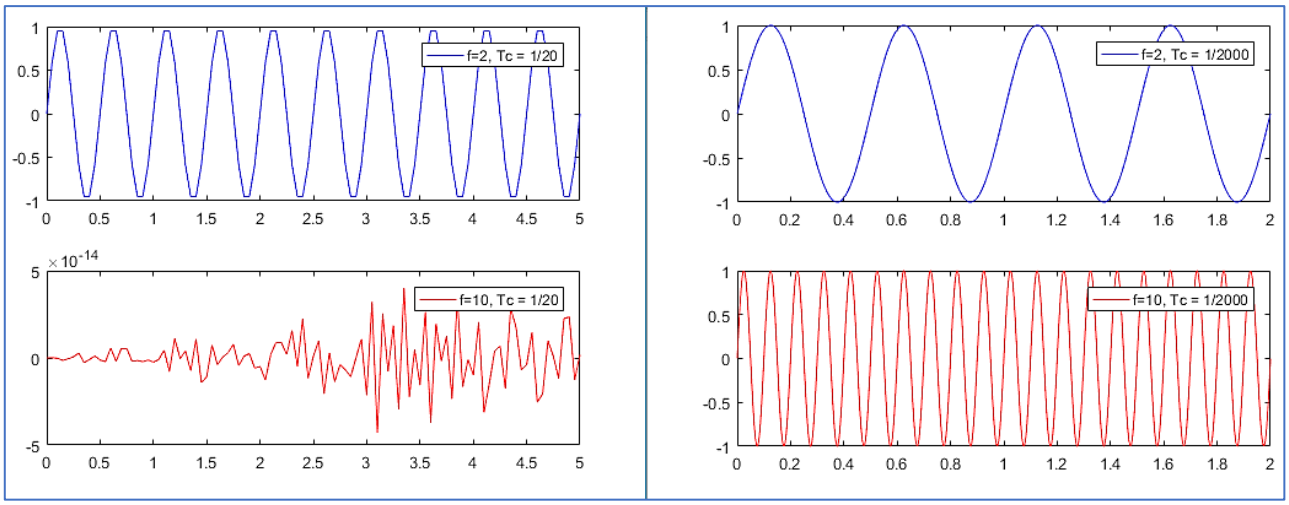




File: ex_tone4.m

```
close all
clear all

%Caratteristiche del segnale
  %Freq campionamento
    fc = 20;
  %Freq segnali 1 e 2:
```

```
    f1 = 2;
    f2 = 10; % 22 Hz

%Periodo di campionamento
dt = 1/fc;
%Indice temporale da 0 a 5 ogni periodo di campionamento
t = [0:dt:5];

%Genero i due segnali
y1 = sin(2*pi*f1*t);
y2 = sin(2*pi*f2*t);

%Visualizzo il segnale 1
subplot(2,1,1);
plot(t, y1,'b');
legend ('f=2, Tc = 1/20')
%Visualizzo il segnale 2
subplot(2,1,2);
plot(t, y2, 'r');
legend ('f=10, Tc = 1/20')

%% Adesso vediamo dt2 = un centesimo di dt:

%Periodo di campionamento
dt2 = dt/100; % = 100/fc
%Indice temporale da 0 a 2 ogni periodo di campionamento dt2
t2 = [0:dt2:2];

figure;
%Visualizzo il segnale 3 (l'1 ma con dt/100)
y3=sin(2*pi*f1*t2);
subplot(2,1,1);
plot(t2, y3,'b');
legend ('f=2, Tc = 1/2000')
%Visualizzo il segnale 4 (il 2 ma con dt/100)
y4=sin(2*pi*f2*t2);
subplot(2,1,2);
plot(t2, y4,'r');
legend ('f=10, Tc = 1/2000')
```

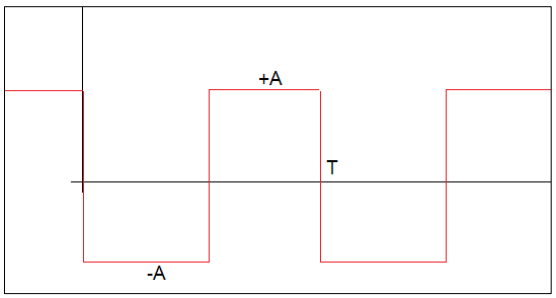


4. Serie di Fourier

Introduzione teorica

Lezione 03/05/2018

Prendiamo come segnale un'onda:



$$f = 100Hz \rightarrow T = 10ms, \qquad s(t) = \begin{cases} A, & t \in \left[0, \frac{T}{2}\right] \\ -A, & t \in \left[\frac{T}{2}, T\right] \end{cases}$$

Sappiamo di poterla "scomporre" in serie di Fourier:

$$\boxed{s(t) = a_0 + \sum_{n=1}^{+\infty} [a_n \cos(\omega_n t) + b_n \sin(\omega_n t)]}, \qquad \omega_n = 2\pi n f$$

$$a_0 = \frac{1}{T}\int_{-\frac{T}{2}}^{\frac{T}{2}} s(t)dt\ , \qquad a_n = \frac{2}{T}\int_{-\frac{T}{2}}^{\frac{T}{2}} s(t)\cos(\omega_n t)\,dt\ , \qquad b_n = \frac{2}{T}\int_{-\frac{T}{2}}^{\frac{T}{2}} s(t)\sin(\omega_n t)\,dt$$

Vogliamo implementare la formula riquadrata su MATLAB, ma prima calcoliamo "carta e penna" la serie:

$$a_0 = \frac{1}{T}\int_{-\frac{T}{2}}^{\frac{T}{2}} s(t)dt = \frac{1}{T}\left[-A\frac{T}{2} + A\frac{T}{2}\right] = 0$$

↳ non ci stupisce che il segnale sia a media nulla.

$$a_n = \frac{2}{T}\left[\int_{-\frac{T}{2}}^{0}(-A)\cos(\omega_n t)\,dt + \int_{0}^{\frac{T}{2}} A\cos(\omega_n t)\,dt\right] = \frac{2}{T}\left[-A\frac{\sin(\omega_n t)}{\omega_n}\right]_{-\frac{T}{2}}^{0} + \frac{2}{T}\left[A\frac{\sin(\omega_n t)}{\omega_n}\right]_{0}^{\frac{T}{2}} =$$

$$\frac{2}{T}\left[-A\left(0 - \frac{\sin\left(\omega_n \frac{T}{2}\right)}{\omega_n}\right) + A\left(\frac{\sin\left(\omega_n \frac{T}{2}\right)}{\omega_n} - 0\right)\right] = 0 \quad \rightarrow \quad a_n = 0, \forall n$$

↳ ciò significa che nella serie di Fourier "non ci saranno coseni".

$$b_n = \frac{2}{T}\left[\int_{-\frac{T}{2}}^{0}(-A)\sin(\omega_n t)\,dt + \int_{0}^{\frac{T}{2}} A\sin(\omega_n t)\,dt\right] = \frac{2}{T}\left[A\frac{\cos(\omega_n t)}{\omega_n}\right]_{-\frac{T}{2}}^{0} + \frac{2}{T}\left[-A\frac{\cos(\omega_n t)}{\omega_n}\right]_{0}^{\frac{T}{2}} =$$

$$\frac{2}{T}\left[A\left(\frac{1}{\omega_n} - \frac{\cos\left(\omega_n \frac{T}{2}\right)}{\omega_n}\right) - A\left(\frac{\cos\left(\omega_n \frac{T}{2}\right)}{\omega_n} - \frac{1}{\omega_n}\right)\right] = \frac{2A}{T\omega_n}\left[1 - \cos\left(\omega_n \frac{T}{2}\right) - \cos\left(\omega_n \frac{T}{2}\right) + 1\right] =$$

$$\frac{4A}{T\omega_n}\left[1 - \cos\left(\omega_n \frac{T}{2}\right)\right] = \frac{4A}{T\omega_n}[1 - \cos(n\pi)] \quad \rightarrow \quad b_n = \begin{cases} 0 & n\ pari \\ \frac{8A}{T\omega_n} = \frac{4A}{\pi n} & n\ dispari \end{cases}$$

Perciò posso scrivere la serie come:

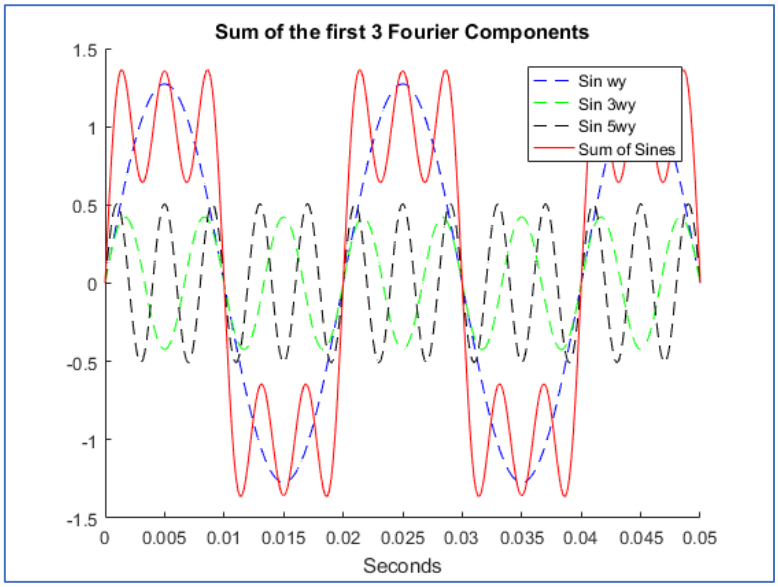
$$s(t) = \sum_{n\ dispari} \left[\frac{4A}{\pi n}\sin(\omega_n t)\right] = \sum_{n=1}^{+\infty} \left[\frac{4A}{\pi(2n+1)}\sin(\omega_{2n+1} t)\right]$$

$$b_1 = \frac{2}{\pi} = 0.66\,, \qquad b_2 = \frac{2}{3\pi} = 0.21\,, \qquad b_3 = \frac{3}{5\pi} = 0.19\,, \qquad (\dots)\,, \qquad b_n \xrightarrow[n\to\infty]{} 0$$

File: coeffFourier.m

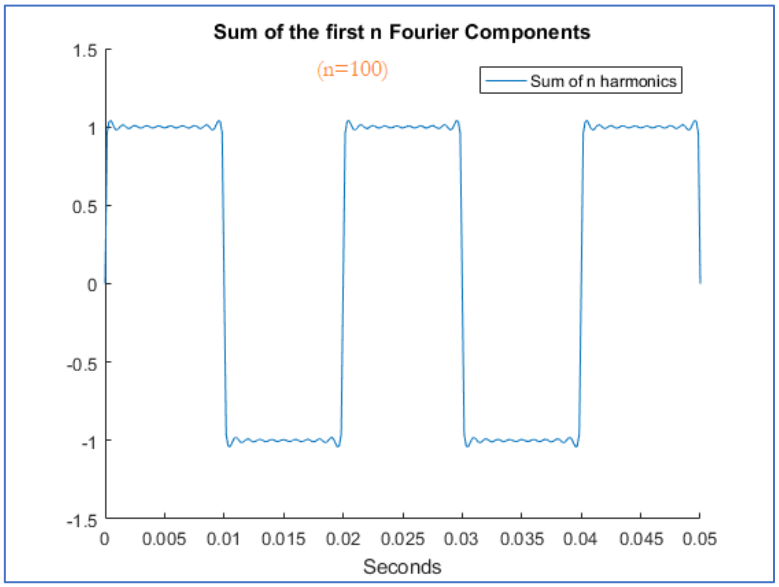
```
%% SERIE DI FOURIER
%Voglio provare a prendere un segnale e usare un'onda quadra ...?

clc;
close all;

%% Somma di tre armoniche

%Definiamo un segnale di ampiezza generica A
A = 1;
%Impostiamo la frequenza di questo segnale
fy = 100; %100Hz
%calcoliamoci il periodo
T = 1/fy;
%calcoliamo anche la pulsazione
wy = 1*pi*fy;

%Scegliamo la frequenza di campionamento
%Ma abbiamo visto che al limite di Nyquist gli effetti sono deleteri!
%Perciò la scegliamo abbondante:
fs = 6000;
%Calcoliamo il periodo di campionamento
tiv = 1/fs;

%Ci sarà un vettore t che, a passi dell'intervallo di campionamento,
    % andrà fino a un numero di periodi che decidiamo (es: 5)
t = 0 : tiv : 5*T;

%Cominciamo a sommare i primi tre coefficienti non nulli
%Dal calcolo "su carta" abbiamo visto che sono b1, b3 e b5
b1 = 4*A/(1*pi);
b3 = 4*A/(3*pi);
b5 = 8*A/(5*pi);
s1 = b1*sin(1*wy*t);
s3 = b3*sin(3*wy*t);
s5 = b5*sin(5*wy*t);
%Eseguo poi la sommatoria sui primi tre coefficienti:
s = s1 + s3 + s5;

figure;
hold on %Voglio mettere tutte le "plot" nella stessa figura

% Definisco il range degli assi (asse x da 0 a 5T, y da -1.5A a 1.5A)
axis([0 5*T -1.5*A 1.5*A]);

%Visualizzo le singole componenti tratteggiate
plot(t,s1, 'b--');
```

```
plot(t,s3, 'g--');
plot(t,s5, 'k--');

%Visualizzo il risultato in rosso
plot(t,s, 'r'); %ascisse = t, ordinate = s

%Gestisco il layout della figura:
    %Etichetta sull'asse delle ascisse
    xlabel('Seconds');
    %Metto il titolo alla figura
    title('Sum of the first 3 Fourier Components');
    %Metto la legenda alla figura
    legend('Sin wy','Sin 3wy','Sin 5wy','Sum of Sines');
```

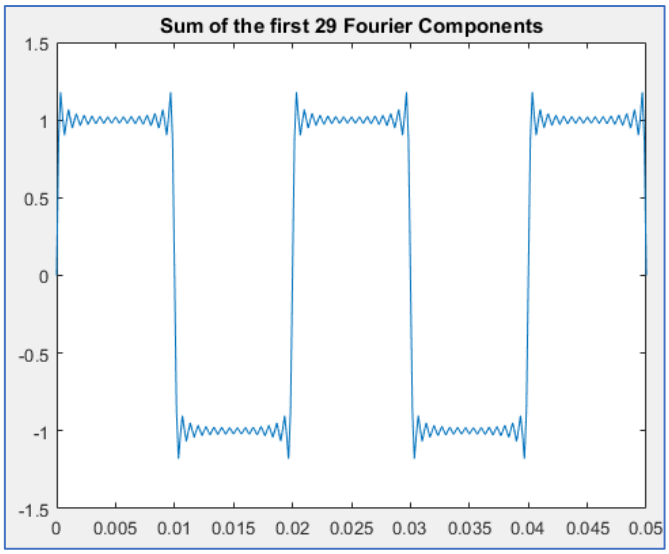


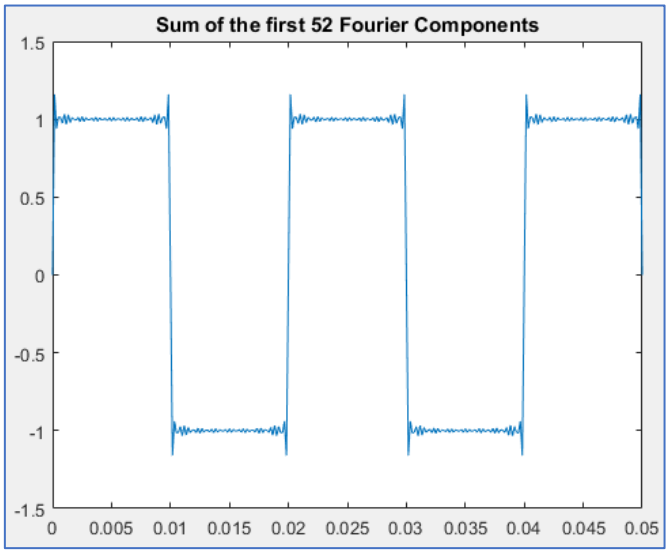

```
%% Somma di più armoniche

%Definiamo il numero di armoniche da sommare
nHarmonics=100;
%Inizializzo il segnale a cui poi dovrò sommare le armoniche
sH=0;

%Calcolo i coefficienti:
for n = 1 : nHarmonics
    %L'operatore mod restituisce il resto della divisione (% in C)
    if (mod(n,2)== 0) %l'indice "i" è pari
        bn = 0;
    else %l'indice "i" è dispari
        bn = 4*A/(pi*n);
    end
    %aggiungo l'armonica al segnale sH
    sH = sH + bn*sin(wy*n*t);
end

figure
hold on
%Definisco anche qui gli assi
axis([0 5*T -1.5*A 1.5*A]);
%Visualizzo poi il grafico
plot(t,sH);

%Gestisco il layout della figura:
```

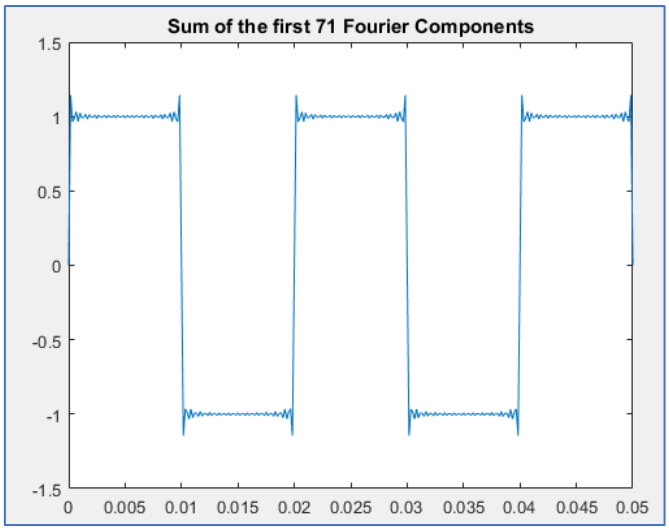
```
    %Etichetta sull'asse delle ascisse
    xlabel('Seconds');
    %Metto il titolo alla figura
    title('Sum of the first n Fourier Components');
    %Metto la legenda alla figura
    legend('Sum of n harmonics');
```




```
%% Somma di più armoniche (2)
% Adesso voglio visualizzare i grafici delle varie somme

nHarmonics2=100;
sH2=0;

figure;

for i = 1 : nHarmonics2
    if (mod(i,2)== 0)
        bn2 = 0;
    else
        bn2 = 4*A/(pi*i);
    end
    sH2 = sH2 + bn2*sin(wy*i*t);
    axis([0 5*T -1.5*A 1.5*A]);
    plot(t,sH2);
    title(['Sum of the first ' num2str(i) ' Fourier Components']);
    getframe;
end
```


Sum of the first 29 Fourier Components
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


Sum of the first 52 Fourier Components
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


Sum of the first 71 Fourier Components
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

5. DFT

Introduzione teorica

Lezione 07/05/2018

La trasformata di Fourier di un segnale $x(t)$ è:

$$X(f) = \int_{-\infty}^{\infty} x(t)e^{-j2\pi ft}dt = \int_{-\infty}^{\infty} x(t)\underbrace{[\cos(2\pi ft) + j\sin(2\pi ft)]}_{Formula\ di\ Eulero}dt$$

È sostanzialmente un'estensione della serie di Fourier con $T \to \infty$.

t è una variabile di tipo continuo che si applica a segnali di tipo contino. Per applicarla a segnali di tipo discreto bisogna passare alla DFT:

$$\left[\begin{matrix}\text{Espressione}\\ \text{di Analisi}\end{matrix}\right]: \ X(k) = \sum_{n=0}^{N-1} x(n)e^{-j\frac{2\pi k}{N}n}, \qquad k \in [0, N-1]$$

$$\left[\begin{matrix}\text{Espressione}\\ \text{di Sintesi}\end{matrix}\right]: \ x(n) = \frac{1}{N}\sum_{k=0}^{N-1} X(K)e^{+j\frac{2\pi k}{N}n}, \qquad n \in [0, N-1]$$

La DFT "mantiene la periodicità". Se facciamo la trasformata di una sequenza $x(n)$ di periodo N, vediamo che:

$$X(k+N) = \sum_{n=0}^{N-1} x(n)e^{-j\frac{2\pi(k+N)}{N}n} = \sum_{n=0}^{N-1} x(n)e^{-j\frac{2\pi k}{N}n}\underbrace{e^{-j\,2\pi}}_{=1} = X(k)$$

C'è una forte assonanza tra la trasformata di Fourier continua e quella discreta, infatti:

$$X(f) = \int_{-\infty}^{\infty} x(t)e^{-j2\pi ft}dt \ \overset{???}{\to} \ X(k) = \sum_{n=0}^{N-1} x(n)e^{-j\frac{2\pi k}{N}n} \quad \Rightarrow \quad f \to \frac{k}{N}, \ t \to n, \ \int \to \sum$$

Al posto di f ci metto $k/N \to$ l'indice che andiamo a utilizzare è quindi funzione del periodo della sequenza trasformata!

$X(k)$ è un numero complesso, vediamo perché:

$$x(K) = x(0)\cdot e^{-j\frac{2\pi\,k}{N}0} + x(1)\cdot e^{-j\frac{2\pi\,k}{N}1} + x(2)\cdot e^{-j\frac{2\pi\,k}{N}2} + \cdots =$$

$$= x(0)\left[\cos\left(-\frac{2\pi\,k}{N}0\right) + j\sin\left(-\frac{2\pi\,k}{N}0\right)\right] + x(1)\left[\cos\left(-\frac{2\pi\,k}{N}1\right) + j\sin\left(-\frac{2\pi\,k}{N}1\right)\right] + \cdots =$$

$$= \left\{x(0)\cos\left(\frac{2\pi\,k}{N}0\right) + x(1)\cos\left(-\frac{2\pi\,k}{N}1\right) + \cdots\right\} + j\left\{-x(0)\sin\left(\frac{2\pi\,k}{N}0\right) - x(1)\sin\left(-\frac{2\pi\,k}{N}1\right) - \cdots\right\} =$$

$$= A_k + j\,B_k \in \mathbb{C}$$

Vediamo quindi un'altra analogia: $X(k)$ non è altro che una somma, una *serie*, di seni e di coseni pesati da dei coefficienti (i campioni del segnale in ingresso), esattamente come era la serie di Fourier! Ricordiamo poi che i numeri complessi

possono essere rappresentati come vettori sul piano complesso aventi modulo e fase, perciò:

$$M = \sqrt{A_k^2 + B_k^2}, \qquad \theta = tg^{-1}\left(\frac{B_k}{A_k}\right)$$

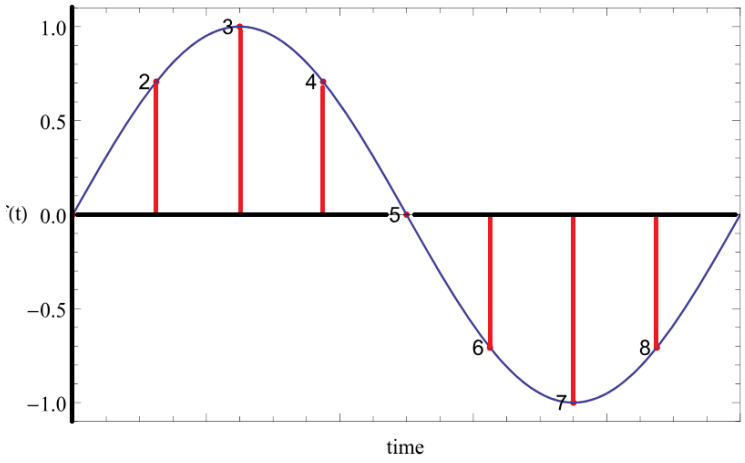
Possiamo però andare oltre e vedere che:

$$A_k + j\,B_k = [A_0 + j\,B_0] + [A_1 + j\,B_1] + \cdots, \qquad M_i = \sqrt{A_i^2 + B_i^2}, \qquad \theta_i = tg^{-1}\left(\frac{B_i}{A_i}\right)$$

$$A_i + j\,B_i = x(i)\cos\left(\frac{2\pi\,k}{N}i\right) - j\cdot x(i)\cdot \sin\left(\frac{2\pi\,k}{N}i\right)$$

Perciò noi stiamo scomponendo il nostro segnale in una sequenza di seni e coseni ognuno avente un determinato modulo ed una certa fase.

Facciamo un esempio: campioniamo un seno a 1Hz di frequenza prendendo 8 campioni in un secondo (campionato a 8Hz):



$$x(t) = sen(t), \qquad A = 1, \qquad \theta = 0, \qquad N = 8 \rightarrow f_s = 8Hz, \qquad x(n) = \left[0, \frac{\sqrt{2}}{2}, 1, \frac{\sqrt{2}}{2}, 0, -\frac{\sqrt{2}}{2}, -1, -\frac{\sqrt{2}}{2}\right]$$

Calcoliamo i coefficienti della DFT:

$$X(k=0) = \sum_{n=0}^{N-1} x(n)e^{-j\,\frac{2\pi 0}{N}n} = \sum_{n=0}^{N-1} x(n) = 0$$

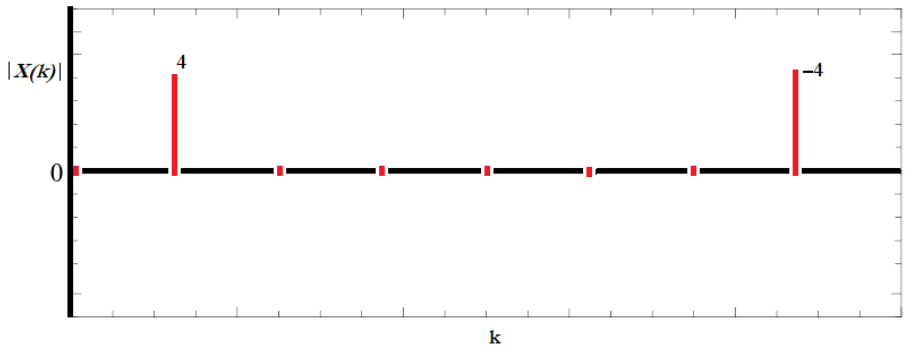
$$X(K=1) = x(n)e^{-j\,\frac{2\pi 1}{N}0} + x(n)e^{-j\,\frac{2\pi 1}{N}1} + x(n)e^{-j\,\frac{2\pi 1}{N}2} + \cdots = 0 + \frac{\sqrt{2}}{2}e^{-j\frac{\pi}{4}} + 1e^{-j\frac{\pi}{2}} + \cdots =$$

$$= 0 + \frac{\sqrt{2}}{2}\left[\cos\left(\frac{\pi}{4}\right) - j\cdot\sin\left(\frac{\pi}{4}\right)\right] + 1\left[\cos\left(\frac{\pi}{2}\right) - j\cdot\sin\left(\frac{\pi}{2}\right)\right] + \cdots = (\ldots) = -4j$$

Andando avanti con i conti otteniamo:

$$X(k) = [0, -4j, 0, 0, 0, 0, 0, 4j]$$

Vediamone graficamente il modulo $|X(k)|$:



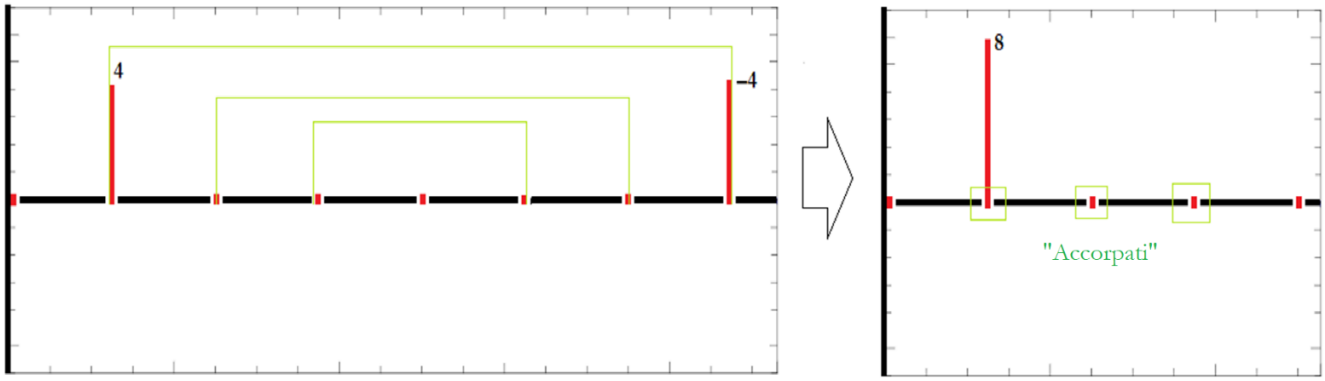
Questa è la DFT del mio segnale di partenza.

Essendo $f=k/N$, ogni volta che aumento di k aumento frequenzialmente di f_s/N → questa è la **risoluzione frequenziale**: ho una risoluzione frequenziale di 1Hz: apprezzo le componenti in frequenza del mio segnale se sono >1Hz, avendo campionato a 8Hz prendendo 8 campioni. Se voglio aumentare il potere risolutivo della mia DFT o aumento il numero di campioni presi o la frequenza di campionamento. Ricordiamoci però che aumentare N significa avere un segnale di partenza più lungo, tempi di osservazione più lunghi. Attenzione: risoluzione ≠ frequenza massima.

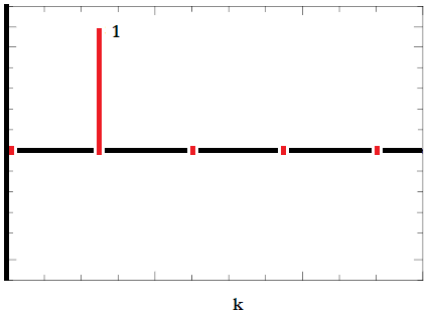
Quello che vedo dalla DFT è che ho un seno a $1Hz$ avendo un picco di energia in $1Hz$ ($k=1$), ma il picco in $k=7$ che mi significa? Sfruttiamo la simmetria hermitiana della DFT per rendere più intuitiva la DFT:

$$X\left(\frac{N}{2}\right)=\sum_{n=0}^{N-1}x(n)e^{-j\,\frac{2\pi N}{N\,2}\,n}=\sum_{n=0}^{N-1}x(n)e^{-j\,2\pi\,n}=\sum_{n=0}^{N-1}x(n)\left[\cos(\pi n)-j\cdot\underbrace{\sin(\pi n)}_{=0}\right]$$

Perciò il campione centrale non ha parte immaginaria, come nemmeno il campione iniziale $X(0)$. Posso quindi ora mettere insieme i campioni simmetrici, quelli con parte immaginaria non necessariamente nulla(?):



Normalizzando rispetto al numero di campioni presi ($N=8$) troviamo poi:



Adesso ho quello che mi aspettavo di trovare: a $k = 1$ $(1Hz)$ ho un seno di ampiezza 1, e non ho altro nelle altre frequenze.

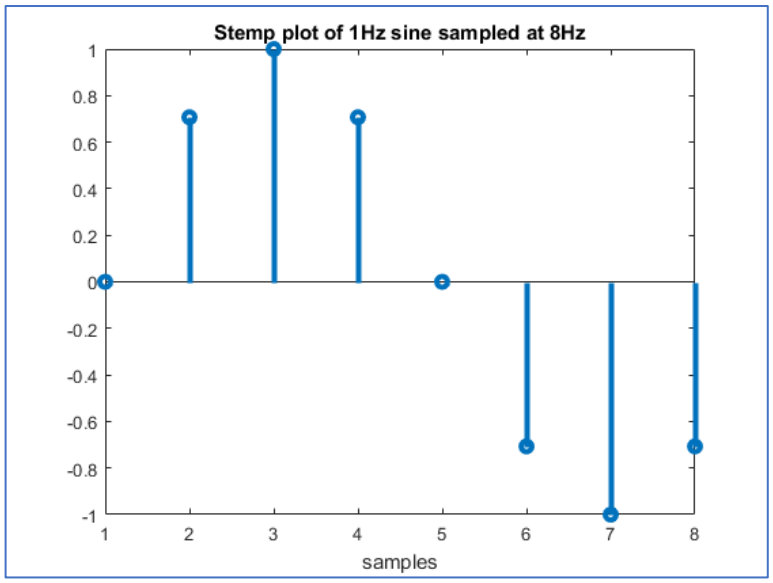
Adesso non mi rimane che guardare la fase:

$$X(k = 1) = -4j \quad \rightarrow \quad \angle X(1) = \begin{cases} +\frac{3}{2}\pi \\ -\frac{1}{2}\pi \end{cases}$$

(…) mancante la conclusione sulla interpretazione del valore della fase (…)

File: dft.m

```
%% Discrete Fourier Transformation
%  Facciamo su MATLAB quello che abbiamo visto "su carta"

close all

f = 1; %frequenza del seno (1Hz)
A = 1; %ampiezza del seno

fs= 8; %frequenza di campionamento (8Hz)
tiv = 1/fs; %periodo di campionamento
N = 8; %Number of samples

%Generiamo il nostro tono
t= 0 : tiv : 1-tiv; %generiamo l'asse dei tempi
x = A*sin(2*pi*f*t);

%Visualizzo i campioni
stem(x, 'LineWidth',3); %con una larghezza della linea pari a 3
xlabel('samples');
title('Stemp plot of 1Hz sine sampled at 8Hz');
```



X

1x8 complex double

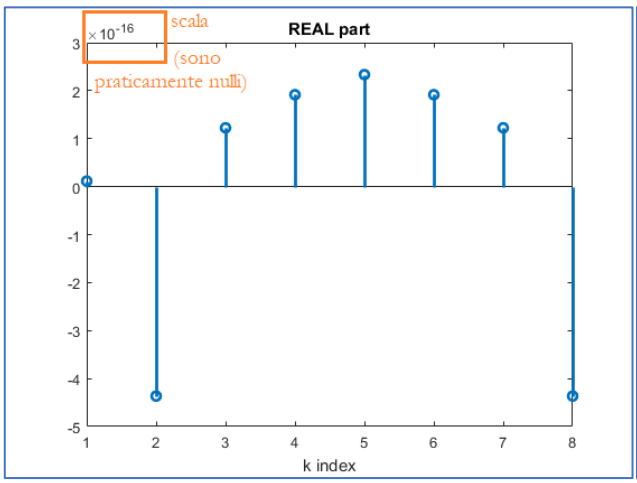
	1	2
1	1.1442e-17 + 0.0000e+00i	-0.0000 - 4.0000i
2	praticamente nullo	-4j

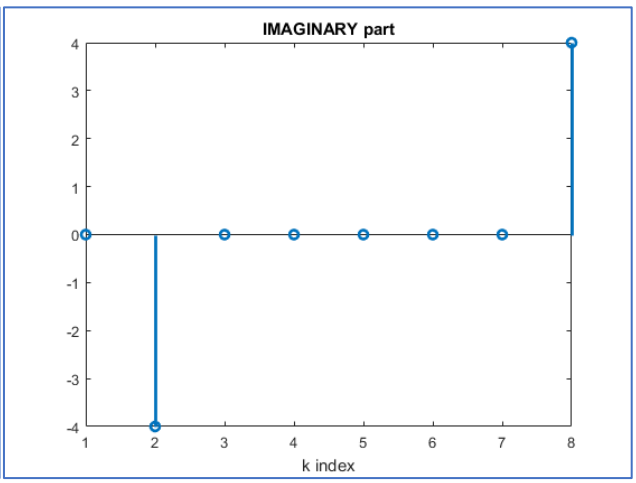
3	4
1.2246e-16 - 1.1102e-16i	1.9155e-16 - 4.4409e-16i
praticamente nullo	praticamente nullo

5	6
2.3349e-16 + 0.0000e+00i	1.9155e-16 + 4.4409e-16i
praticamente nullo	praticamente nullo

7	8
1.2246e-16 + 1.1102e-16i	-0.0000 + 4.0000i
praticamente nullo	+4j

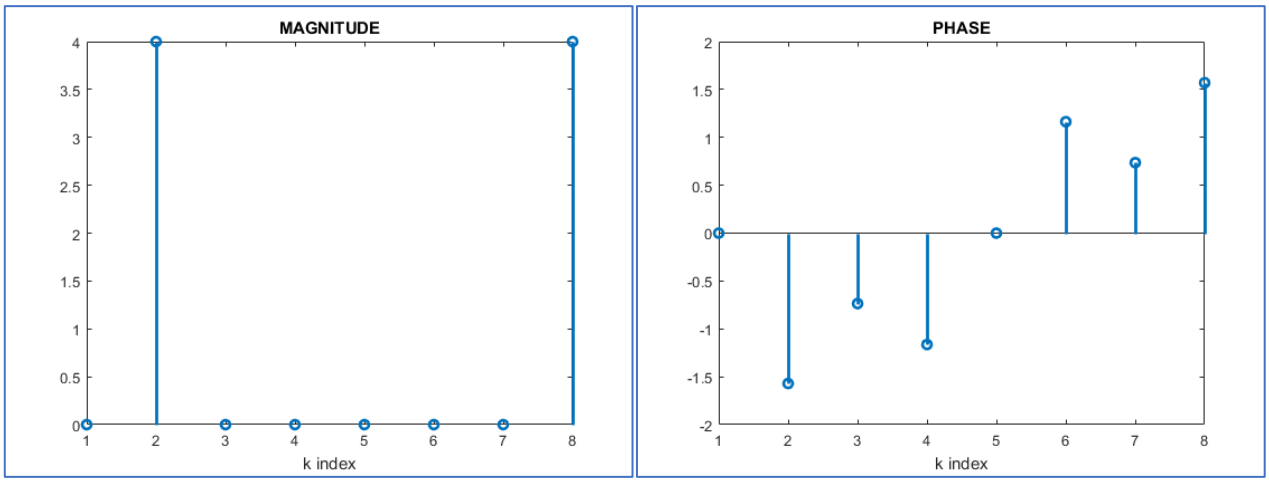
```
%Adesso dobbiamo calcolare la DFT
X = fft(x); %calcoliamo la DFT con l'algoritmo FFT

%Adesso divido parte reale ed immaginaria
realX = real(X);
imagX = imag(X);
%e le visualizzo separate
figure
stem(realX, 'LineWidth',2); %con una larghezza della linea pari a 2
xlabel('k index');
title('REAL part');
figure
stem(imagX, 'LineWidth',2);
xlabel('k index');
title('IMAGINARY part');
```



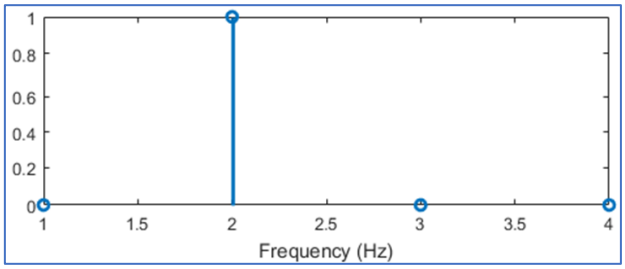



```
%Allo stesso modo posso ragionare su modulo e fase
magX = abs(X); %Magnitude = modulo
phaseX = angle(X);
%e li visulizzo
figure
stem(magX, 'LineWidth',2);
xlabel('k index');
title('MAGNITUDE');
```

```
figure
stem(phaseX, 'LineWidth',2);
xlabel('k index');
title('PHASE');
```




```
%Sfrutto la simmetria e prendo solo i primi N/2 campioni
Xnew = X(1 : floor(N/2)); %MATLAB parte da 1, sarebbe il campione X(0)
%Abbiamo messo "floor" arrotondando per eccesso nel caso di N dispari

%Se invece volessi fare come ho fatto nel foglio di teoria:
Xnew = 2*X(1 : floor(N/2));
Xnew = Xnew/N; %Lo ho normalizzato

%Costruisco il vettore delle frequenze
f= (0: floor(N/2))*(fs/N);
%questo è l'indice sequenziale, notare che è in funzione di fs e di N
%inftatti (fs/N) è la risoluzione frequenziale

%Adesso visualizzo il modulo
magXnew = abs(Xnew); %Magnitude = modulo
figure
stem(magXnew, 'LineWidth',2);
xlabel('Frequency (Hz)');
```

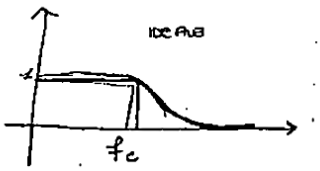


6. Filtro di Butterworth

Introduzione teorica

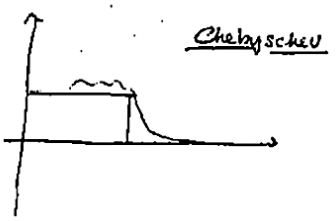
(Vedi Dispense sui Filtri Analogici – Appendice)

Lezione 10/05/2018 *(Appunti MartiPi)*

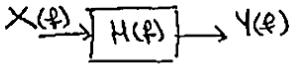


Butterworth

$$|H(s)| = |H(j\omega)| = \frac{1}{1+\left(\frac{\omega}{\omega_c}\right)^{2N}}$$



All'aumentare dell'ordine N sono più reipidi ma:
- Butterworth ha una fase sempre meno lineare
- Chebyschev ha più ripple



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

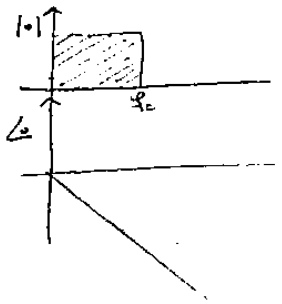
$X = |X| e^{j\angle X}$

$Y = |H| e^{j\angle H} |X| e^{j\angle X} = \underbrace{|X| \cdot |H|}_{|Y|} e^{j\underbrace{\angle X + \angle H}_{\angle Y}}$

$Y = |Y| e^{j\angle Y}$

$Z = |Z| e^{j\angle Z}$

Vorrei una fase lineare. Perché? Per non distorcere il segnale. Perché la fase lineare non distorce? Perché introduce un ritardo proporzionale alla frequenti. Sfasamento in questo caso è funzione lineare della frequenza: un 2Hz ha ritardo doppio rispetto a un 1Hz ma ha anche una frequenza doppia → in uscita ogni componente è sfasata della stessa quantità.



$$\omega_{NORMALIZZATA} = \frac{f_{car}}{(f_s/2)}$$

↓

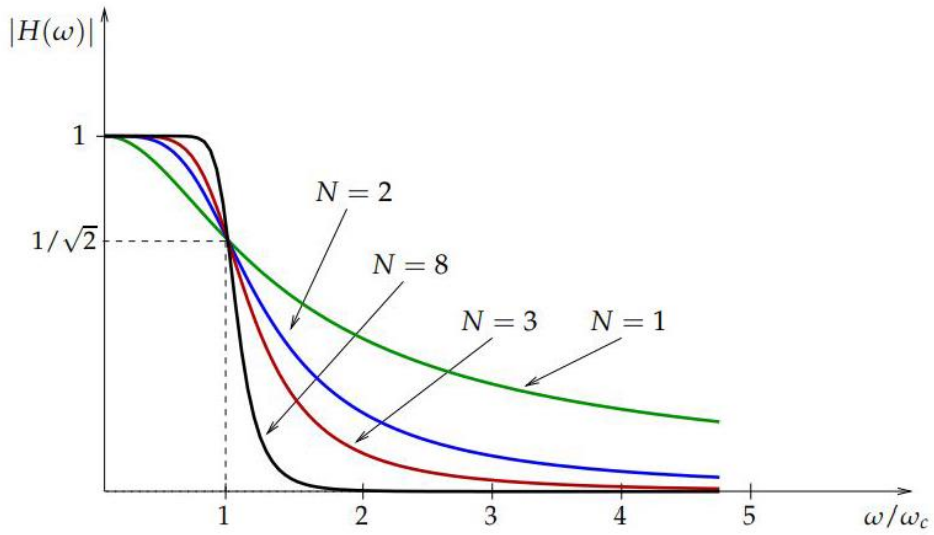
uno dei parametri della funzione Butterworth (MATLAB)

File: ButterworthSquareWave.m

```
%% FILTRAGGIO BUTTERWORTH DI ONDA QUADRA

%In questo esempio creiamo un'onda quadra con la somma
% delle prime 100 armoniche della serie di fourier
% e poi la filtriamo butterworth

clc;
close all;

%% SERIE DI FOURIER (SOMMA PRIME 100 ARMONICHE)

%Definiamo un segnale di ampiezza generica A
A = 1;
%Impostiamo la frequenza di questo segnale
fy = 100; %100Hz
%calcoliamoci il periodo
T = 1/fy;
%calcoliamo anche la pulsazione
wy = 2*pi*fy;

%Scegliamo la frequenza di campionamento
%Ma abbiamo visto che al limite di Nyquist gli effetti sono deleteri!
%Perciò la scegliamo abbondante:
fs = 6000;
%Calcoliamo il periodo di campionamento
tiv = 1/fs;

%Ci sarà un vettore t che, a passi dell'intervallo di campionamento,
% andrà fino a un numero di periodi che decidiamo (es: 5)
t = 0 : tiv : 5*T;

%Definiamo il numero di armoniche da sommare
nHarmonics=100;
%Inizializzo il segnale a cui poi dovrò sommare le armoniche
s=0;

%Calcolo i coefficienti:
for n = 1 : nHarmonics
%L'operatore mod restituisce il resto della divisione (% in C)
if (mod(n,2)== 0) %l'indice "i" è pari
bn = 0;
else %l'indice "i" è dispari
bn = 4*A/(pi*n);
end
%aggiungo l'armonica al segnale sH
s = s + bn*sin(wy*n*t);

end

% ------------------------------------------------------------------
% e fino a qua era esattamente come l'esercizio con serie di Fourier

%% BUTTERWORTH FILTERING OF SQUARE WAVE

%Definisco la frequenza di taglio del filtro butterworth
fcut = 500;
%Definisco l'ordine del filtro butterworth
N = 10;
```

```
%Definisco la frequenza di taglio normalizzata per il butterworth
normW = fcut / (fs/2);

%Genero la funzione di trasferimento del filtro
[b, a] = butter(N, normW);
    % returns the transfer function coefficients
    % of an nth-order lowpass digital Butterworth filter
    % with normalized cutoff frequency normW.
```

↓ (Dalle dispense sui Filtri Analogici in appendice) ↓



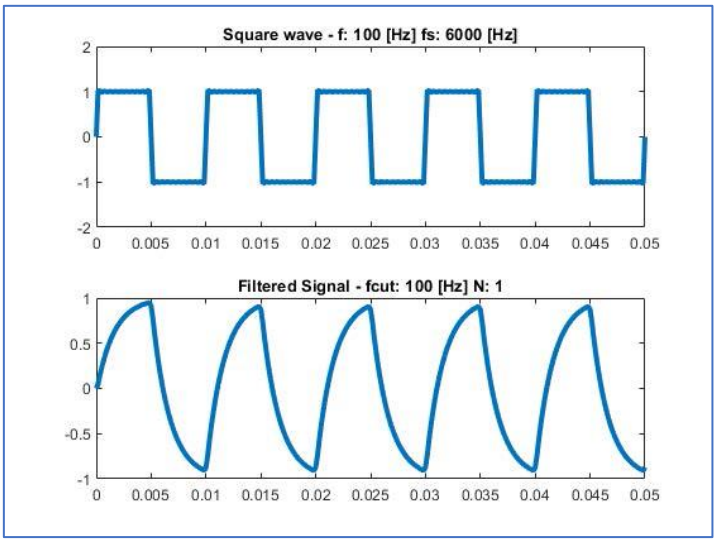
Riguardo alla figura si possono fare le seguenti osservazioni:

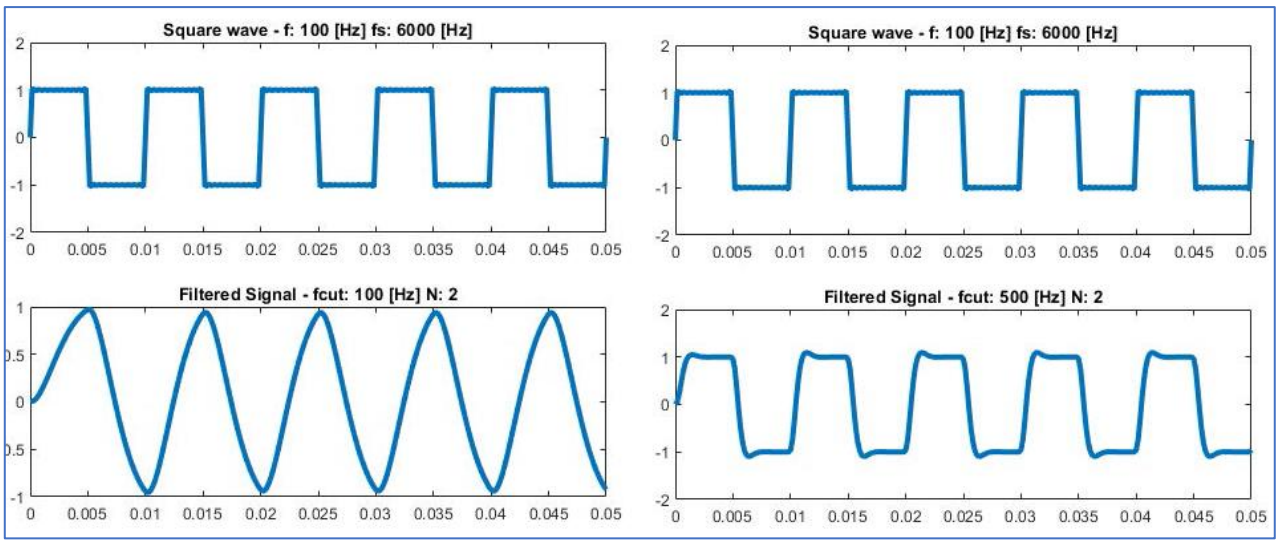
- la frequenza di taglio a 3 dB ω_c, è indipendentemente dall'ordine N del filtro;
- l'attenuazione nella banda proibita dipende da N in modo critico: risulta infatti un'attenuazione di $20N$ dB per decade;
- non sono presenti oscillazioni né in banda passante né in banda proibita: il filtro di Butterworth è quello che presenta la maggior "piattezza" in banda passante.

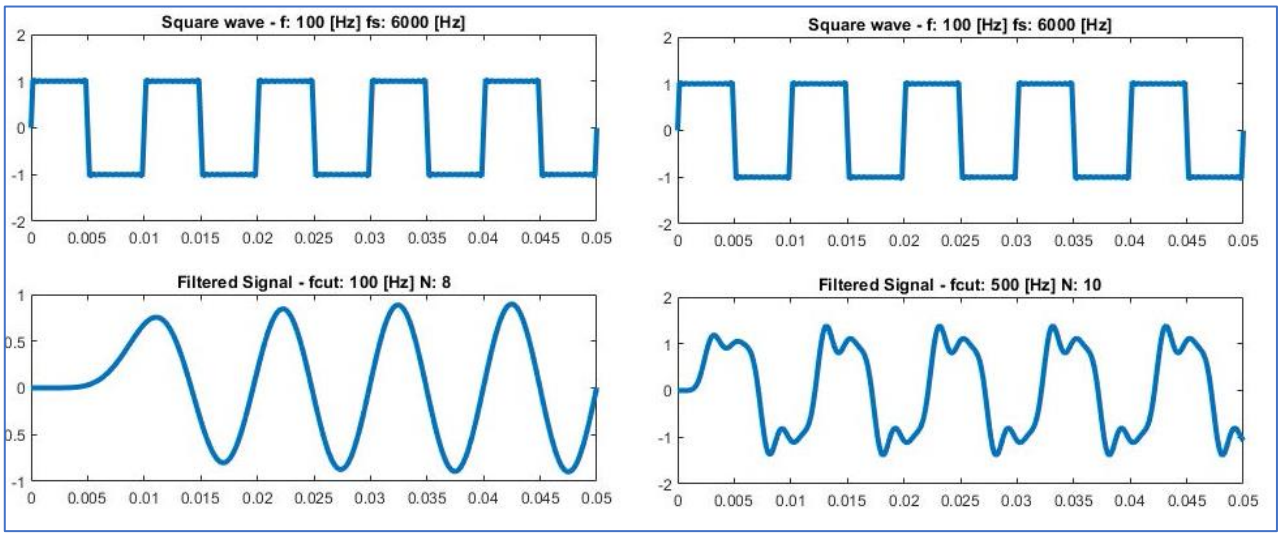
In una tipica situazione di progetto, il parametro ω_c è fissato essendo la frequenza di taglio desiderata, mentre l'ordine N viene scelto in modo tale da soddisfare la richiesta di attenuazione in banda proibita.

```
%Filtro s e genero il segnale filtrato y
y = filter(b, a, s);

%Rappresento sia s che y
figure;

%segnale originale s
subplot(2,1,1);
plot(t, s, 'LineWidth', 3);
title(['Square wave - f: ' num2str(fy) ' [Hz] fs: ' num2str(fs) ' [Hz]']);

%segnale filtrato y
subplot(2,1,2);
plot(t, y, 'LineWidth', 3);
title(['Filtered Signal - fcut: ' num2str(fcut) ' [Hz] N: ' num2str(N)]);
```

↓ (Esempi fatti aumentando e variando ordine e frequenza di taglio nel file MATLAB) ↓

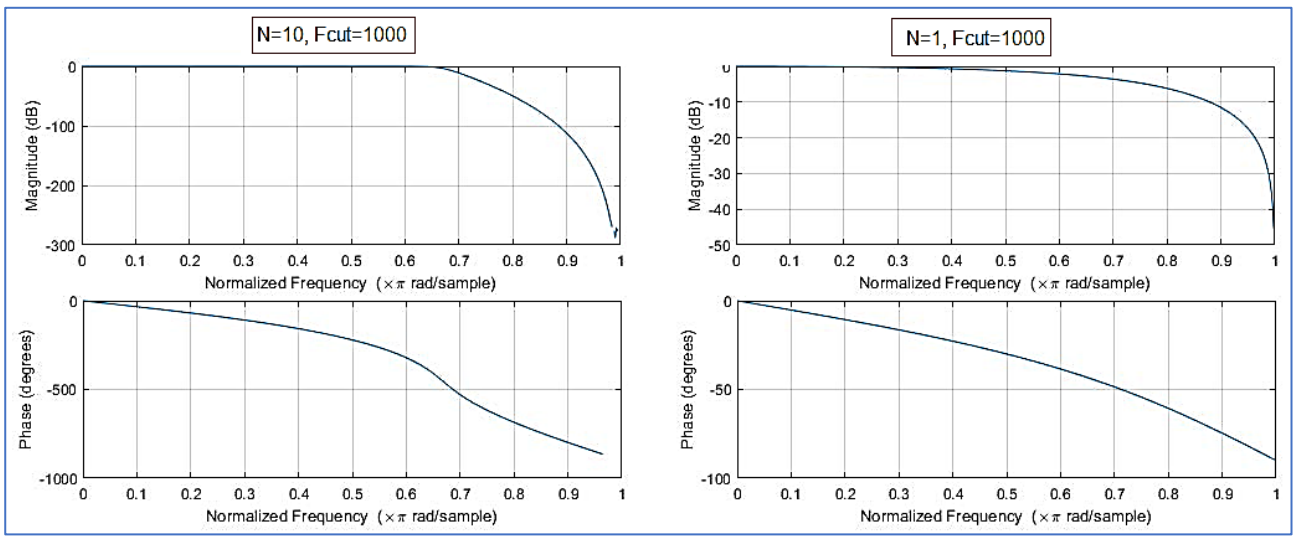




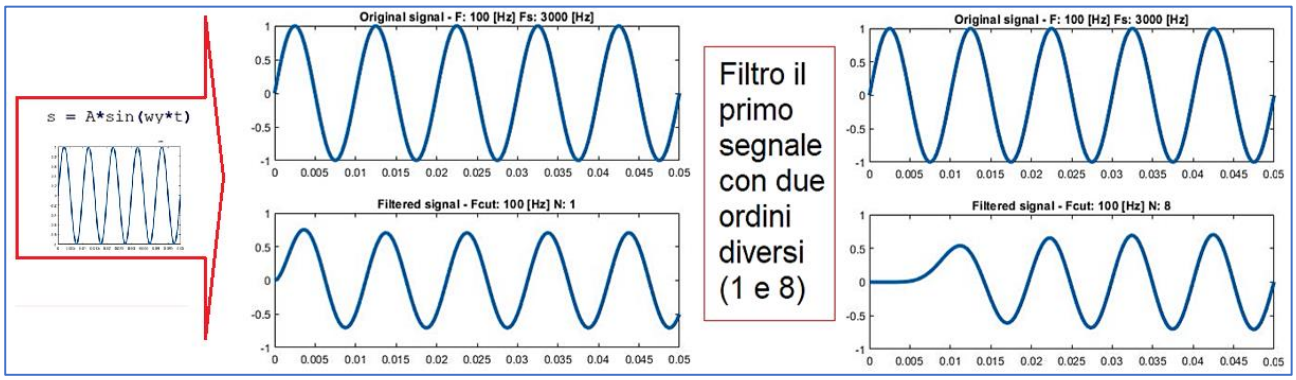


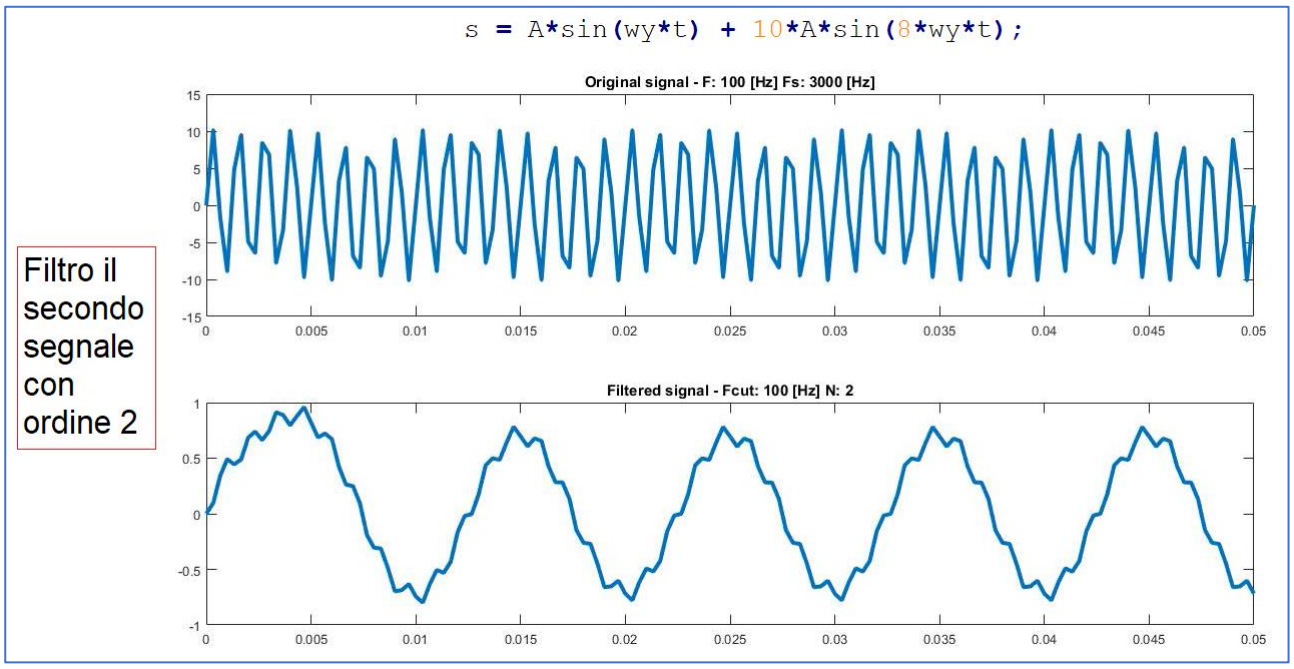
File: ButterworthSusoidalWave.m

```
%% FILTRAGGIO BUTTERWORTH DI TONO PURO

%In questo esempio creiamo un tono puro (sinusoidale)
% e poi la filtriamo butterworth

clc;
close all;

% Sinusoidal wave parameters:
    %Definisco ampiezza e frequenza
    A = 1;
    fy = 100; %100Hz
    %Calcolo periodo e pulsazione
    T = 1/fy;
    wy = 2 * pi * fy;

%Sampling parameters:
    fs = 3000;
    tiv = 1/fs;

%Genero l'asse dei tempi
t = 0:tiv:5*T;

%Alcuni Test li faccio con questo segnale:
    %s = A*sin(wy*t);
    %plot(t, s, 'LineWidth', 3);
%Poi li faccio con questo nuovo segnale:
    s = A*sin(wy*t) + 10*A*sin(8*wy*t);

% Butterworth filter parameters
fcut = 100; %frequenza di taglio del filtro
normW = fcut/(fs/2); %frequenza normalizzata
N = 6; %ordine del filtro

% Genero i coefficienti del filtro
[b, a] = butter(N, normW);

% Visualize Filter Response
figure;
freqz(b, a);
```



```
% Permorm the filtering
y = filter(b, a, s);

% Visualize Filtered Signal
figure;
% Segnale originale s
    subplot(2,1,1);
    plot(t, s, 'LineWidth', 3);
    title(['Original signal - F: ' num2str(fy) ' [Hz] Fs: ' num2str(fs) '
[Hz]']);
% Segnale filtrato y
    subplot(2,1,2);
    plot(t, y, 'LineWidth', 3);
    title(['Filtered signal - Fcut: ' num2str(fcut) ' [Hz] N: ' num2str(N)]);
```

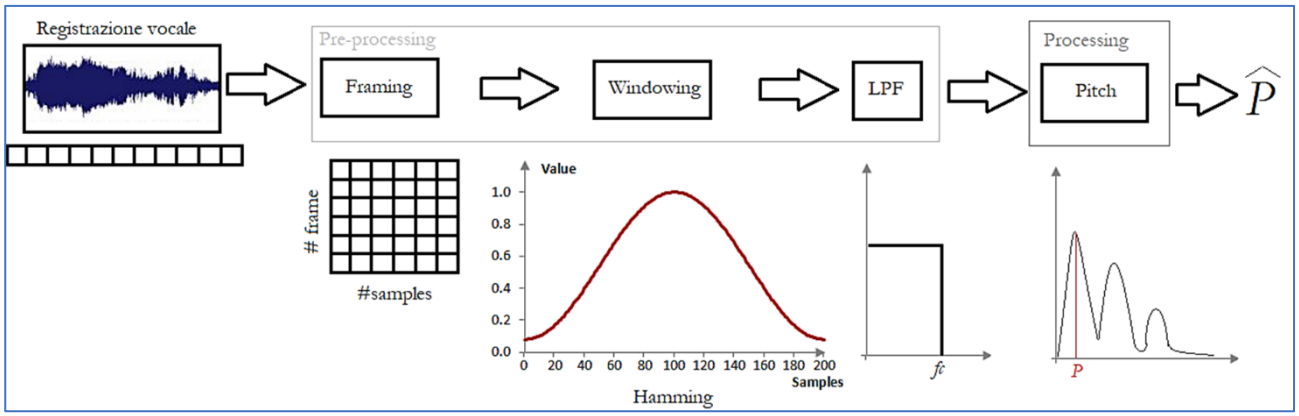




7. Elaborazione del segnale – Frequenza di Pitch

Introduzione teorica

ETSI - Lezione 14/05/2018



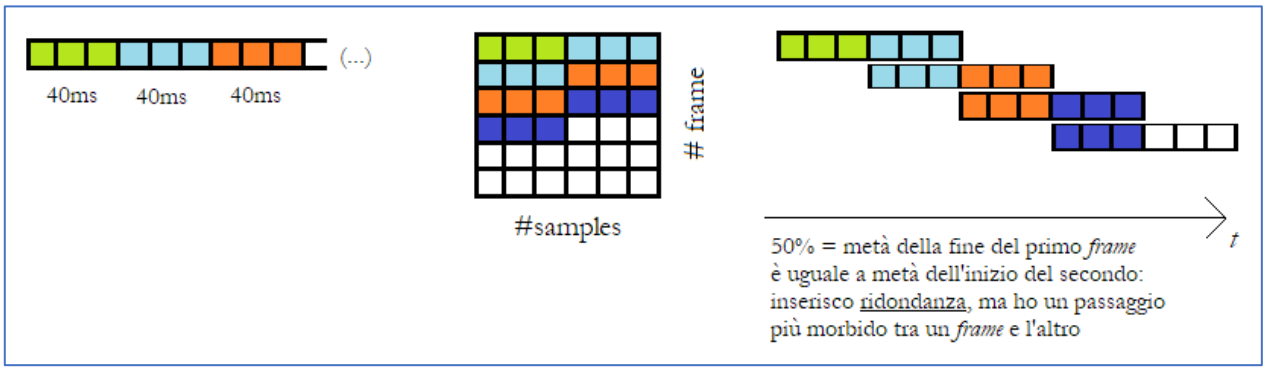
Elaborazioni:

- **Framing**: Il segnale vocale può essere reso stazionario prendendo frame di intervalli temporali da 20-40ms.
- **Windowing**: rendere più morbido possibile il segnale in uscita, enfatizzando i campioni centrali (finestra di Hamming).
- **LPF**: filtraggio passabasso a frequenza di taglio f_c.
- **Pitch**: andremo a stimare la frequenza fondamentale della componente della voce.

Per fare questo in MATLAB dovremo scrivere una funzione che registri in maniera sincrona e non asincrona, per poi eseguire tutte le elaborazioni necessarie. Per registrare e visualizzare il segnale registrato usiamo la funzione scritta da noi:

```
function myVoice = voiceRecorder(recordLenghtInSec,fs)
```

Quando taglio in frame voglio che il taglio sia il più morbido possibile. Se io volessi una sovrapposizione del 50% (`overlapPercentage=0.5`) vuole dire che voglio la prima parte (tipo i primi 40ms) sulla prima riga della matrice, e poi voglio che sul frame dopo venga ripreso il 50% nell'altro:



Il numero di righe dipende dal numero di campioni che voglio nel mio frame. Quanti campioni per frame utilizziamo? Il segnale vocale è considerato stazionario tra i 20 e i 40ms, facciamo quindi frame da 20-40 e vediamo quanti campioni

ci vengono in 20-40ms → ha senso un numero in funzione della durata del frame (40ms):

$$\#\text{SamplePerFrame} = f_s \cdot \underbrace{(40 \cdot 10^{-3})}_{\text{durata frame}}$$

Per il *framing* useremo la funzione:

```
function toFrames (audioData,fs, overlapPercentage)
```

Per la finestratura uso la funzione:

```
function outSignal = doWindowing(inputSignal)
```

Per il *filtering* bisogna sapere che tutto ciò che è più grande di 1KHz è difficile che sia voce parlata: un passabasso con f_c a 1KHz aumenta la robustezza del segnale. Applicheremo quindi un Butterworth con frequenza normalizzata 1KHz:

$$\omega_T = \frac{f_c}{f_s/2}$$

ETSI - Lezione 17/05/2018

Dobbiamo, dopo aver eseguito il filtraggio LPF con Butterworth, identificare la frequenza di Pitch. Dobbiamo prima di tutto ricavare lo spettro di potenza con la funzione:

```
function [ps,f] = powerSpectrum(audioSamples,  fs, doPlot)
```

Poi calcolo il Pitch con:

```
function pitch = estimatePitch(audioSamples, fs)
```

Creiamo una funzione che fa tutta l'elaborazione e che ci grafica il Pitch e ne calcola la media:

```
function pitches = processData (audioSamples,fs)
```

(Il filtraggio a 500Hz aiuta che il rumore non disturbi eccessivamente il calcolo del Pitch. Ciononostante, a volte notiamo che si "satura" il Pitch attorno a 500Hz.)

- Vediamo prima tutte le funzioni, poi vediamo lo script che le esegue ↓↓↓

File: voiceRecorder.m

```
function myVoice = voiceRecorder(recordLenghtInSec,fs)

% audiorecorder creates an 8000 Hz, 8-bit, 1 channel audiorecorder object.
% A handle to the object is returned.

% audiorecorder(Fs, NBITS, NCHANS) creates an audiorecorder object with
% sample rate Fs in Hertz, number of bits NBITS, and number of channels NCHANS.
% Common sample rates are 8000, 11025, 22050, 44100, 48000, and 96000 Hz.
% The number of bits must be 8, 16, or 24. The number of channels must
% be 1 or 2 (mono or stereo).

myVoice = audiorecorder(fs,8,1);
    % 1 canale, 8 bit, frequenza di campionamento in ingresso

% audiorecorder Properties:
    % StartFcn  - Function to execute one time when recording starts.
    % StopFcn   - Function to execute one time when recording stops.
```

```
%ci servono per gestire la sincronicità delle funzioni

myVoice.StartFcn = 'disp(''Start Recording'')';
myVoice.StopFcn = 'disp(''Done Recording'')';

%Questa è una chiamata a "record" asincrona
%record (myVoice,recordLenghtInSec)

%voglio una chiamata sincrona: uso il seguente metodo:
% recordblocking - Record, and do not return control
     % until recording completes. This method requires a second input
     % for the length of the recording in seconds:
     % recordblocking(recorder,length)
recordblocking (myVoice,recordLenghtInSec);

%alla fine myVoice conterrà i campioni audio

%la funzione infatti ritorna myVoice (vedi heading)

end
```

File: toFrames.m

```
function frames = toFrames (audioData,fs, overlapPercentage)

%Lunghezza frame: 40ms
nSamplePerFrame = fs*0.04;
nSampleOverlapped = round(overlapPercentage*nSamplePerFrame);

% buffer: Buffer a signal vector into a matrix of data frames.
%Y = buffer(X,N)
    %partitions signal vector X into nonoverlapping data
    %segments (frames) of length N.  Each data frame occupies one
    %column in the output matrix, resulting in a matrix with N rows.
%Y = buffer(X,N,P)
    %specifies an integer value P which controls the amount
    %of overlap or underlap in the buffered data frames.
    % - If P>0, there will be P samples of data from the end of one frame
    %   (column) that will be repeated at the start of the next data frame.
    % - If P<0, the buffering operation will skip P samples of data
    %   after each frame, effectively skipping over data in X, and thus
    %   reducing the buffer "frame rate".
    % - If empty or omitted, P is assumed to be zero
    %   (no overlap or underlap).

frames = buffer (audioData,nSamplePerFrame,nSampleOverlapped);
%La funzione buffer mette i dati IN COLONNA
% (al contrario di come vogliamo noi la matrice)
% quindi la traspongo
frames = frames';

end
```

File: doWindowing.m

```
function outSignal = doWindowing(inputSignal, doPlot)
    %doPlot è una variabile che se a 0 non disegna grafico
    %       altrimenti disegna grafico
    %       (è una buona pratica di programmazione)
```

```
    %hamming - returns the N-point symmetric Hamming window in a column
        % hamming(N,SFLAG) generates the N-point Hamming window
        %  using SFLAG window sampling. SFLAG may be either
        %  'symmetric' or 'periodic'. By default, a symmetric window
        %  is returned.

    %Creo la finestra di Hamming
    L = length(inputSignal);
    h = hamming(L);
    %la voglio però in riga e non in colonna
    h = h';

    %filtro il segnale
    outSignal = inputSignal .* h;

    if (doPlot)
        figure
        %Visualizzo i segnali
        %segnale originale
        subplot (2,2,1);
        plot(inputSignal,'LineWidth', 1);
        title ('Original Time Signal');
        %Segnale finestrato
        subplot (2,2,2);
        plot(outSignal,'LineWidth', 1);
        title ('Windowed Time Signal');
        %Finestra
        subplot (2,2,3);
        plot(h,'LineWidth', 1);
        title ('Hamming Window');
    end
end
```

File: doFilter.m

```
function outSignal = doFilter(inputSignal, N, fs, fcut, doPlot)
    %doPlot è una variabile che se a 0 non disegna grafico
    %       altrimenti disegna grafico
    %       (è una buona pratica di programmazione)


    %Genero i coefficienti del filtro di Butterworth
    normW = fcut / (fs/2);
    [b,a] = butter (N, normW);

    %Eseguo il filtraggio
    outSignal = filter(b,a,inputSignal);

    if (doPlot)
        %Visualizzo i segnali
        figure
        %segnale pre filtering
        subplot (2,1,1);
        plot(inputSignal,'LineWidth', 1);
        title(['Pre-filtering Time Signal - N: ' num2str(N) ' Fs: '
                num2str(fs) ' [Hz] Fcut: ' num2str(fcut) ' [Hz]']);
        %Segnale finestrato
        subplot (2,1,2);
        plot(outSignal,'LineWidth', 1);
```

```
        title(['Filtered Time Signal - N: ' num2str(N) ' Fs: '
                num2str(fs) ' [Hz] Fcut: ' num2str(fcut) ' [Hz]']);

        %Visualizzo il filtro
        figure
        %Filtro
        freqz(b, a);
        title (['Butterworth Filter - N: ' num2str(N) ' Fs: '
                num2str(fs) ' [Hz] Fcut: ' num2str(fcut) ' [Hz]']);
    end

end
```

File: powerSpectrum.m

```
function [ps,f] = powerSpectrum(audioSamples,fs,doPlot,SignalName)
    %doPlot è una variabile che se a 0 non disegna grafico
    %       altrimenti disegna grafico
    %       (è una buona pratica di programmazione)

    %SignalName è una variabile che ho messo io per stampare il nome
    %           del segnale ogni volta (sprintf nella figura)

    %Capiamo prima di tutto quanto è grosso il frame audio in ingresso
    [r, N] = size(audioSamples); %numero righe e colonne di audioSample

    %Genero l'asse dei tempi
    tiv = 1/fs;
    t = 0 : tiv : (tiv*(N-1));

    %Genero la trasformata di Fourier
    y = fft(audioSamples);
    % y sono i coeff. di Fourier
    %devo prima di tutto farne i moduli
    P2 = abs(y);
    %normalizzo il modulo rispetto al numero di elementi
    P2 = P2/N;
    %prendo i primi N/2 elementi (arrotondati per difetto)
    P1 = P2(1 : floor(N/2));
    %moltiplico per 2 tutti gli elementi tranne il primo e l'N/2-esimo
    % (vedi teoria e pratica della FFT)
    P1(2 : (end-1)) = 2* P1(2 : (end-1));

    %generiamo l'asse delle frequenze
    %voglio che vada da 0 a fs/2, a passi di fs/N
    f = fs/N * (0 : (floor(N/2)-1) );
    %scrivo la cosa in questo modo arzigogolato
    %(le N si semplificano alla fine)

    %genero il power spectrum
    ps = power (P1,2);

    if (doPlot)
        figure;
        %segnale nel tempo
        subplot(2,1,1);
        plot(t,audioSamples);
        xlabel('Time [s]');
        title( [sprintf(SignalName) ' Frequency signal']);
```

```
        %power spectrum
        subplot(2,1,2);
        plot(f,ps);
        xlabel('Frequency [Hz]');
        title( [ sprintf(SignalName) ' Frequency signal']);
    end
end
```

File: estimatePitch.m

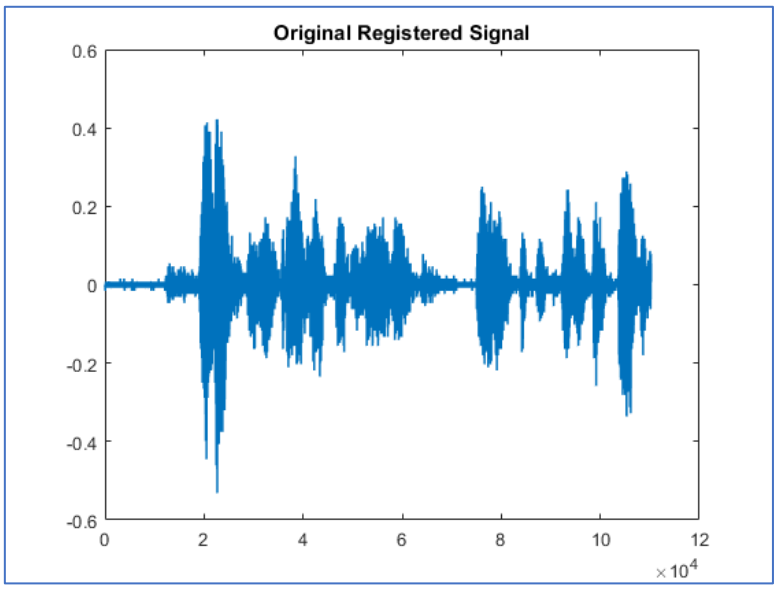
```
function pitch = estimatePitch(audioSamples, fs)

    [ps,f] = powerSpectrum (audioSamples,fs,false,'DoNotDisplayThis');

    [m, idx] = max(ps); %m è il massimo, idx è l'indice massimo

    pitch = f(idx);

end
```

File: processData.m

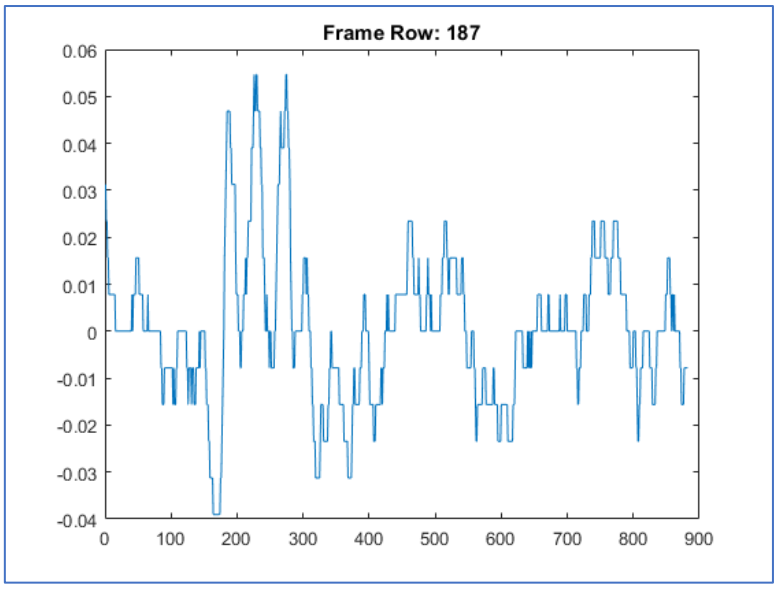
```
function pitches = processData (audioSamples,fs)
    %questa funzione fa tutto

    %frequenza di taglio del LPF
    fcut = 500;

    %Dividi in frame
    overlapPercentage = 0.50; % overlap del 50%
    frames = toFrames(audioSamples,fs,overlapPercentage);

    [numFrames, numSamples] = size (frames);

    %alloco prima l'array (aumento l'efficenza)
    pitches = zeros (1, numFrames);

    for i=1 : numFrames
        %estraggo un frame
        singleFrame = frames (i, :);
        %adesso eseguiamo il windowing
        frameWindowed = doWindowing(singleFrame, false);
        %eseguo il filtraggio
        frameFiltered = doFilter(frameWindowed, 2, fs, fcut, false);
        %calcolo il pitch
        pitches(i) = estimatePitch(frameFiltered, fs);
    end

    %visualizzo tutti i pitch
    figure
    plot (pitches)
    xlabel('Number of Frames');
    ylabel('Pitch [Hz]');

    %visualizzo il valor medio di pitch
    disp(['Average Pitch:' num2str(mean(pitches)) ' Hz']);

end
```

File: script1.m

```
%% Registrazione
%Eseguo voiceRecorder per 5 secondi
clc
clear
close all

fs=22050;
recordingLenght=5;

myVoice = voiceRecorder(recordingLenght,fs);

%adesso estraiamo i campioni audio in un array audioSamples
audioSamples = getaudiodata(myVoice);

%visualizziamo l'array audioSamples
figure
plot (audioSamples, 'LineWidth',1);
title('Original Registered Signal');
```

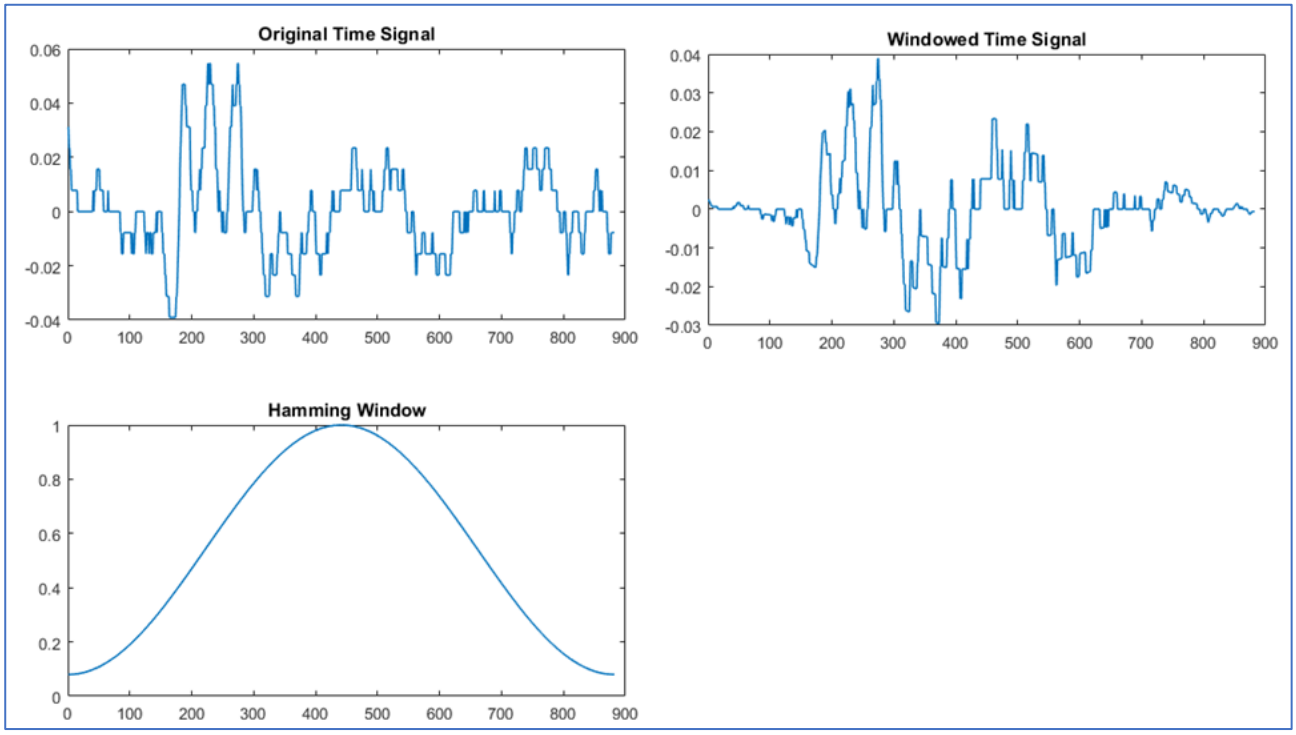


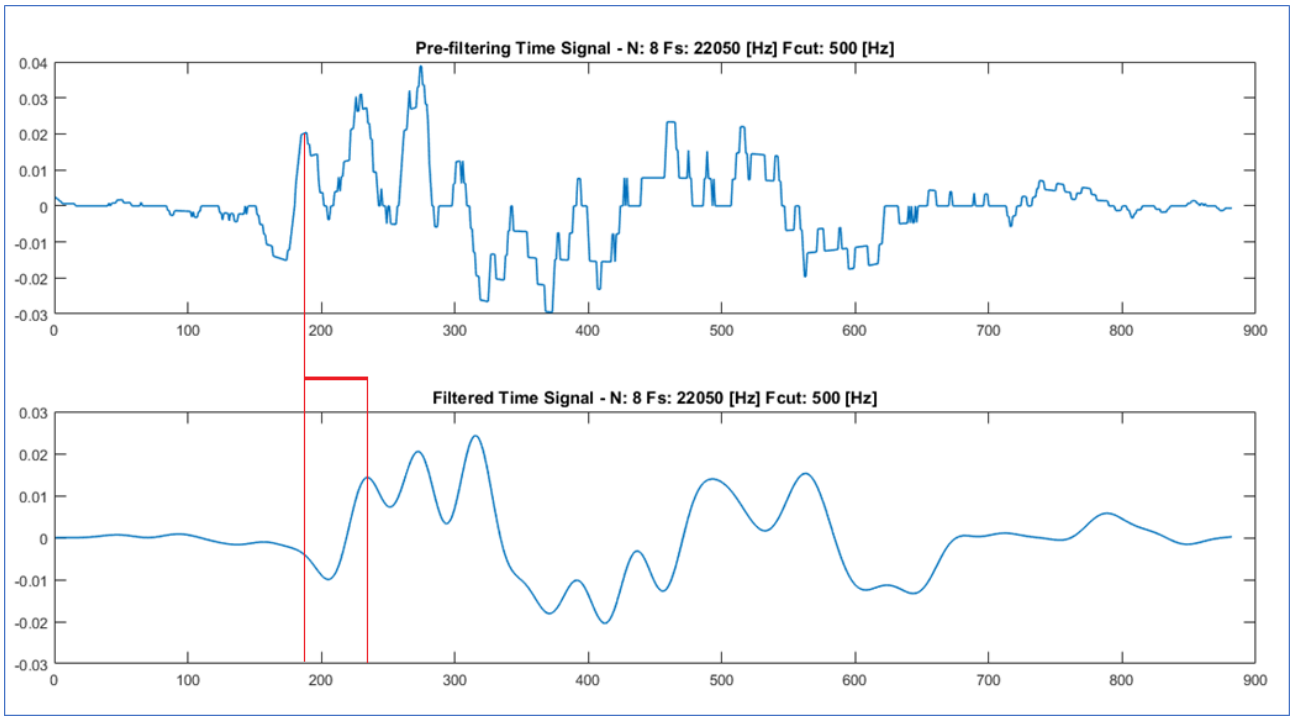

```
%Se volessimo sentire l'array audioSamples
%sound(audioSamples,fs);

%% Procedura passo-passo

%adesso eseguiamo il framing
overlapPercentage=0.5;
frames = toFrames(audioSamples,fs,overlapPercentage);

%visualizziamo i frame della riga frameRow
figure
frameRow = 187;
plot(frames(frameRow,:));
title (['Frame Row: ' num2str(frameRow)]);
    %plot di riga frameRow, tutte le colonne (fig_2)
```




```
%adesso eseguiamo il windowing
windowedFrame = doWindowing(frames(frameRow,:), true);
    %lo eseguo su riga frameRow, tutte le colonne
```

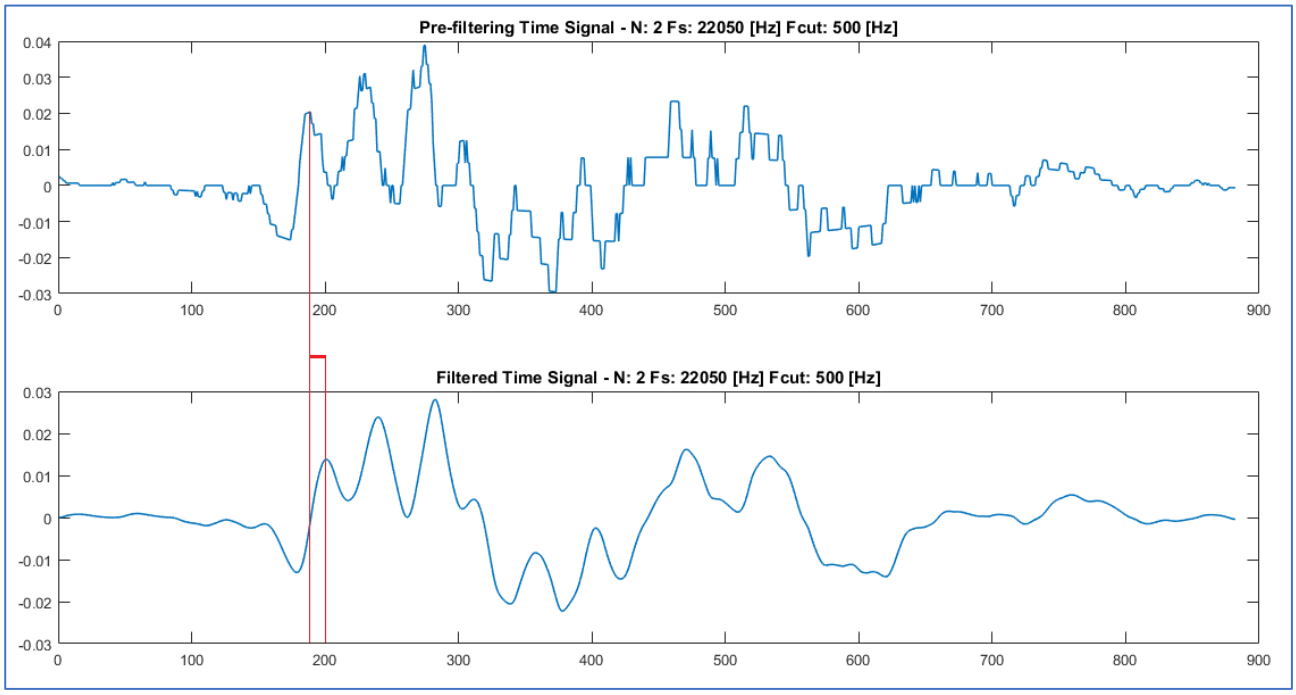
```
%eseguo il filtraggio
fcut = 500;
filteredFrame = doFilter(windowedFrame, 8, fs, fcut, true);
```

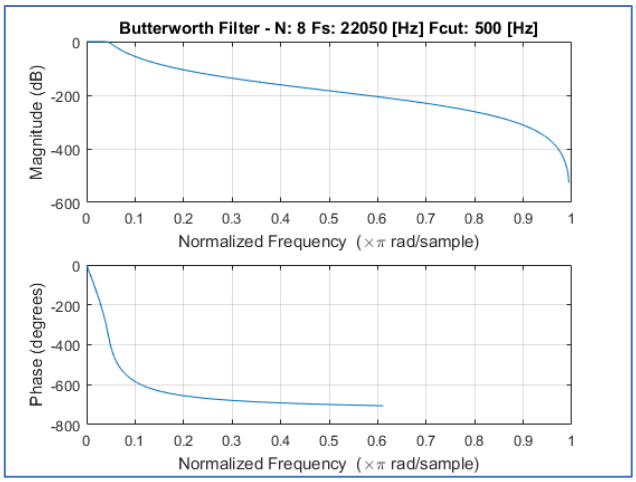


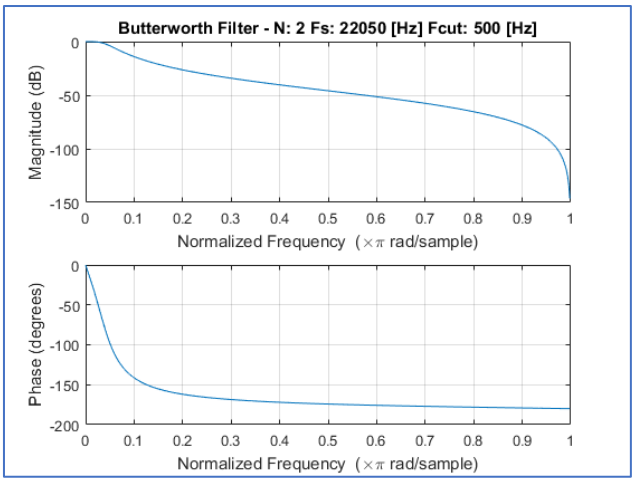


Il segnale è sicuramente pulito, ma con N=8 lo sfasamento è significativo!
Molto meglio usare ordine N=2, anche se leggermente più sporco mi da molto meno sfasamento

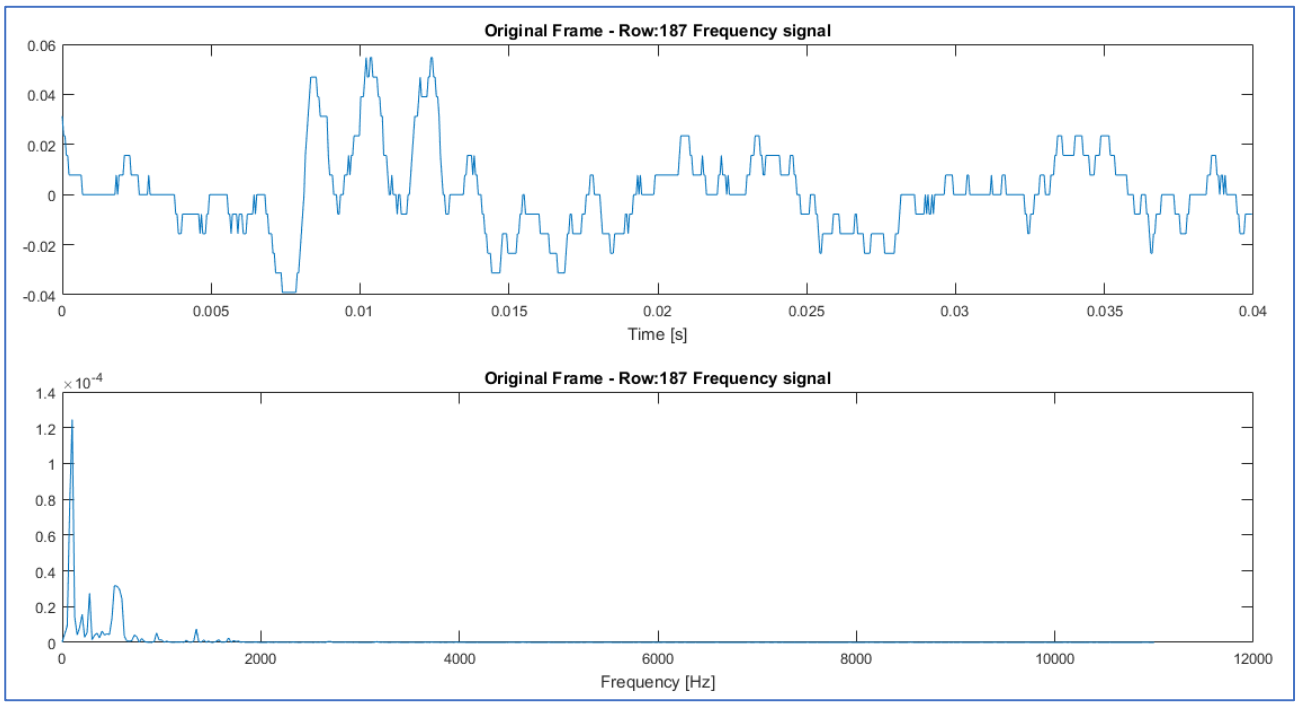
```
filteredFrame = doFilter(windowedFrame, 2, fs, fcut, true);
```



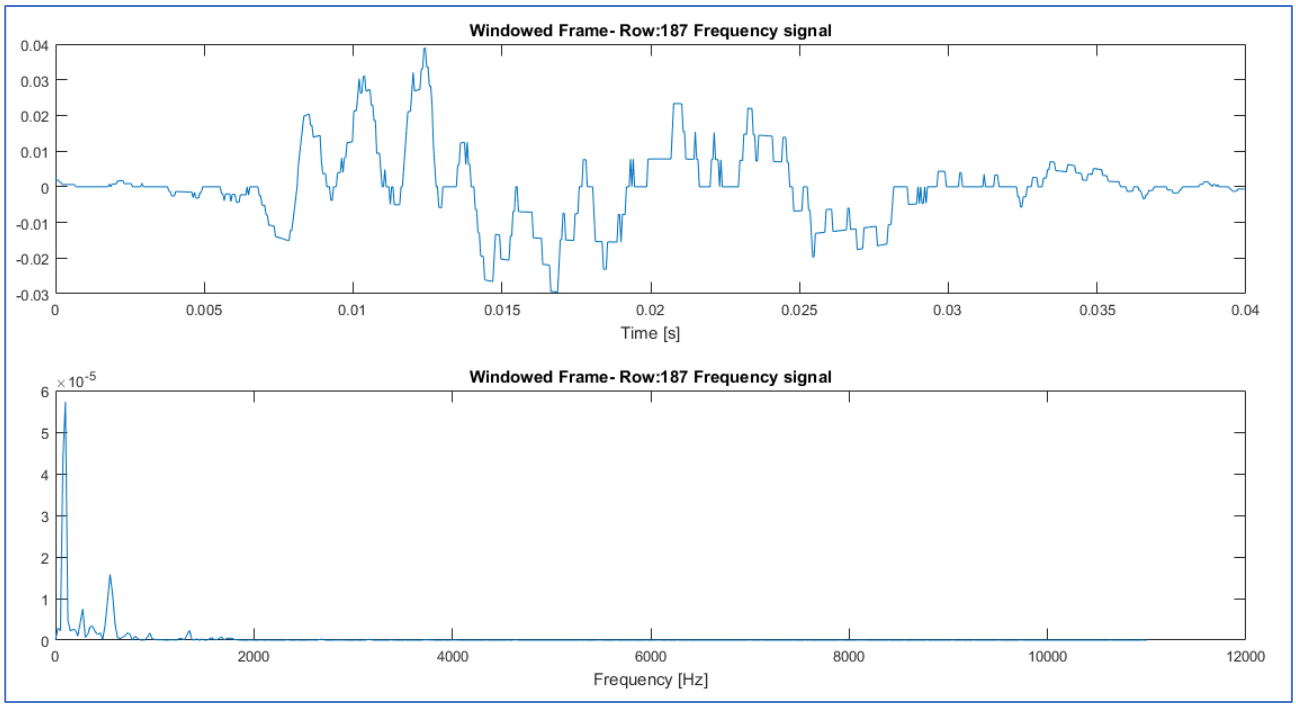




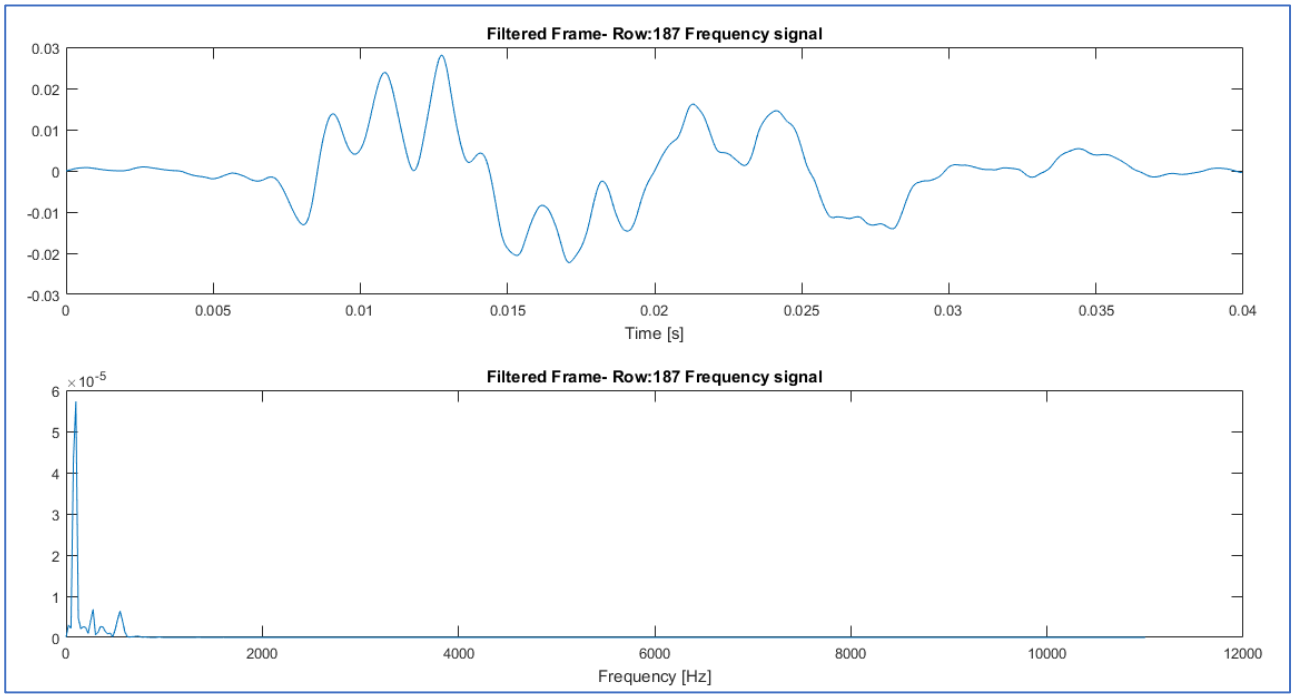

```
%calcolo il power spectrum del frame originale
[ps,f] = powerSpectrum(frames (frameRow, :), fs , true, ['Original Frame - Row:'
num2str(frameRow)]);
```



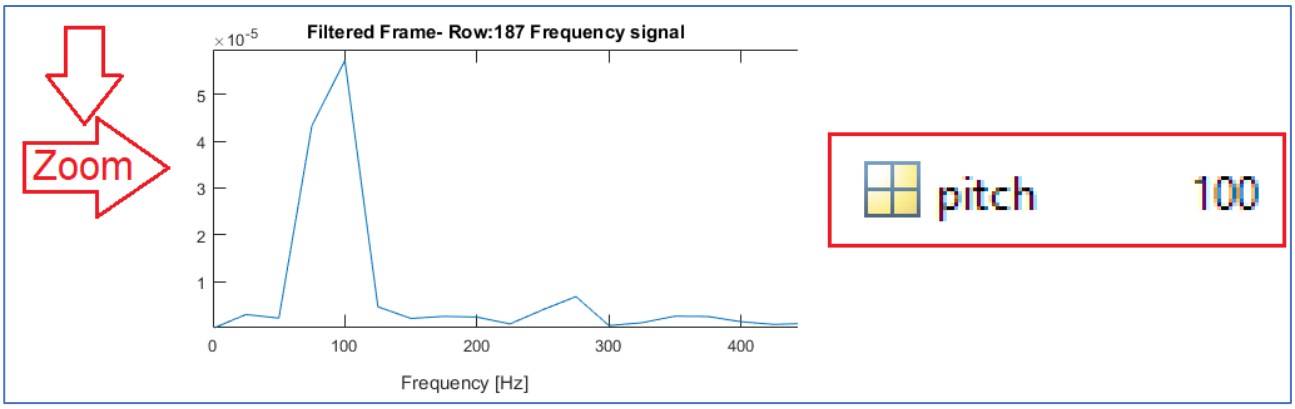

```
%calcolo il power spectrum del frame finestrato non filtrato
[ps,f] = powerSpectrum(windowedFrame , fs , true, ['Windowed Frame- Row:'
num2str(frameRow)]);
```



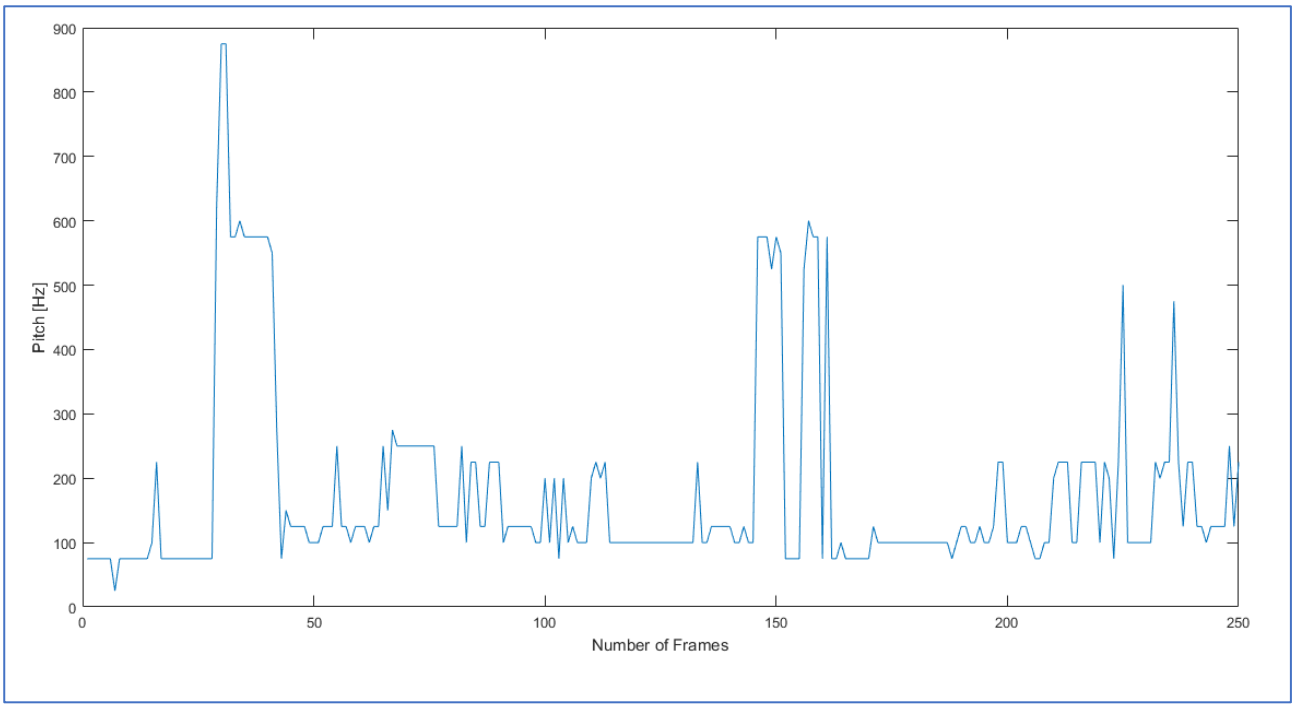

```
%calcolo il power spectrum del frame filtrato
[ps,f] = powerSpectrum(filteredFrame , fs , true, ['Filtered Frame- Row:'
num2str(frameRow)]);
```




```
%calcolo quindi il pitch
pitch = estimatePitch(filteredFrame,fs);
```




```
%% Funzione completa

processData(audioSamples,fs);
```




```
Command Window
  Average Pitch:177.4 Hz
```

8. Voice Activity Detector

Introduzione teorica

ETSI - Lezione 05/06/2018

Se c'è molto rumore non riusciamo a risolvere il problema banalmente filtrando. Due soluzioni:

- Filtro adattato sulle bande del rumore
- Fare una selezione dei frame che tenga solo quelli in cui c'è davvero voce → VAD (Voice Activity Detector)

Il VAD si basa sulla stima del pitch e su altre cose. Di VAD in letteratura ce ne sono tantissimi. Il primo ed il più semplice è un VAD basato sull'energia ("se c'è energia c'è voce"). È un VAD "poco furbo", si può raffinare andando a vedere *dove* è questa energia: il grosso dell'energia vocale è attorno a 1KHz, allora possiamo vedere quanta energia del segnale è in questo intorno. È quello che faremo oggi

In processData.m inseriremo il VAD:

```
for i=1 : numFrames
    %estraggo un frame
    singleFrame = frames (i, :);

    QUI INSERIREMO IL VAD

    %adesso eseguiamo il windowing
    frameWindowed = doWindowing(singleFrame, false);
    %eseguo il filtraggio
    frameFiltered = doFilter(frameWindowed, 2, fs, fcut, false);
    %calcolo il pitch
    pitches(i) = estimatePitch(frameFiltered, fs);
end
```

Per la **Formula di Friis** lo stadio più cruciale per il rumore è il primo, perciò il primo blocco non dovrà introdurre rumore!

Oltre alla funzione per il VAD, creiamo anche una funziona che valuta l'energia nelle sottobande chiamata `energySubBands`.

File: energySubBands.m

N.B: vengono sfruttati tutte le funzioni del paragrafo precedente (*6. Elaborazione del Segnale – Frequenza di Pitch*) ad eccezione delle versioni "2,0" ecc…!!!!

```
function energies = energySubBands(inputSignal, fs, bandWidth, startFreq,
numberOfSubBands)

    %calcoliamo lo spettro (false->no grafico, 'Signal'-> nome segnale)
    [ps,f] = powerSpectrum (inputSignal, fs, false, 'Signal');

    energies = zeros(1,numberOfSubBands); %vettore colonna pre-allocato
```

```
    for i=1 : numberOfSubBands
        %Devo capire da quale campione devo partire (la f potrebbe essere
        %  a 0 come no)
        %Devo capire quali indici usare: quello più basso sarà di
        %  startFreq, ma io voglio che parta da quel campione ed arrivi fino
        %  in fondo
        %Per fare ciò io uso la variabile f precedentemente identificata da
        %  PowerSpectrum, perchè se mi hanno dato, ad esempio, una startFreq
        %  di 300Hz, voglio che il mio valore di f sia >=300

        freq = startFreq + ((i-1)*bandWidth);
        %freq è quindi il valore di partenza

        %find: Find indices of nonzero elements.
        %    I = find(X) returns the linear indices corresponding to
        %    the nonzero entries of the array X.  X may be a logical expression.
        %    Use IND2SUB(SIZE(X),I) to calculate multiple subscripts from
        %    the linear indices I.
        %
        %    I = find(X,K) returns at most the first K indices corresponding to
        %    the nonzero entries of the array X.  K must be a positive integer,
        %    but can be of any numeric type.

        %vogliamo quindi l'indice basso dal quale partire
        % mettiamo uno per identificare il primo indice frequenziale > freq
        idx_low = find (f>= freq,1);
        %e quello alto
        idx_high = find (f>= freq + bandWidth,1);

        %il power spectrum ha già i campioni al quadrato
        % devo quindi solo fare la somma
        energies (1,i) = sum( ps(idx_low : idx_high) );
    end

    figure
    plot (energies);
    xlabel('Number of subbands');
    ylabel ('Energy');
end
```

File: doVAD.m

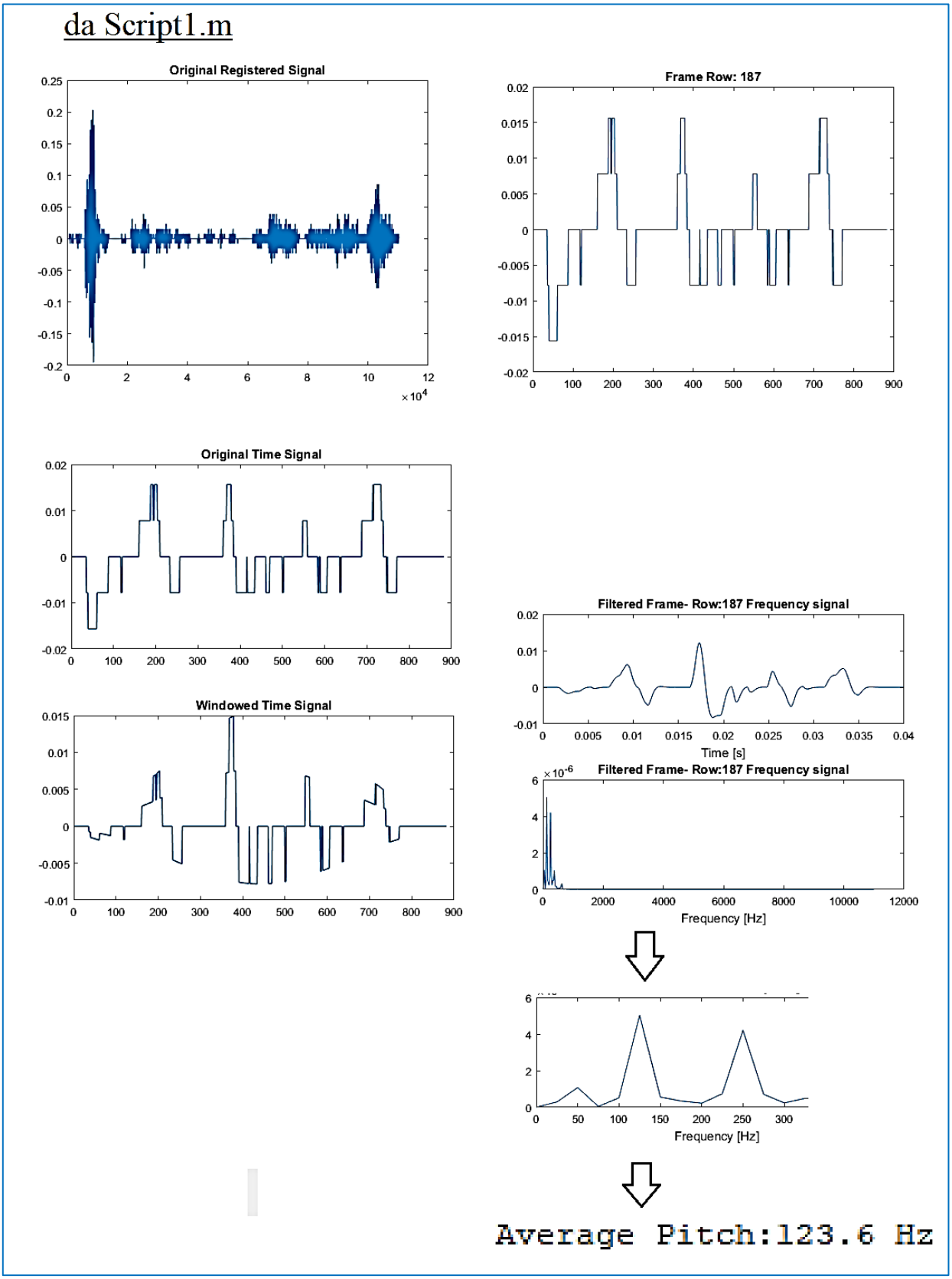
```
function [res, ratio] = doVAD (inputSignal,fs,fmax,thr)
    %(thr è la soglia, "threshold", sopra la quale il frame è vocale)

    %calcoliamo lo spettro (false->no grafico, 'Signal'-> nome segnale)
    [ps,f] = powerSpectrum (inputSignal, fs, false, 'Signal');

     idx = find (f>= fmax,1);
     energyFmax = sum(ps(1 : idx)); %qui dentro ho l'energia fino a fmax
     energyTotal = sum(ps);

     ratio = energyFmax/energyTotal; %ritorno questo come output della funzione

     %imposto anche il result (res)
     res = false; %valore di default
     if (ratio >= thr)
         res = true; %se supera la soglia è un segnale vocale
     end
end
```

File: processData.m (2.0)

```
function pitches = processData (audioSamples,fs)  %questa funzione fa tutto

    fcut = 500;    %frequenza di taglio del LPF

    %Dividi in frame
    overlapPercentage = 0.50; % overlap del 50%
    frames = toFrames(audioSamples,fs,overlapPercentage);

    [numFrames, numSamples] = size (frames);

    %alloco prima l'array (aumento l'efficenza)
    pitches = zeros (1, numFrames);

    for i=1 : numFrames
        %estraggo un frame
        singleFrame = frames (i, :);

        %verifichiamo l'energia (usiamo il VAD)
        fmax = 1000; % 1 KHz
        thr = 0.95; % Soglia al 95%
        [res ratio(i)] = doVAD(singleFrame, fs, fmax, thr);
        %res è un booleano che mi dice se tenere o no il frame la ratio è una
            informazione non molto utile (ma volendo possiamo graficarla)

        %uso quindi res per capire se tenere o no il frame:
        if (res)
            %adesso eseguiamo il windowing
            frameWindowed = doWindowing(singleFrame, false);
            %eseguo il filtraggio
            frameFiltered = doFilter(frameWindowed, 2, fs, fcut, false);
            %calcolo il pitch
            pitches(i) = estimatePitch(frameFiltered, fs);
        end
    end

    %grafico la ratio
    figure;
    plot (ratio)
    xlabel('Frames');
    ylabel(['Ratio, fmax = ' num2str(fmax)]);

    %visualizzo tutti i pitch
    figure
    plot (pitches)
    xlabel('Number of Frames');
    ylabel('Pitch [Hz]');

    %visualizzo il valor medio di pitch
    disp(['Average Pitch:' num2str(mean(pitches)) ' Hz']);

end
```

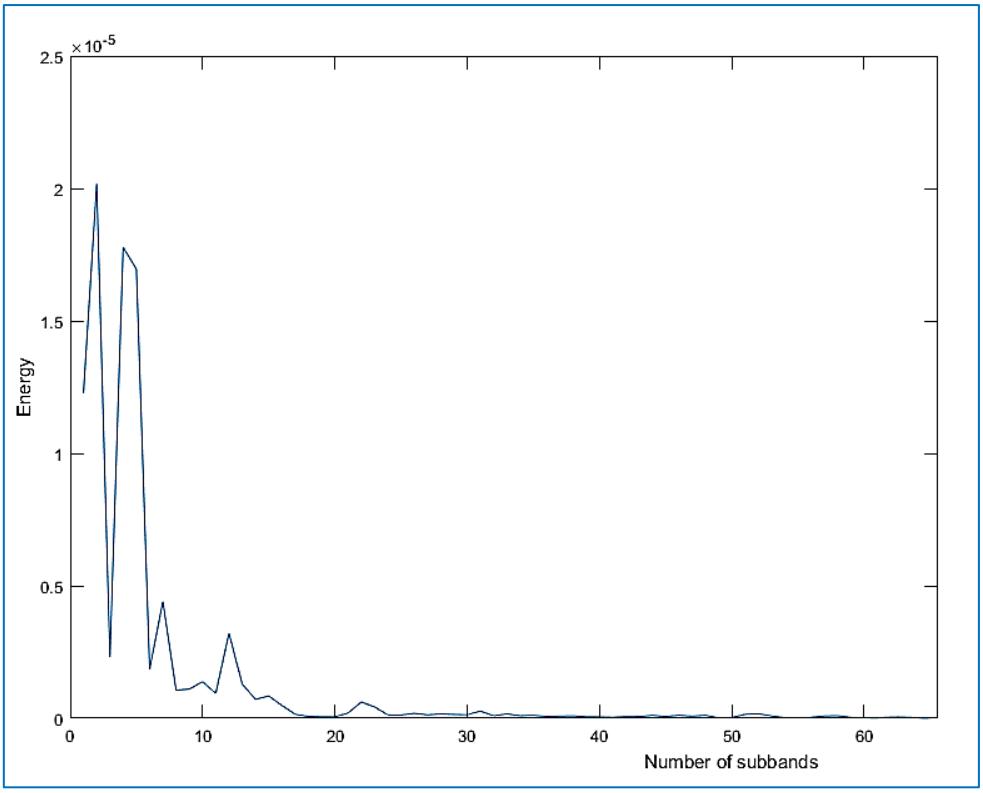
File: script2.m

```
%eseguo il primo script
script1;
```



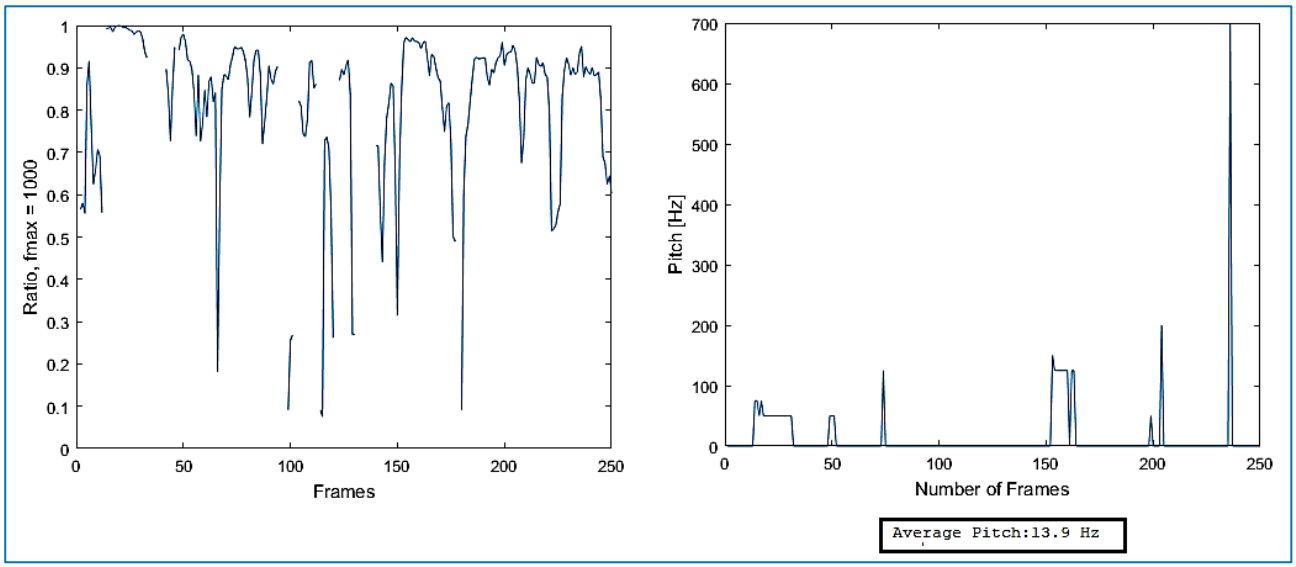
Adesso che abbiamo visto come script1.m eseguiva l'analisi di questo segnale, eseguiamo script2.m e confrontiamo il risultato:

```
startFreq = 50;
bandWidth = 50;
numberOfSubBands = 100;
energies = energySubBands(frames(187,:), fs, bandWidth, startFreq,
numberOfSubBands);
```



Questo si vede che non è un frame vocale perché ci sono due picchi (?)

```
processData(audioSamples,fs);
```



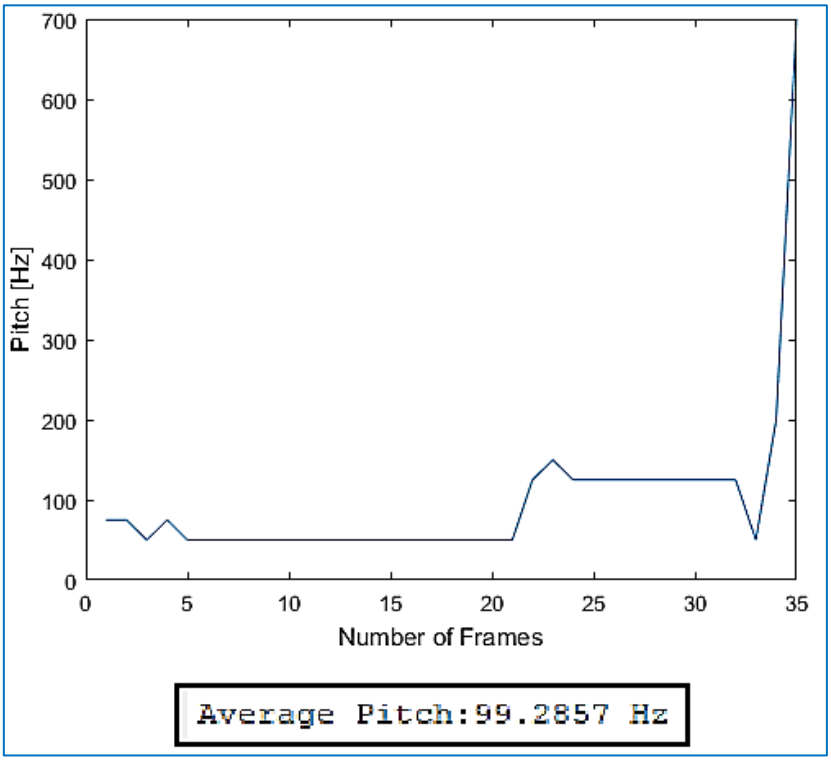
Il pitch medio è bassissimo perché ci sono un sacco di valori a zero, dobbiamo migliorare il calcolo della media per eliminare questo problema. Lo vedo nel nuovo file ↓↓↓

File: processData.m (3.0)

```
function pitches = processData (audioSamples,fs)
    %questa funzione fa tutto

    fcut = 500; % frequenza di taglio del LPF

    %Dividi in frame
    overlapPercentage = 0.50; % overlap del 50%
    frames = toFrames(audioSamples,fs,overlapPercentage);

    [numFrames, numSamples] = size (frames);

       %alloco prima l'array (aumento l'efficenza)
       %pitches = zeros (1, numFrames);
    %Ma %FIX: evito il problema dei pitch nulli
    pitches = [];

    idxPitch=1; %FIX: evito il problema dei pitch nulli
    for i=1 : numFrames
        %estraggo un frame
        singleFrame = frames (i, :);

        %verifichiamo l'energia (usiamo il VAD)
        fmax = 1000; % 1 KHz
        thr = 0.95; % Soglia al 95%
        [res ratio(i)] = doVAD(singleFrame, fs, fmax, thr);
        %res è un booleano che mi dice se tenere o no il frame
        %la ratio è una informazione non molto utile
        %    (ma volendo possiamo graficarla)

        %uso quindi res per capire se tenere o no il frame:
        if (res)
            %adesso eseguiamo il windowing
            frameWindowed = doWindowing(singleFrame, false);
            %eseguo il filtraggio
            frameFiltered = doFilter(frameWindowed, 2, fs, fcut, false);
            %calcolo il pitch
            %FIX: evito il problema dei pitch nulli
            pitches(idxPitch) = estimatePitch(frameFiltered, fs);
            idxPitch = idxPitch + 1;
        end
    end

    %grafico la ratio
    figure;
    plot (ratio)
    xlabel('Frames');
    ylabel(['Ratio, fmax = ' num2str(fmax)]);

    %visualizzo tutti i pitch
    figure
    plot (pitches)
    xlabel('Number of Frames');
    ylabel('Pitch [Hz]');

    %visualizzo il valor medio di pitch
    disp(['Average Pitch:' num2str(mean(pitches)) ' Hz']);

end
```

File: script2.m (eseguito nuovamente)

```
%eseguo il primo script
%script1.m

startFreq = 50;
bandWidth = 50;
numberOfSubBands = 100;
energies = energySubBands(frames(187,:), fs, bandWidth, startFreq,
numberOfSubBands);

processData(audioSamples,fs);
```



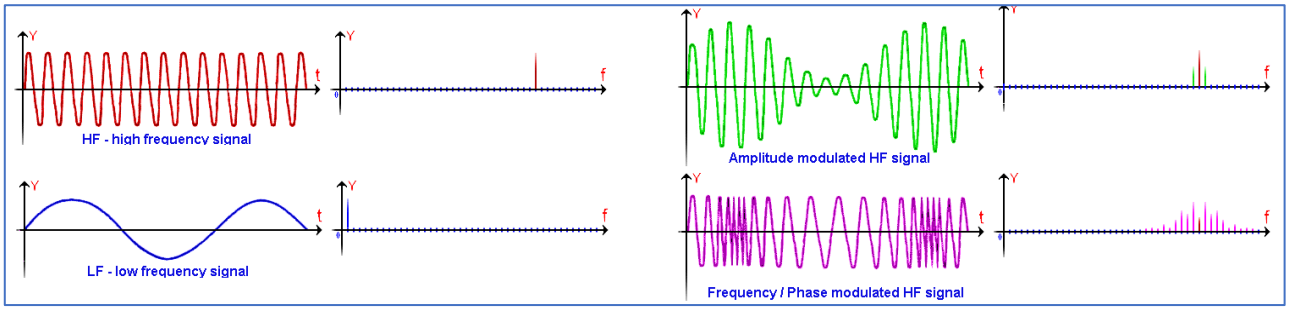
Adesso non ho più valori di pitch nulli, la media è di nuovo corretta

9. Analisi Cepstrum

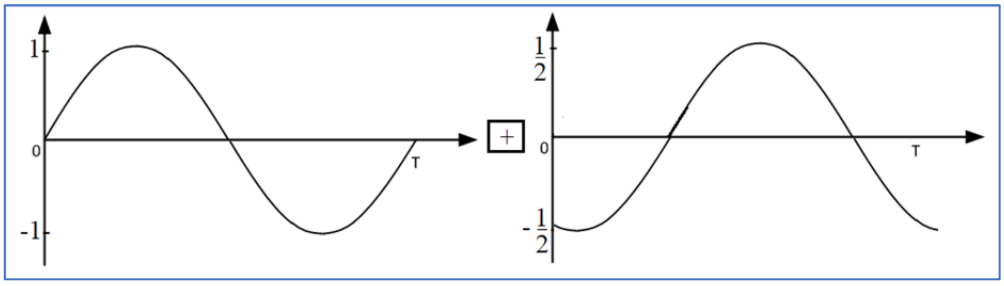
Introduzione teorica

ETSI - Lezione 07/06/2018

Spesso nel tempo non è molto intuitivo capire che i segnali in questioni sono dovuti ad una somma di diverse componenti:



L'analisi di Fourier ci permette una scomposizione frequenziale che mette in luce quello che l'analisi temporale non riesce a fare. Tenendo conto di questo, c'è un momento in cui l'analisi di Fourier non è che "fallisca", ma in cui questa non ci viene troppo in aiuto (non visualizza proprio bene questa scomposizione frequenziale). È il caso di due segnali dalla stessa frequenza sommati insieme:

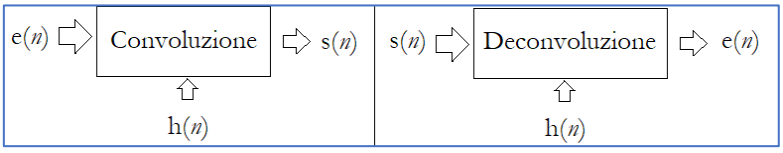


La somma di questi due segnali sarà, nel dominio della frequenza, un unico picco, avendo la stessa frequenza. Ma ciò non mi è molto utile. Come fare? Applico l'analisi di Fourier due volte, perché Fourier mette in evidenza la periodicità, e se farlo una volta mi evidenzia la periodicità nel tempo (evidenziando le frequenza), farlo due volte mi mette in evidenza la periodicità frequenziale. Questo è alla base della analisi cepstrale. Vediamolo. Prendiamo un segnale con rumore $q(n)$:

$$x(n) = x_1(n) \underbrace{+}_{(!!!)} q(n) \Rightarrow F[x(n)] = F[x_1(n) + q(n)] = F[x_1(n)] + \underbrace{F[q(n)]}_{\substack{\text{Filtraggio} \\ \text{Ideale} \rightarrow \approx 0}} \simeq F[x_1(n)]$$

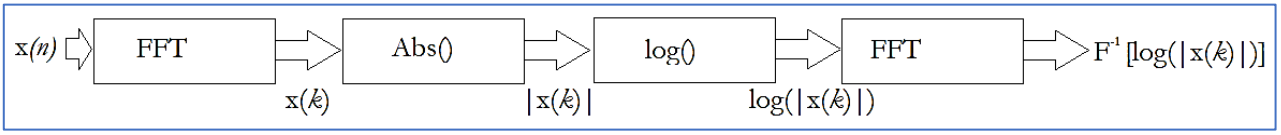
Il fatto che il rumore sia additivo (**!!!**) è una ipotesi fortissima e per nulla scontata. Infatti se spesso si modella il segnale vocale:

$$s(n) = \underbrace{e(n)}_{\substack{\text{Flusso} \\ \text{di aria}}} * \underbrace{h(n)}_{\substack{\text{Tratto} \\ \text{vocale}}} \quad \rightarrow \quad F[e(n)] + F[h(n)]$$



Questo si chiama **filtraggio omomorfico** (*homomorphic filtering*). La famiglia di filtri omomorfici fa questa cosa qui, questi filtri cercano di ricondurre segnali del tipo $s(n) = e(n) * h(n)$ in segnali del tipo $F[e(n)] + F[h(n)]$, ovvero **segnali additivi**.

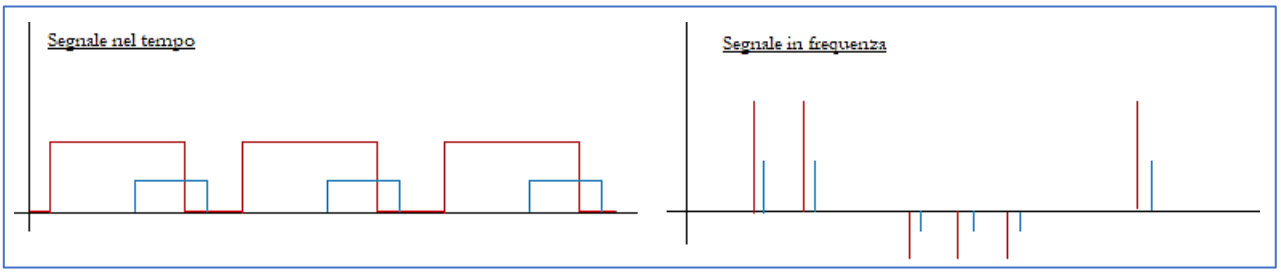
$$s(n) = e(n) * h(n) \underset{DFT}{=} E(f) \cdot H(f) \underset{log}{=} \log E(f) + \log H(f) \underset{DFT}{=} F[\log E(f)] + F[\log H(f)]$$



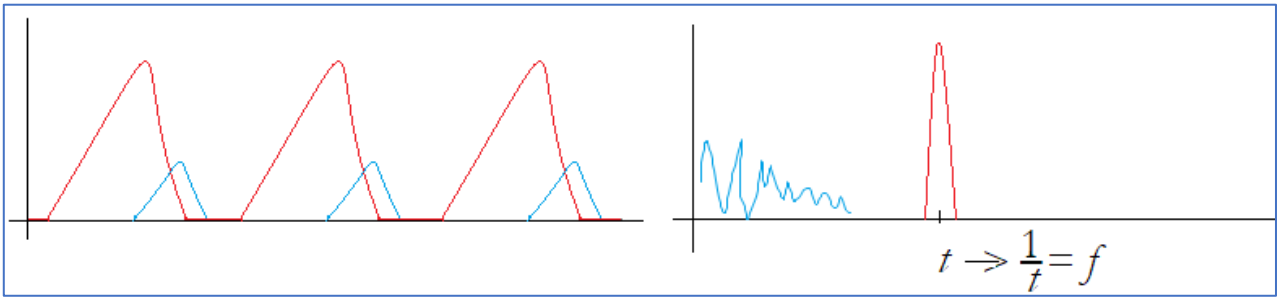
$$c(k) = \frac{1}{N}\sum_{n=0}^{N-1} \log|X(n)|e^{\frac{j2\pi kn}{N}}$$

k è un indice temporale! MA non è proprio un indice temporale puro → ***quefrency***. Infatti, ***cepstrum*** è l'inversione di *spectrum*.

Nel caso dell'**eco**, in cui ho delle repliche attenuate ritardate, si forma una situazione del genere come analisi in frequenza (nell'esempio ho una serie di onde rettangolari):



Nel caso della voce al posto dei rettangoli avremo delle formanti, ed ogni formante – a causa dell'eco – ha delle ***sidebands***, e me ne accorgo dallo spettro, ma visivamente non è di nuovo molto chiaro, mentre facendo un *cepstrum*, come vedremo su MATLAB, avremo un picco forte, che ci darò una informazione sui periodi frequenziali:

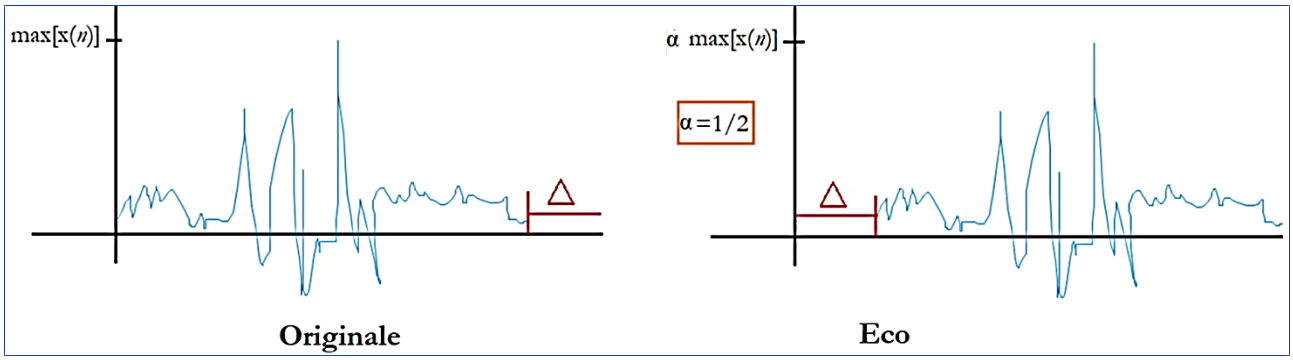


Essendo la *quefrency* un dominio in qualche modo temporale, se io prendo il valore di *quefrency* del picco (**MFCC** = *Medium Frequency Cepstrum Coefficient*) e ne faccio il reciproco, questo valore di frequenza indica la periodicità del mio spettro ($1/t = f$).

Usiamo il sample `load mtlb` come file audio. Dovremo modellare un fenomeno di eco del tipo:

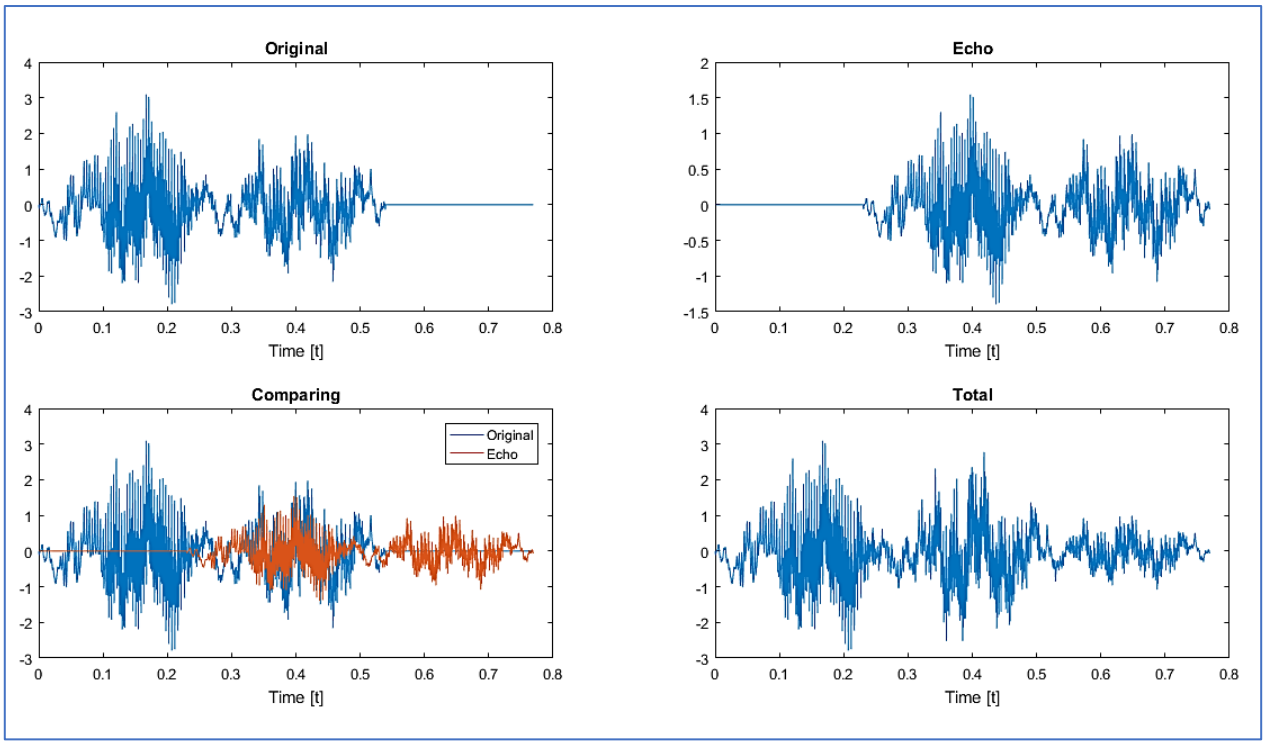
$$y(n) = x(n) + Ax(n - N)$$

Lo facciamo nel file script `cepstrumAnalysis.m`.

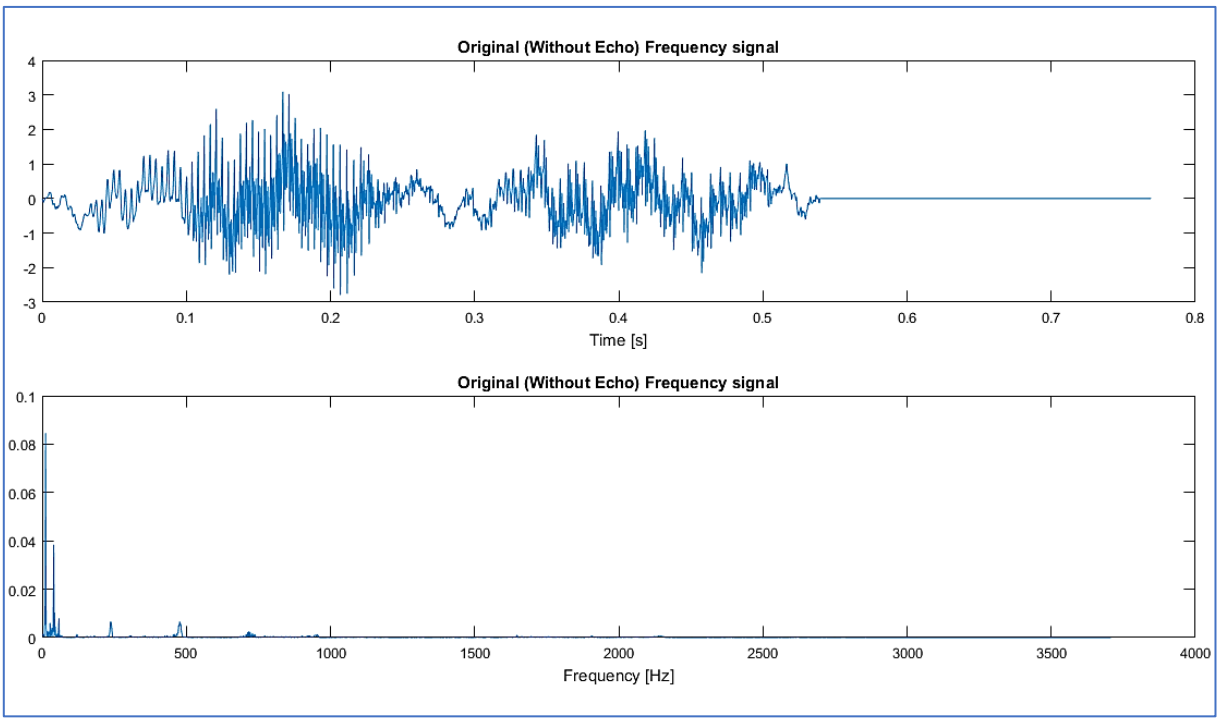


File: cepstrumAnalysis.m

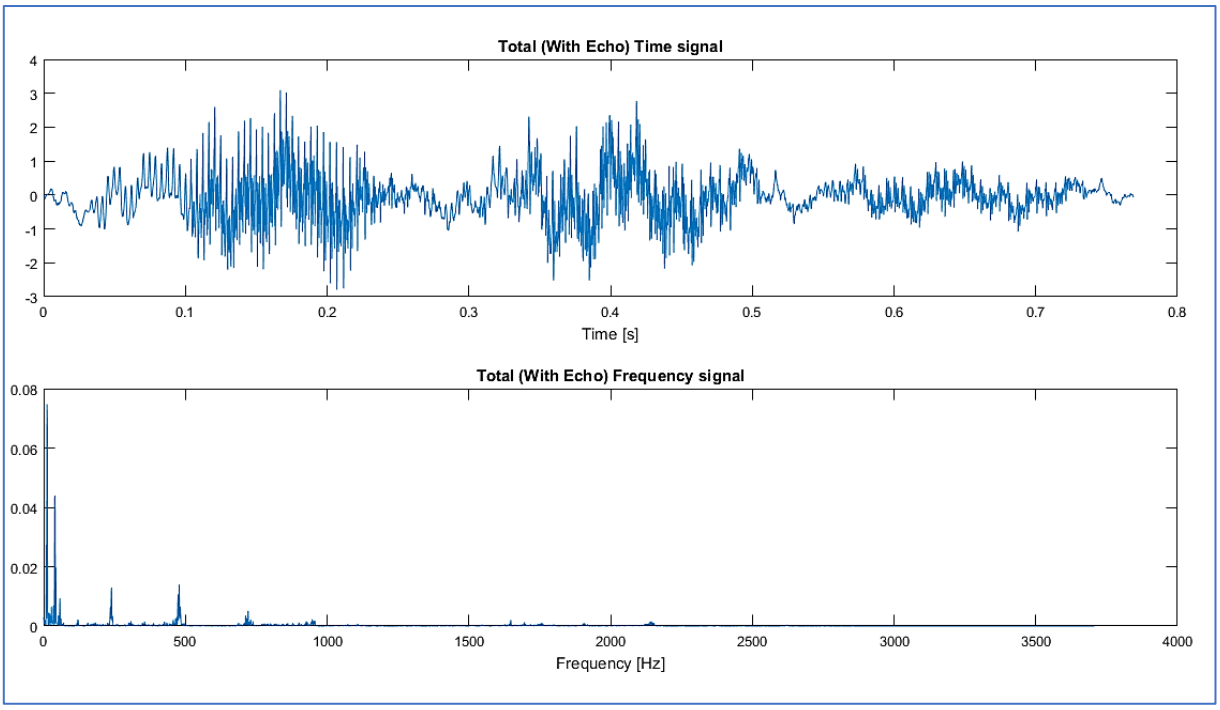
```
%% Signal Definition

clc;
close all;

load mtlb; %è un file di default di matlab che usiamo come sample audio
%sound(mtlb)

%Dovremo modellare un fenomeno di eco y(n)=x(n)+Ax(n-N)
delay = 0.23;
alpha = 0.5;
delta = round(Fs*delay);

%definisco il segnale originale
orig=[mtlb; zeros(delta,1)];
%definisco il segnale dell'eco
echo = alpha*[zeros(delta,1); mtlb];
%definisco il segnale totale
mtEcho = orig + echo;
%sound(mtEcho)

%genero l'asse dei tempi
t = (0 : length(mtEcho)-1)/Fs;

figure; %fig.1
  %segnale originale
    subplot (2,2,1);
    plot(t, orig);
    xlabel('Time [t]');
    title('Original');
  %eco
    subplot (2,2,2);
    plot(t, echo);
    xlabel('Time [t]');
    title('Echo');
  %confronto
    subplot (2,2,3);
    plot(t, [orig,echo]);
    legend('Original','Echo');
    xlabel('Time [t]');
    title('Comparing');
  %segnale totale
    subplot (2,2,4);
    plot(t, mtEcho);
    xlabel('Time [t]');
    title('Total');
```



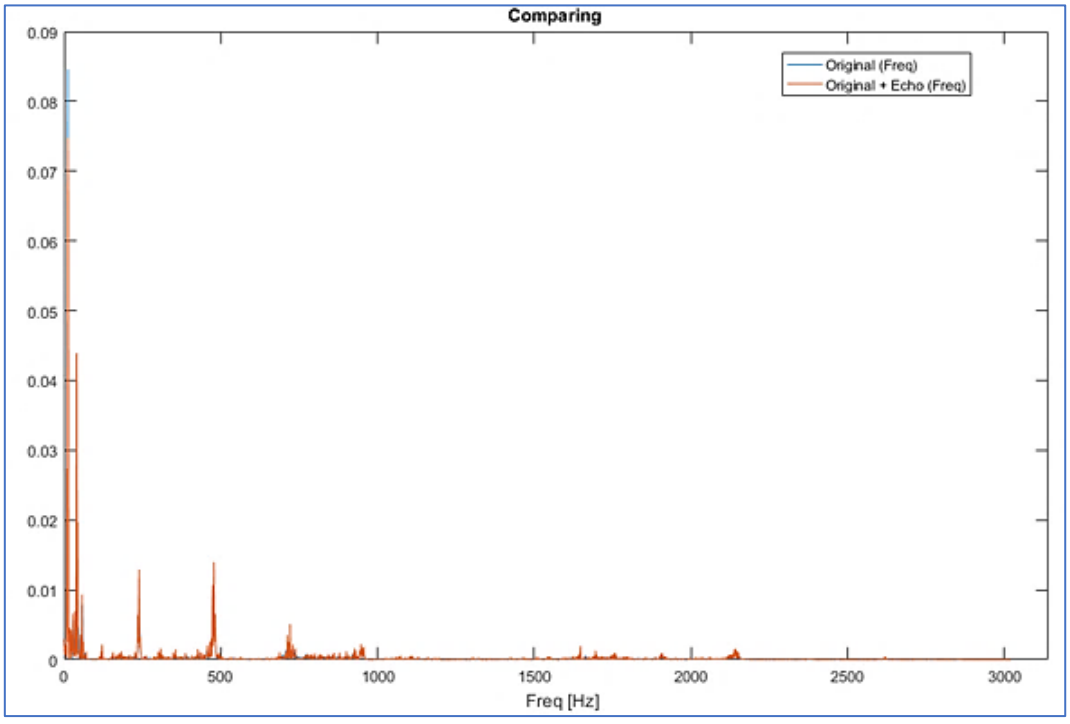

```
%% Fourier Analysis

%Eseguo il power spectrum dell'originale (fig.2)
[psOrig,f] = powerSpectrum(orig',Fs,true,'Original (Without Echo)');
    %dobbiamo passagli il segnale trasposto!!!
```

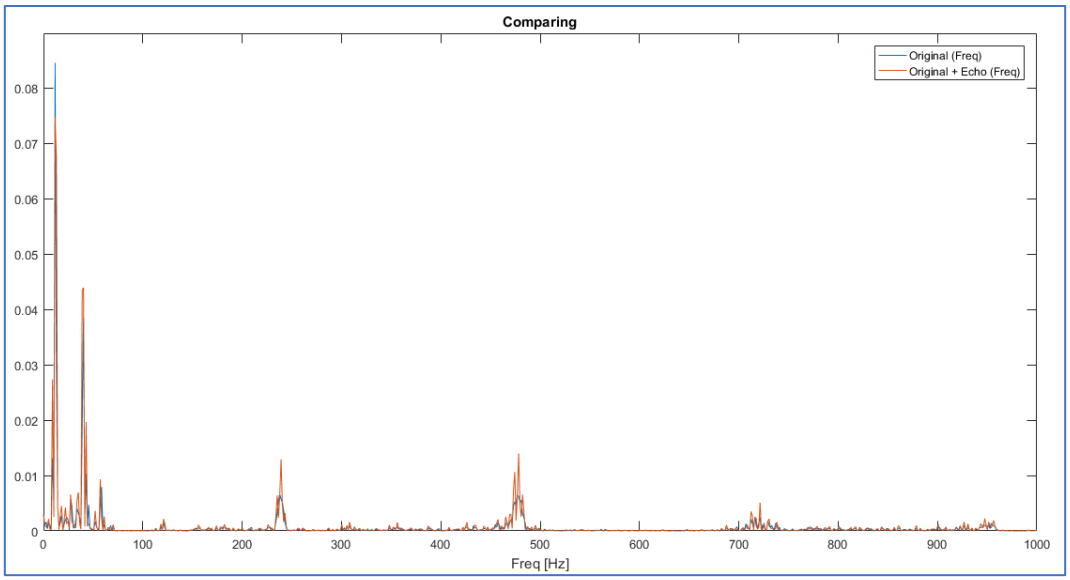


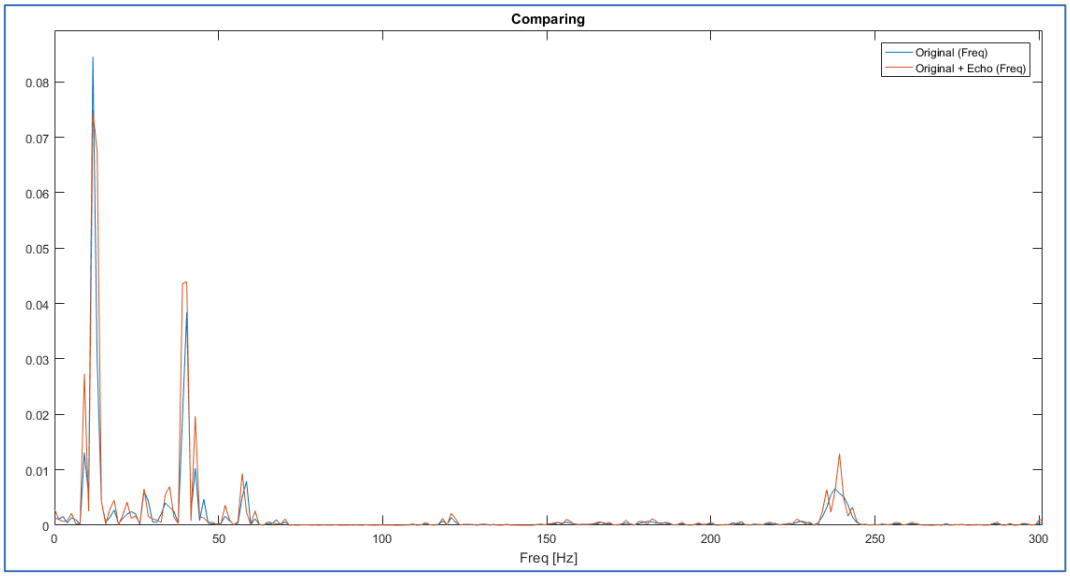

```
%Eseguo il power spectrum del totale (fig.3)
[psMtEcho,f] = powerSpectrum(mtEcho',Fs,true,'Total (With Echo)');
```

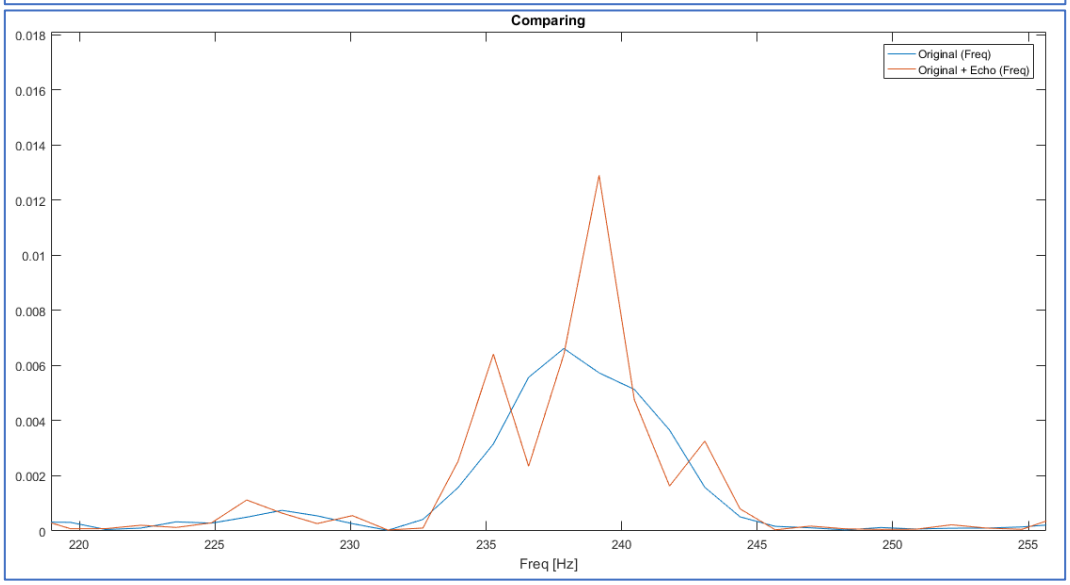


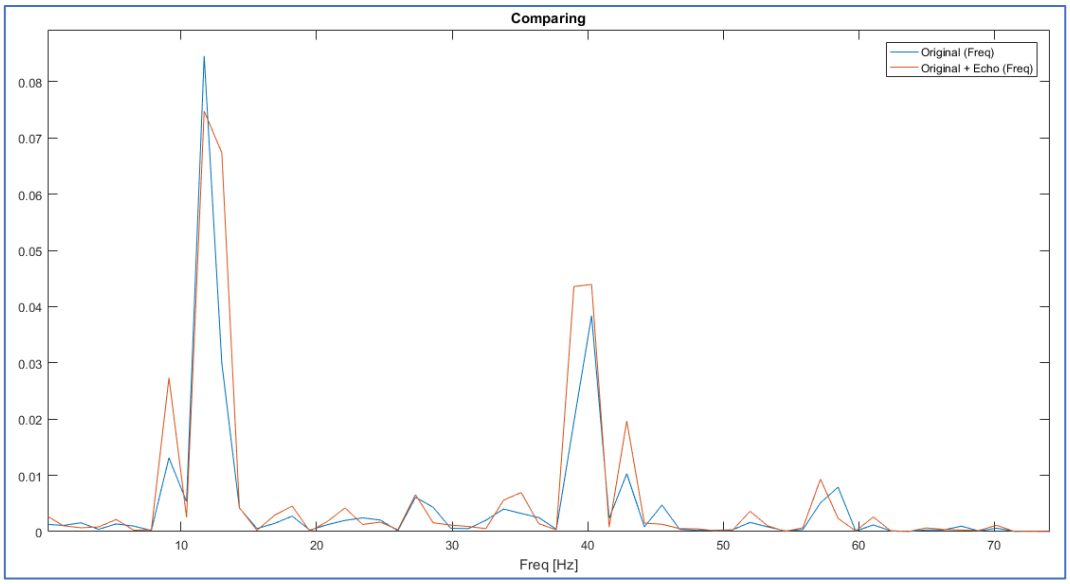

```
figure; %fig.4-8
  %confronto
    plot(f, [psOrig',psMtEcho']); %DEVO TRASPORLI (Perche?)
    legend('Original (Freq)','Original + Echo (Freq)');
    xlabel('Freq [Hz]');
    title('Comparing');
```




Comparing
Original (Freq)
Original + Echo (Freq)
0.08
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0
0
100
200
300
400
500
600
700
800
900
1000
Freq [Hz]

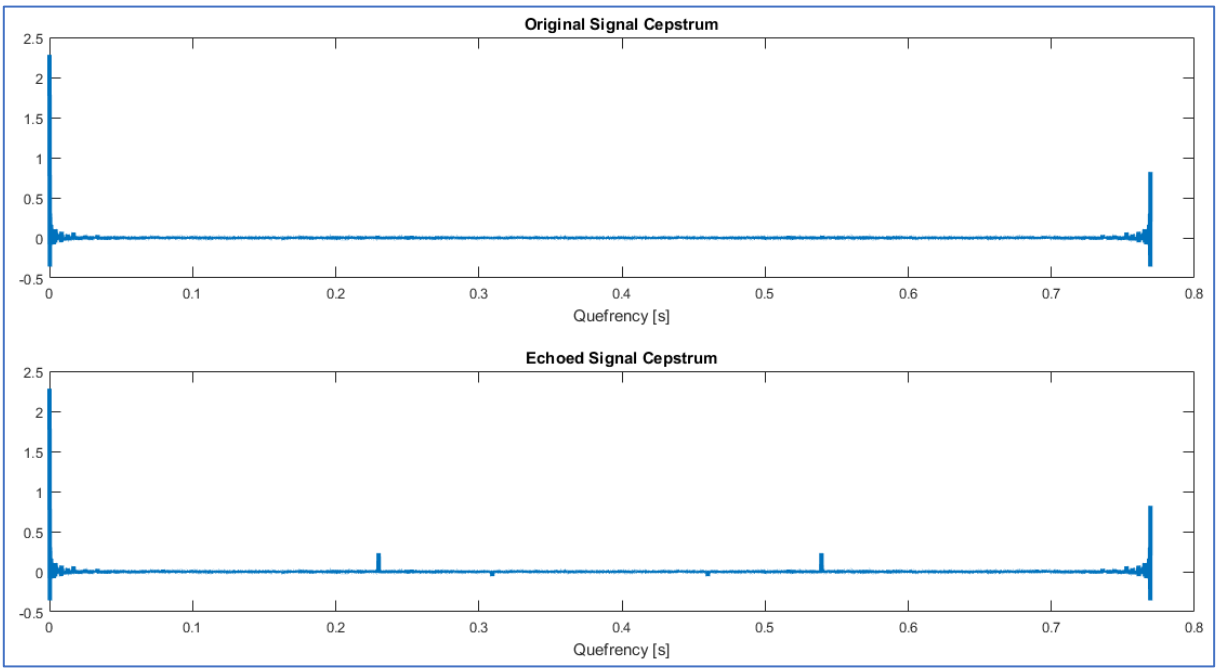

Comparing
Original (Freq)
Original + Echo (Freq)
0.08
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0
0
50
100
150
200
250
300
Freq [Hz]

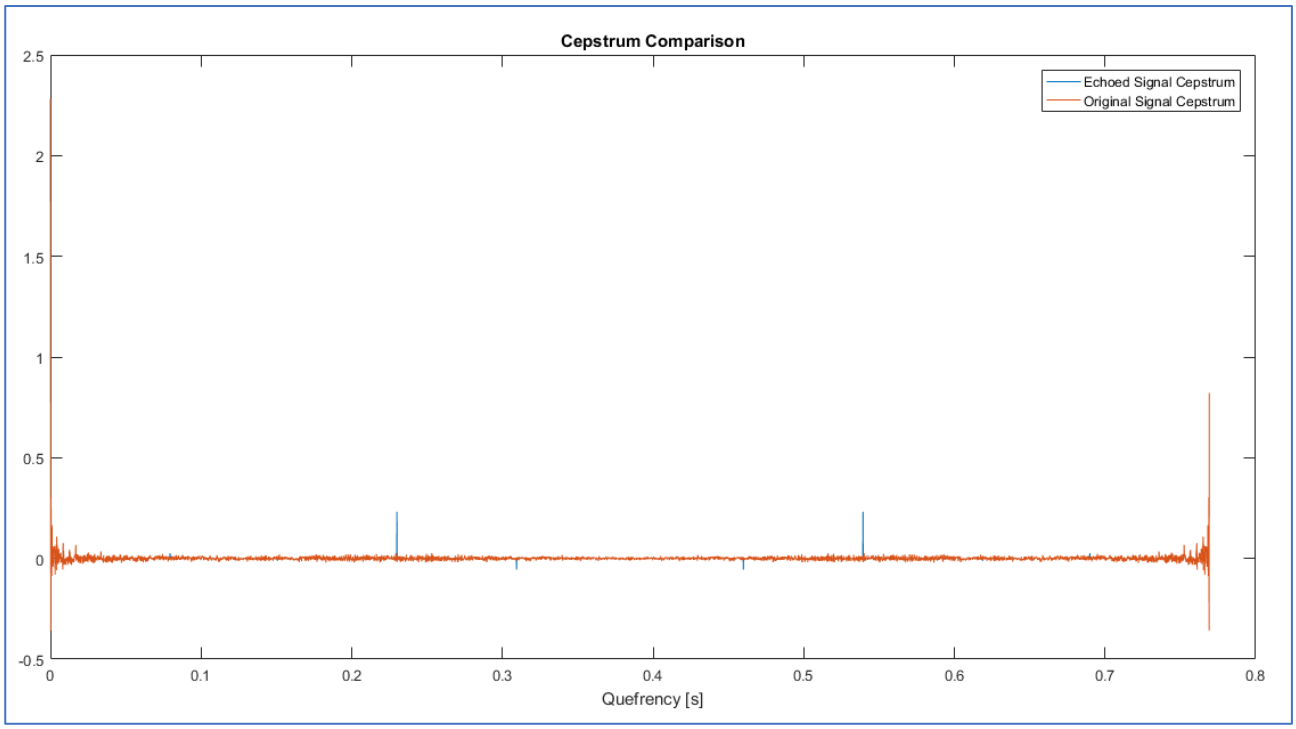

Comparing
Original (Freq)
Original + Echo (Freq)
0.018
0.016
0.014
0.012
0.01
0.008
0.006
0.004
0.002
0
220
225
230
235
240
245
250
255
Freq [Hz]

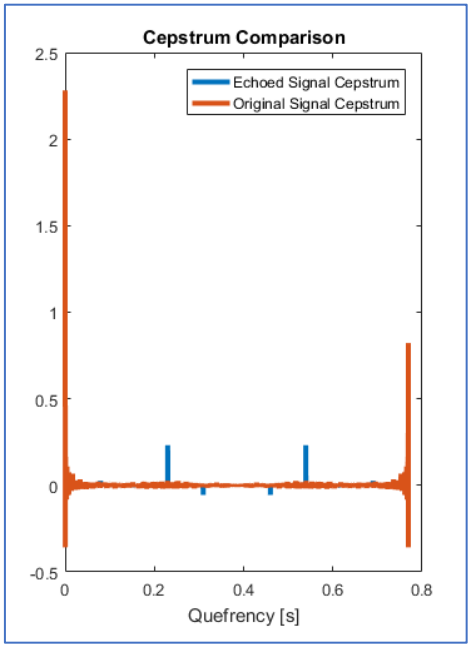


Dalla comparazione o semplicemente dall'osservazione dei *power spectrum* non riusciamo a ricavare nulla di utile. Come vedremo adesso ↓ invece, l'analisi cepstrale ci aiuterà a identificare le repliche dovute al fenomeno di eco.

```
%% Cepstrum Analysis

cOrig = real(ifft(log(abs(fft(orig)))));
%fft -> abs -> log -> ifft -> tolgo le componenti complesse
                              % (il segnale dev'essere reale)
cEcho = real(ifft(log(abs(fft(mtEcho)))));

figure; %fig.9-10
  %Confronto cepstrum (vedo che sono identici!!)
    plot(t, [cEcho, cOrig]);
    xlabel('Quefrency [s]');
    legend('Echoed Signal Cepstrum', 'Original Signal Cepstrum');
    title('Cepstrum Comparison');
```







Dalla comparazione dei *cepstrum* si vede chiaramente come il segnale abbia, nel caso di eco, dei picchi minori, come visto nell'introduzione teorica.

10. Appendice

(Consta di allegati senza continuità con l'elaborato. Auguri.)

Capitolo 3

Filtri Analogici

I filtri ideali sono caratterizzati da funzioni di trasferimento a modulo costante in banda passante, nullo in banda proibita e fase lineare. Poiché tali filtri non sono causali, essi possono essere soltanto "approssimati" da filtri fisicamente realizzabili. Il problema della realizzazione di filtri per una data applicazione non è quindi banale, e richiede almeno tre passi per la sua soluzione:

1. l'individuazione delle *specifiche* del filtro data la particolare applicazione;
2. la determinazione della funzione di trasferimento di un filtro soddisfacente le specifiche individuate;
3. la realizzazione fisica di un sistema la cui funzione di trasferimento coincide con quella determinata.

La determinazione di specifiche sufficientemente precise è il primo passo per ottenere un filtro da adottare per una data applicazione. Al fine di descrivere le specifiche del filtro che si intende realizzare, è necessaria la conoscenza dei parametri che permettano di valutare la qualità dell'approssimazione rispetto a un filtro ideale, quali la *frequenza di taglio a 3 dB*, la *frequenza di stop*, l'ampiezza della *banda di transizione*, l'*attenuazione*, la *linearità della fase*.

L'individuazione del filtro viene poi ottenuta selezionando la funzione di trasferimento di un filtro causale che soddisfa le specifiche assegnate; in questo capitolo ci limitiamo a

richiamare la soluzione di questo problema all'interno delle famiglie di *filtri di Butterworth e di Chebyshev*.

L'ultima fase consiste nella realizzazione fisica del sistema di cui è nota la funzione di trasferimento. Le realizzazioni fisiche possono essere classificate sulla base delle componenti costituenti il sistema: filtri a elementi *RLC passivi*, filtri a elementi *RC attivi*, filtri a *microonde*, filtri a *cristallo*, filtri *elettromeccanici*. In questo capitolo accenniamo alle tecniche di analisi per circuiti RLC passivi, richiamando i principi di Kirkoff, e alla progettazione di filtri di Butterworth mediante circuiti a elementi attivi come gli amplificatori operazionali.

3.1 Caratteristiche dei Filtri Analogici

Nel Capitolo 2 abbiamo introdotto la nozione di filtro ideale e delineato le principali tipologie: filtro passabasso, passaalto e passabanda. In questo paragrafo studiamo la realizzazione pratica di filtri ideali; faremo riferimento esclusivamente a filtri passabasso, poiché le considerazioni riportate di seguito possono essere estese senza difficoltà agli altri casi.

I grafici (modulo e fase) della tipica funzione di trasferimento di un filtro ideale sono riportati in Figura 3.1.

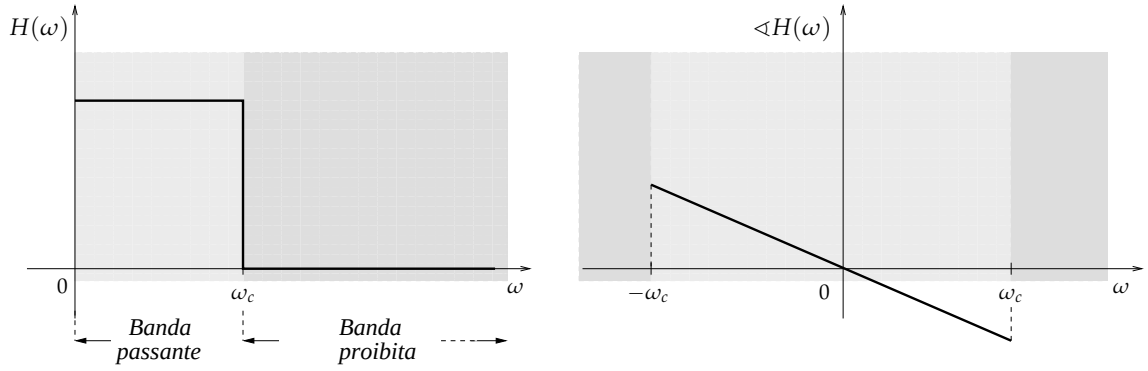


Figura 3.1 Modulo e fase della funzione di trasferimento di un filtro ideale con frequenza di taglio ω_c.

La funzione di trasferimento $H(\omega)$ di un filtro ideale passabasso possiede le seguenti caratteristiche:

1. $|H(\omega)|$ è costante nella banda passante ed è identicamente nullo nella banda proibita;
2. la banda passante e la banda proibita sono confinanti (separate dalla frequenza di taglio);
3. la risposta in fase $\sphericalangle H(\omega)$ è lineare; questo significa che le varie componenti armoniche nella banda passante hanno tutte lo stesso ritardo temporale.

Purtroppo la risposta all'impulso di un sistema che realizza un filtro ideale è del tipo $\text{sinc}(\omega_c t)$; essa assume valori differenti da 0 per $t < 0$ e quindi tale sistema risulta essere non causale. Questo significa che ogni sistema realizzabile non potrà mai verificare contemporaneamente le caratteristiche 1., 2. e 3.; in questo paragrafo introduciamo alcuni parametri che descrivono la "bontà" con cui un filtro realizzabile in pratica approssima un filtro ideale.

Indicando con $H(\omega)$ la funzione di trasferimento di un eventuale filtro passabasso realizzabile, sappiamo che $H(\omega)$ è completamente specificata dal suo modulo $|H(\omega)|$ e dalla sua fase $\sphericalangle H(\omega)$. La Figura 3.2 mostra la tipica forma di $|H(\omega)|$ e $\sphericalangle H(\omega)$ per un filtro passabasso realizzabile.

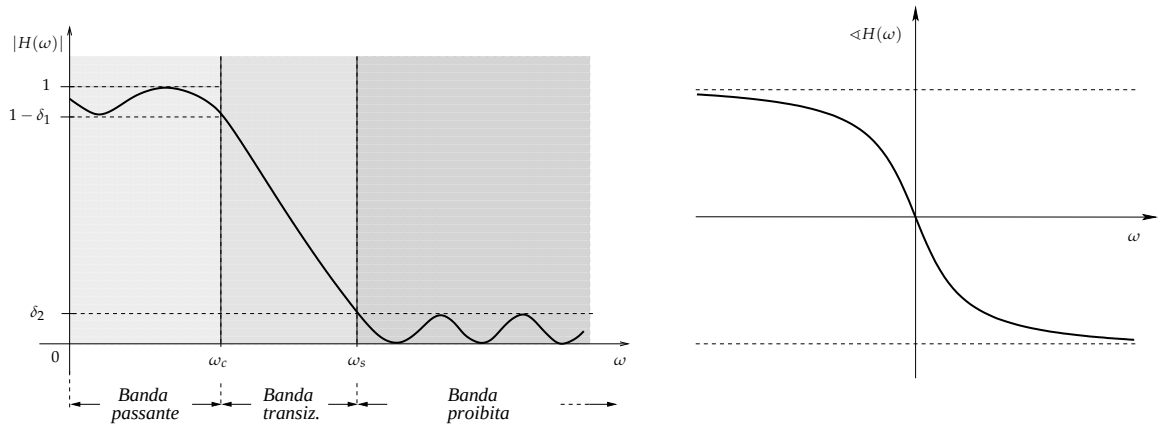


Figura 3.2 Modulo e fase di un filtro passabasso realizzabile.

Rispetto a un filtro ideale possiamo rilevare le seguenti differenze.

1. L'ampiezza $|H(\omega)|$ non è costante nella banda passante e non è identicamente nulla nella banda proibita; si possono rilevare inoltre oscillazioni (ripple) di ampiezza non trascurabile sia nella banda passante che in quella proibita. Un parametro importante è l'ampiezza della massima oscillazione in banda proibita δ_2, o equivalentemente, l'attenuazione $-20\log_{10}\delta_2$ dB.

2. La banda passante e la banda proibita non confinano, ma sono separate da una banda detta *banda di transizione*. Parametri importanti sono la frequenza di taglio ω_c a 3 dB, la frequenza di stop ω_s e la dimensione della banda di transizione $\omega_s - \omega_c$.

3. La fase $\sphericalangle H(\omega)$ non risulta essere lineare.

Analizziamo ora separatamente il significato fisico del modulo e della fase della funzione di trasferimento di un filtro.

Guadagno

Osserviamo innanzitutto che $|H(\omega)|$ è collegato al concetto di *guadagno*. A tal riguardo ricordiamo che se $F(\omega)$ e $G(\omega)$ sono le trasformate di Fourier rispettivamente dell'ingresso e dell'uscita di un sistema con funzione di trasferimento $H(\omega)$, vale che

$$G(\omega) = H(\omega)F(\omega).$$

Questo implica:

$$|H(\omega)|^2 = \frac{|G(\omega)|^2}{|F(\omega)|^2} = \frac{|G(\omega)|^2 d\omega}{|F(\omega)|^2 d\omega},$$

dove $|G(\omega)|^2 d\omega$ e $|F(\omega)|^2 d\omega$ rappresentano la potenza (o l'energia) delle componenti a frequenza tra ω e $\omega + d\omega$ rispettivamente nel segnale di uscita e in quello di ingresso;

$|H(\omega)|^2$ individua allora il *guadagno*, cioè il rapporto tra la potenza (o l'energia) del segnale in uscita e quello in ingresso alle varie frequenze.

Si osservi che per filtri senza oscillazioni in banda proibita, l'attenuazione risulta essere $-g(\omega_s)$, dove ω_s la frequenza di stop. In molte applicazioni, anziché considerare direttamente il grafico di $|H(\omega)|$, si preferisce dare il grafico del guadagno $|H(\omega)|^2$ in scala logaritmica; il guadagno può essere pertanto convertito nel seguente numero $g(\omega)$ di decibel:

$$g(\omega) = 10 \log_{10} |H(\omega)|^2 \text{ dB} = 20 \log_{10} |H(\omega)| \text{ dB}.$$

Esempio 3.1.1.

Un sistema che a una data frequenza ω converte un ingresso $F(\omega)$ di 5 V in una uscita $G(\omega)$ di 1 V, ha un guadagno a quella frequenza dato in dB da:

$$20 \log_{10} \frac{1}{5} \text{ dB} = -13.97 \text{ dB}.$$

Poiché in un filtro realizzabile la banda passante e quella proibita non confinano, bisogna attribuire un significato convenzionale alla frequenza di taglio. Una usuale nozione è quella di *frequenza di taglio a 3dB*, cioè la frequenza per la quale si ha un guadagno del 50% di quello in banda passante, che misurato in decibel equivale a

$$20 \log_{10} \frac{1}{\sqrt{2}} \text{ dB} \approx -3 \text{ dB}.$$

Se $|H(\omega)| \approx 1$ in banda passante, allora la frequenza di taglio a 3 dB è quella frequenza ω_c per cui:

$$|H(\omega_c)|^2 = \frac{1}{2}.$$

Esempio 3.1.2.

Determinare la frequenza di taglio a 3 dB del filtro passabasso con guadagno

$$|H(\omega)|^2 = \frac{1}{(1 + \omega^2/100)}.$$

Il guadagno in banda passante 1 e la frequenza di taglio ω_c quella per cui $\frac{1}{(1+\omega^2/100)} = 1/2$. Risolvendo l'equazione si ottiene $\omega_c = 10$ Hz.

Un utile nozione che permette di coprire grandi range di frequenze quella di *decade*: una decade è un intervallo tra una frequenza ω e 10ω. La misura di attenuazione di un filtro è data usualmente in dB/decade: α dB/decade significa che il filtro ha un'attenuazione di α dB ogni decade di frequenza.

Esempio 3.1.3.

Si consideri un filtro con funzione di trasferimento $H(\omega)$ tale che

$$|H(\omega)|^2 = \frac{1}{1+\omega^{2N}}.$$

Vale che:

$$10\log_{10}|H(\omega)|^2 = -20\log_{10}(1+\omega^{2N}) \approx -20N\log_{10}\omega, \qquad \text{(per } \omega \text{ suff. grande)}.$$

L'attenuazione in banda proibita allora:

$$20N\log_{10}10\omega - 20N\log_{10}\omega = 20N \text{ dB/decade}.$$

Un filtro non ha virtualmente attenuazione nella banda passante, mentre l'attenuazione in banda proibita è un importante parametro di prestazione del filtro. La banda proibita è data dall'insieme delle frequenze per le quali il guadagno è inferiore a una opportuna soglia di attenuazione che normalmente viene stabilita in funzione della particolare applicazione. Se indichiamo con ω_s, come in Figura 3.2, la frequenza di stop, cioè la frequenza di inizio della banda proibita, le frequenze tra ω_c ed ω_s costituiscono la banda di transizione.

Esempio 3.1.4.

Si consideri il filtro con funzione di trasferimento

$$|H(\omega)|^2 = \frac{1}{1+(\omega/100)^8}.$$

Determinare l'ampiezza della banda di transizione sapendo che la frequenza di stop corrisponde ad un'attenuazione di 40 dB.

La frequenza di taglio a 3 dB di 100 Hz. La frequenza di stop ω_s verifica per ipotesi

$$40 = -20\log_{10}|H(\omega_s)|.$$

Risolvendo tale equazione si ottiene $\omega_s \approx 316$ Hz. La dimensione della banda di transizione risulta $316 - 100 = 216$ Hz.

Linearità della fase

Discutiamo ora l'effetto prodotto dalla non linearità della fase della funzione di trasferimento del filtro. A tal riguardo, consideriamo per semplicità un sistema che ammette una funzione di trasferimento con modulo $|H(\omega)| = G(\omega)$ e fase $\sphericalangle H(\omega) = \phi(\omega)$, così che:

$$H(\omega) = G(\omega)e^{i\phi(\omega)}.$$

Il segnale (complesso) $e^{i\omega_1 t}$ di frequenza ω_1 viene trasformato dal sistema nel segnale $G(\omega_1)e^{i(\omega_1 t+\phi(\omega_1))}$. Se la fase lineare, cioè $\phi(\omega) = -t_0\omega$ per un'opportuna costante $t_0 \in \mathbb{R}$, il segnale di uscita

$$G(\omega_1)e^{i\omega_1(t-t_0)}.$$

Il segnale di uscita risulta allora essere lo stesso segnale di ingresso ritardato di un tempo t_0, qualsiasi sia la frequenza ω_1: una fase lineare comporta un ritardo temporale uguale per tutte le componenti armoniche.

Una fase non lineare crea invece ritardi differenti per le componenti a diversa frequenza, creando una distorsione complessiva del segnale. Per certe applicazioni (ad esempio nei modem) le distorsioni create dalla non linearità della fase devono essere eliminate quanto più possibile; in altre applicazioni la non linearità può essere utilizzata per dar luogo a effetti speciali.

3.2 Famiglie di Filtri Causali

Abbiamo visto che un filtro ideale non è causale e quindi può essere soltanto approssimato con filtri realizzabili fisicamente. A questo riguardo abbiamo introdotto parametri che denotano la bontà nell'approssimarne il guadagno (dimensione della banda di transizione, attenuazione, oscillazioni) o la fase (linearità).

La progettazione di un filtro è fortemente dipendente dall'applicazione; in certi casi (per esempio nei sistemi audio) è richiesta un'ottima risposta in fase. In altre applicazioni la linearità della fase è di scarso rilievo, mentre critica è l'accuratezza nell'approssimare il guadagno, e così via.

In aiuto al progettista, sono state introdotte e analizzate varie classi di filtri usualmente disponibili in sistemi di calcolo automatico per la progettazione, l'implementazione e la simulazione di filtri, quali ad esempio le primitive in ambiente MATLAB. Le principali famiglie sono quella dei filtri di *Butterworth*, di *Chebyschev*, di *Cauer* (o *ellittici*) e di *Bessel*. In Tabella 3.1 mostrata una grossolana valutazione comparativa della bontà di questi filtri (a parità di ordine); analizzeremo poi più in dettaglio le classi di filtri di Butterworth e Chebyschev.

Tabella 3.1 Caratteristiche qualitative di famiglie di filtri.

Filtro	**Accuratezza guadagno**	**Linearit fase**
Butterworth	media	media
Chebyschev	buona	cattiva
Ellittico	ottima	pessima
Bessel	cattiva	buona

3.2.1 Filtri di Butterworth

I filtri di Butterworth costituiscono una famiglia di filtri che soddisfa bene i requisiti sul guadagno in banda passante e meno bene in banda di transizione. Sebbene non esibiscano una fase lineare in banda passante, l'approssimazione non è troppo cattiva e sono tra i più semplici filtri elettronici da realizzare. Un filtro di Butterworth è caratterizzato da 2 parametri: l'*ordine* N e *frequenza di taglio* ω_c.

La forma generale del modulo della funzione di trasferimento di un filtro di Butterworth di ordine N e frequenza di taglio ω_c è:

$$|H(\omega)| = \frac{1}{|B_N(i\frac{\omega}{\omega_c})|} = \frac{1}{\sqrt{1+\left(\frac{\omega}{\omega_c}\right)^{2N}}},$$

dove $B_N(s)$ è un opportuno polinomio detto N-esimo polinomio di Butterworth, mentre la scalatura ω/ω_c rispetto a ω denota la frequenza normalizzata alla frequenza di taglio.

Si può dimostrare che, se N è pari, il polinomio $B_N(s)$ è dato dal prodotto di $N/2$ polinomi di secondo grado del tipo $s^2 + bs + s$ con $b > 0$, mentre se N è dispari allora è presente anche il fattore $s + 1$. La Tabella 3.2 mostra la fattorizzazione dei primi otto polinomi di Butterworth. Una proprietà interessante è che tutti i polinomi hanno le radici che giacciono sul cerchio di raggio unitario.

Tabella 3.2 I primi otto polinomi di Butterworth fattorizzati.

$B_1(s) = s + 1$
$B_2(s) = s^2 + 1.414s + 1$
$B_3(s) = (s + 1)(s^2 + s + 1)$
$B_4(s) = (s^2 + 0.765s + 1)(s^2 + 1.848s + 1)$
$B_5(s) = (s + 1)(s^2 + 0.618s + 1)(s^2 + 1.618s + 1)$
$B_6(s) = (s^2 + 0.518s + 1)(s^2 + 1.414s + 1)(s^2 + 1.932s + 1)$
$B_7(s) = (s + 1)(s^2 + 0.445s + 1)(s^2 + 1.247s + 1)(s^2 + 1.802s + 1)$
$B_8(s) = (s^2 + 0.390s + 1)(s^2 + 1.111s + 1)(s^2 + 1.663s + 1)(s^2 + 1.962s + 1)$

La risposta in frequenza di alcuni filtri di Butterworth è riportata in Figura 3.3. Riguardo alla figura si possono fare le seguenti osservazioni:

- la frequenza di taglio a 3 dB ω_c, è indipendentemente dall'ordine N del filtro;
- l'attenuazione nella banda proibita dipende da N in modo critico: risulta infatti un'attenuazione di $20N$ dB per decade;
- non sono presenti oscillazioni né in banda passante né in banda proibita: il filtro di Butterworth è quello che presenta la maggior "piattezza" in banda passante.

In una tipica situazione di progetto, il parametro ω_c è fissato essendo la frequenza di taglio desiderata, mentre l'ordine N viene scelto in modo tale da soddisfare la richiesta di attenuazione in banda proibita.

Esempio 3.2.1.

Determinare l'ordine di un filtro di Butterworth con frequenza di taglio di 100 Hz e frequenza di stop, con attenuazione a 40 dB, di 150 Hz.

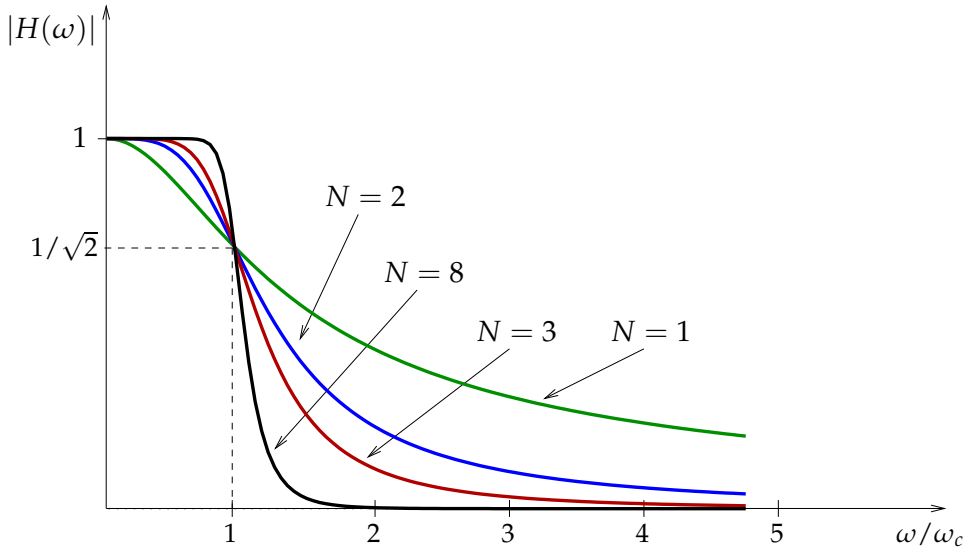


Figura 3.3 Risposta in frequenza di filtri di Butterworth.

Il guadagno di un filtro di Butterworth di ordine N, con frequenza di taglio pari a 100 Hz, è:

$$|H(\omega)|^2 = \frac{1}{1 + \left(\frac{\omega}{100}\right)^{2N}}.$$

La frequenza di stop è per ipotesi la frequenza ω_s che produce un'attenuazione di 40 dB, cioè:

$$20 \log_{10} |H(\omega_s)| = -40.$$

Si ottiene pertanto l'equazione:

$$20 \log_{10} \left(1 + \left(\frac{150}{100}\right)^{2N}\right) = 40,$$

dalla quale si ricava che $N \geq 5.68$. Il filtro di ordine 6 soddisfa la specifica.

3.2.2 Filtri di Chebyschev

Nei filtri di Butterworth la funzione guadagno è monotona decrescente sia in banda passante che in quella proibita: un filtro di Butterworth quindi approssima bene un filtro ideale all'inizio della banda passante, ma l'approssima male alla fine della banda passante e in principio di quella proibita.

Un approccio più efficiente è quello di "distribuire" l'accuratezza dell'approssimazione uniformemente lungo tutta la banda passante o quella proibita: questo può essere ottenuto scegliendo un'approssimazione che ha oscillazioni della stessa ampiezza su tutta la banda passante o quella proibita.

Questa è l'idea base su cui è progettata la classe dei filtri di Chebyschev: il guadagno di un filtro di Chebyschev ha infatti oscillazioni di uguale ampiezza in banda passante ed è monotono decrescente in banda proibita, o viceversa. A parità di ordine, un filtro di

Chebyschev ha banda di transizione più stretta e miglior attenuazione di quello di Butterworth; tuttavia i filtri di Chebyschev sono generalmente più complessi da realizzare di quelli di Butterworth e hanno una peggiore risposta in fase.

La forma generale del modulo della funzione di trasferimento di un filtro di Chebyschev di ordine N e frequenza di taglio ω_c è:

$$|H(\omega)| = \frac{1}{\sqrt{1+\varepsilon^2 C_N^2(\frac{\omega}{\omega_c})}},$$

dove ε è una costante minore di 1 e $C_N(\nu)$ è un opportuno polinomio chiamato polinomio di Chebyschev di ordine N. L'ennesimo polinomio di Chebyschev è definito come:

$$C_N(\nu) = \begin{cases} \cos(n \cdot \cos^{-1}(\nu)), & \text{se } 0 \leq \nu \leq 1 \\ \cosh(n \cdot \cosh^{-1}(\nu)), & \text{se } \nu > 1 \end{cases},$$

e può essere ottenuto dalla seguente equazione di ricorrenza:

$$C_N(\nu) = \begin{cases} 1, & \text{se } N = 0 \\ \nu, & \text{se } N = 1 \\ 2\nu C_{N-1}(\nu) - C_{N-2}(\nu), & \text{se } N > 1 \end{cases}.$$

I grafici dei polinomi $C_6(\nu)$ e $C_{12}(\nu)$ sono mostrati in figura 3.4.

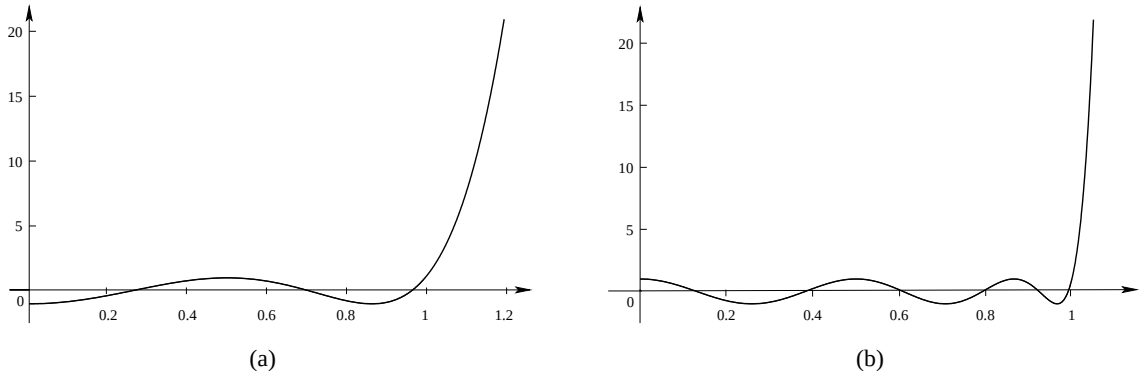


Figura 3.4 (a) Grafico del polinomio di Chebyshev $C_6(\nu)$ e **(b)** del polinomio $C_{12}(\nu)$.

A giustificazione dell'equazione precedente possiamo definire innanzitutto la variabile $\phi = \cos^{-1}(\nu)$, così da poter scivere $C_N(\nu) = \cos(n\phi)$. Sfuttando ora l'identità trigonometrica $\cos(\alpha \pm \beta) = \cos(\alpha)\cos(\beta) \mp \sin(\alpha)\sin(\beta)$, si determina

$$\cos((n+1)\phi) + \cos((n-1)\phi) = 2\cos(n\phi)\cos(\phi).$$

Dalla posizione $\phi = \cos^{-1}(\nu)$ si ricava dunque:

$$C_{N+1}(\nu) = 2\nu C_N(\nu) - C_{N-1}(\nu)$$

con cui generare i polinomi di qualunque ordine, come quelli riportati in Tabella 3.3.

In Figura 3.5 viene mostrata la risposta in frequenza di alcuni filtri di Chebyschev.

Tabella 3.3 I primi sei polinomi di Chebyschev.

$C_0 = 1$
$C_1 = \nu$
$C_2 = 2\nu^2 - 1$
$C_3 = 4\nu^3 - 3\nu$
$C_4 = 8\nu^4 - 8\nu^2 + 1$
$C_5 = 16\nu^5 - 20\nu^3 + 5\nu$
$C_6 = 32\nu^6 - 48\nu^4 + 18\nu^2 - 1$

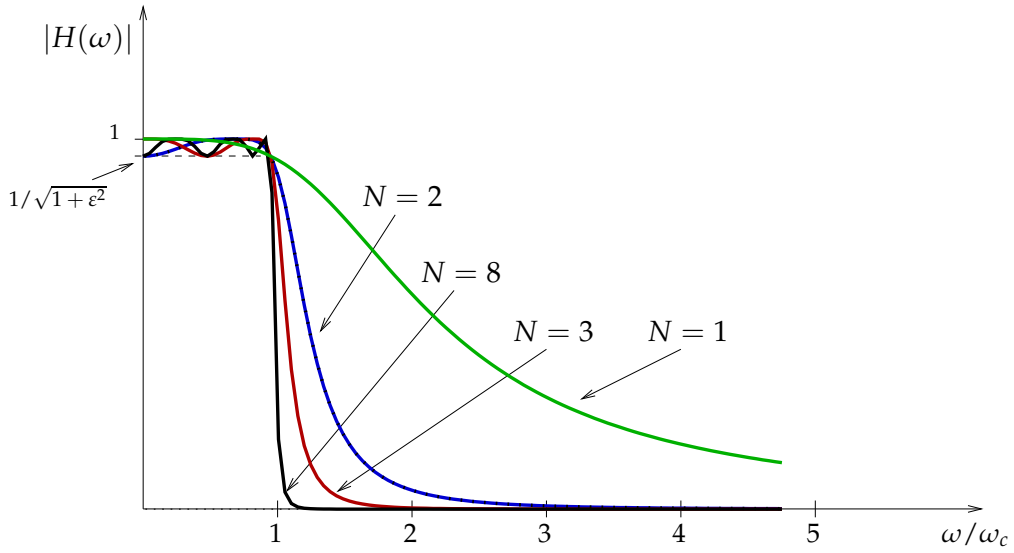


Figura 3.5 Risposta in frequenza di filtri di Chebyschev.

Si può osservare che il guadagno decresce molto rapidamente attorno alla frequenza di taglio anche per piccoli valori di N (per esempio $N = 8$). Tuttavia questi filtri hanno lo svantaggio, rispetto a quelli di Butterworth, di presentare oscillazioni in banda passante dipendenti da ε; le oscillazioni hanno tutte la stessa ampiezza e sono sempre contenute nell'intervallo $[1/\sqrt{1+\varepsilon^2}, 1]$, indipendentemente dall'ordine del polinomio.

Riassumendo, un filtro di Chebyschev è specificato dai tre parametri: ε, ω_c e N; in una tipica situazione di progetto il parametro ε determina l'ampiezza dell'oscillazione ammessa in banda passante, la frequenza ω_c specifica la frequenza di taglio desiderata e l'ordine N viene scelto in modo da soddisfare la richiesta di attenuazione in banda proibita.

Esempio 3.2.2.

Si determini l'ordine di un filtro di Chebyshev con un'ondulazione di 1 dB e in grado di fornire un'attenauzione di 40 dB in corrispondenza di $\omega/\omega_c = 2$. Si faccia riferimento alla banda passante del filtro a 3 dB.

Il parametro ε è legato all'ondulazione espressa in decibel consentita in banda passante γ come

$$\varepsilon^2 = 10^{\gamma/10} - 1.$$

Se $\gamma = 1$ dB si ricava il valore massimo di oscillazione pari a $\varepsilon = 0.5088$.
Il guadagno in frequenza normalizzata invece risulta:

$$|H(\omega)|^2 = \frac{1}{1 + \varepsilon^2 C_N^2(\nu)}.$$

Un'attenuazione di 40 dB si ha quando $|H(\omega)|^2 = 0.01$, che implica risolvere

$$(0.01)^2 = \frac{1}{1 + (0.5088)^2 C_N^2(2)}.$$

Utilizzando la definizione di n-esimo polinomio di Chebyshev per $\nu > 1$, la soluzione dell'equazione precedente comporta la scelta del polinomio che in $\nu = 2$ assume valore 196.5, che equivale a risolvere:

$$C_N(2) = \cosh(n \cdot \cosh^{-1}(2)) = 196.5.$$

Risolvendo rispetto a n otteniamo che l'intero $n = 5$ soddisfa i requisiti.

3.3 Realizzazione di Filtri Analogici

Dato un filtro con risposta in frequenza $H(\omega)$, la sua realizzazione consiste nel progettare un circuito elettrico che, visto come sistema, ha $H(\omega)$ come funzione di trasferimento. Questo circuito, dopo un accurato esame di sue eventuali imperfezioni come capacità parassite o altro, potrà essere implementato ottenendo la realizzazione fisica del filtro.

Le realizzazioni di filtri analogici possono essere classificate sulla base delle componenti costituenti; elenchiamo qui brevemente le principali classi con alcune caratteristiche.

Filtri RLC passivi. Essi consistono di componenti passive come induttanze, condensatori e resistenze. Per il loro funzionamento non richiedono potenza aggiuntiva, ma spesso esibiscono perdite significative in banda passante; la presenza di induttanze, inoltre, pone seri problemi alla loro miniaturizzazione. Lavorano bene fino a 500 Hz. Filtri RLC passivi saranno analizzati più in dettaglio in Sezione 3.3.1.

Filtri RC attivi. Questi filtri non contengono induttanze, ma resistenze, condensatori e amplificatori operazionali. Si prestano bene alla miniaturizzazione, ma richiedono potenza aggiuntiva. Lavorano bene da 1 Hz fino a 500 Hz. In Sezione 3.3.2 si analizza la realizzazione di filtri di Butterworth con componenti attive.

Filtri a microonde. Essi consistono di componenti passive come le linee di trasmissione, linee di trasmissione accoppiate e cavità risonanti. Non richiedono potenza aggiuntiva e sono utilizzate per filtri che lavorano sopra i 300 MHz.

Filtri a cristallo. Sono costituiti da risuonatori piezoelettrici ed lavorano dai 10 KHz ai 200 MHz. Con risuonatori di quarzo si possono realizzare filtri passabanda con ampiezza di banda veramente ristretta.

Filtri elettromeccanici. Sono costituiti da risuonatori meccanici: per prima cosa il segnale elettrico viene trasformato in vibrazioni meccaniche, viene poi applicato il filtro e infine il segnale è riconvertito in segnale elettrico. Questi filtri lavorano fino a 200 MHz.

Nel resto della sezione, analizziamo più in dettaglio i filtri RLC passivi e i filtri RC attivi.

3.3.1 Circuiti ad Elementi Passivi

Un filtro analogico può essere realizzato mediante una rete di resistenze, induttanze e condensatori, detto anche circuito ad elementi passivi. Il sistema così ottenuto riceve in ingresso una tensione $v(t)$ posta tra due punti della rete e dà in uscita una tensione $u(t)$ misurata tra altri due punti della rete. Un esempio di circuito è mostrato in Figura 3.6

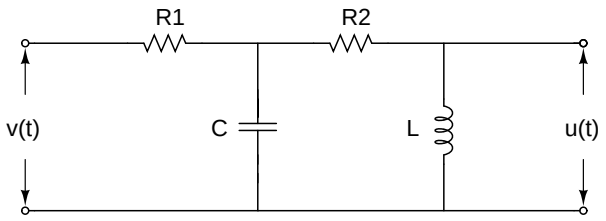


Figura 3.6 Circuito ad elementi passivi.

Gli elementi basilari di tali reti sono la resistenza, il condensatore e l'induttanza, rappresentati coi simboli mostrati in Figura 3.7.

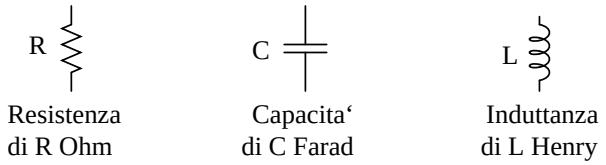


Figura 3.7 Resistenza, condensatore e induttanza.

Ponendo ai capi di tali elementi una tensione di $v(t)$, si crea una corrente di $i(t)$ tale che:

- per la resistenza R vale: $v(t) = Ri(t)$;
- per l'induttanza L vale: $v(t) = L\dfrac{d}{dt}i(t)$;
- per il condensatore C vale: $v(t) = \dfrac{1}{C}\displaystyle\int_{-\infty}^{t} i(\tau)d\tau$.

Indicando ora con $V(\omega)$ e $I(\omega)$ rispettivamente la trasformata di Fourier di $v(t)$ e $i(t)$, applicando le regole di derivazione e integrazione (vedi Tabella 2.2), le precedenti equazioni si riscrivono nella rappresentazione in frequenza come segue:

- per la resistenza R vale: $V(\omega) = RI(\omega)$;
- per l'induttanza L vale: $V(\omega) = i\omega LI(\omega)$;
- per il condensatore C vale: $V(\omega) = \dfrac{1}{i\omega C}I(\omega)$.

La relazione di ingresso-uscita di un circuito può essere ricavata risolvendo le equazioni ottenute dai due principi di Kirkoff:

I. La somma delle correnti entranti in un nodo è uguale alla somma delle correnti uscenti.

II. La somma delle tensioni lungo un circuito chiuso è 0.

Detta $V(\omega)$ la trasformata di Fourier dell'ingresso $v(t)$ e $U(\omega)$ la trasformata di Fourier dell'uscita $u(t)$, applicando i principi di Kirkoff nella rappresentazione in frequenza si ottiene in generale che:

$$U(\omega) = H(\omega)V(\omega),$$

dove $H(\omega)$ dipende dalla struttura del circuito e può essere interpretata come la funzione di trasferimento di un sistema lineare tempo-invariante; in tal modo è possibile realizzare dei filtri lineari con reti a elementi passivi.

Esempio 3.3.1.

Analizziamo le caratteristiche del filtro ottenuto dal seguente circuito:

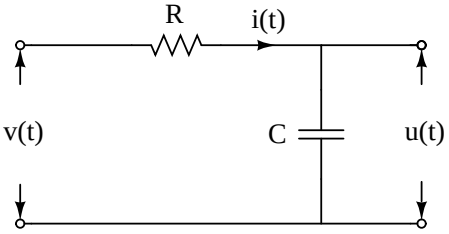


Figura 3.8 Circuito RC.

Applicando il principio II nella rappresentazione in frequenza, vale:

$$V(\omega) = RI(\omega) + \frac{1}{i\omega C} I(\omega).$$

Il comportamento del condensatore è dato da:

$$U(\omega) = \frac{1}{i\omega C} I(\omega).$$

Dividendo membro a membro le due uguaglianze precedenti, si ottiene la seguente funzione di trasferimento $H(\omega)$:

$$H(\omega) = \frac{U(\omega)}{V(\omega)} = \frac{1}{1 + i\omega RC}.$$

Risulta quindi che guadagno e fase siano:

$$|H(\omega)|^2 = \frac{1}{1 + (\omega RC)^2}, \qquad \sphericalangle H(\omega) = -\arctan RC\omega.$$

I grafici del guadagno e della fase sono dati in Figura 2.10; la rete precedentemente descritta implementa quindi un filtro passabasso di Butterworth di ordine 1 con frequenza di taglio a 3 dB uguale a $1/RC$. La fase in banda passante è solo approssimativamente lineare.

Esempio 3.3.2.

Si consideri il filtro realizzato dal circuito RLC in Figura 3.9. Provare che il circuito realizza un filtro di Butterworth di ordine 2.

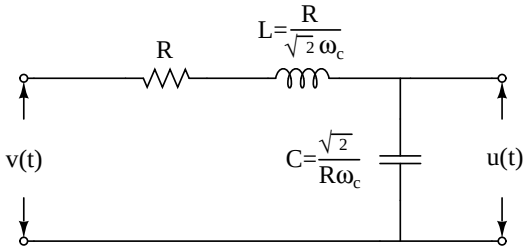


Figura 3.9 Filtro di Butterworth di ordine 2.

Applicando il principio II nella rappresentazione in frequenza, vale:

$$V(\omega) = RI(\omega) + i\omega LI(\omega) + \frac{1}{i\omega C} I(\omega).$$

Il comportamento del condensatore è dato da:

$$U(\omega) = \frac{1}{i\omega C} I(\omega).$$

Dividendo membro a membro le due uguaglianze precedenti, si ottiene la seguente funzione di trasferimento $H(\omega)$:

$$H(\omega) = \frac{U(\omega)}{V(\omega)} = \frac{1}{1 + iRC\omega - LC\omega^2}.$$

Poiché $L = \frac{R}{\sqrt{2}}$ e $C = \frac{\sqrt{2}}{R\omega_c}$, si ottiene infine:

$$H(\omega) = \frac{1}{1 + i\sqrt{2}\frac{\omega}{\omega_c} - \left(\frac{\omega}{\omega_c}\right)^2}$$

Risulta quindi che guadagno e fase siano:

$$|H(\omega)|^2 = \frac{1}{1 + \left(\frac{\omega}{\omega_c}\right)^4}, \qquad \sphericalangle H(\omega) = -\arctan \frac{\sqrt{2}\frac{\omega}{\omega_c}}{1 - \left(\frac{\omega}{\omega_c}\right)^2}.$$

3.3.2 Realizzazione di Filtri di Butterworth con Circuiti ad Elementi Attivi

In questa sezione presentiamo una tecnica per la realizzazione di filtri di Butterworth analogici mediante circuiti ad elementi attivi, contenenti resistenze, condensatori e amplificatori operazionali. Il fatto che non sia richiesta alcuna induttanza è, come detto, un vantaggio importante nella pratica poiché le induttanza sono voluminose, contengono resistenze e capacità parassite e dissipano considerevole potenza. Il metodo può essere facilmente esteso alla costruzione di altri tipi di filtri.

I passi principali possono essere riassunti come segue.

1. Si consideri la funzione di trasferimento del filtro. Nel caso del filtro di Butterworth di ordine N con frequenza di taglio ω_c, risulta

$$H(\omega) = \frac{1}{B_N(\frac{s}{\omega_c})},$$

dove $s = i\omega$ e $B_N(z)$ è il polinomio di Butterworth di ordine N.

2. Si decompone $B_N(z)$ come prodotto $p_1(z) \cdot p_2(z) \cdots p_m(z)$, dove $p_i(z)$ è del tipo $z+1$ oppure $z^2 + bz + 1$, con b reale positivo. In Tabella 3.2 sono riportate le fattorizzazioni dei primi otto polinomi di Butterworth.

3. Si realizzano separatamente i sistemi S_i che hanno $1/p_i(s/\omega_c)$ come funzione di trasferimento ($1 \leq i \leq m$).

4. Si costruisce il sistema S ottenuto ponendo in cascata i sistemi S_i, in modo che l'uscita di S_i sia l'ingresso di S_{i+1} ($1 \leq i < m$), come rappresentato Figura 3.10.

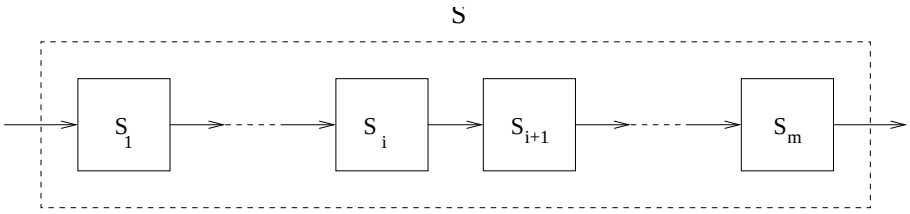


Figura 3.10 Cascata dei sistemi S_i, con $1 \leq i \leq m$.

La costruzione è corretta poiché il sistema complessivo S ha come funzione di trasferimento $H(\frac{s}{\omega_c})$ il prodotto delle funzioni di trasferimento dei sistemi S_i, cioè:

$$H\left(\frac{s}{\omega_c}\right) = \frac{1}{p_1(\frac{s}{\omega_c})} \cdots \frac{1}{p_m(\frac{s}{\omega_c})} = \frac{1}{p_1(\frac{s}{\omega_c}) \cdots p_m(\frac{s}{\omega_c})} = \frac{1}{B_N(\frac{s}{\omega_c})}.$$

Il passo 4. mostra come un arbitrario filtro di Butterworth possa essere ottenuto da una composizione in cascata di sistemi che hanno funzioni di trasferimento del tipo:

$$H_1\left(\frac{s}{\omega_c}\right) = \frac{1}{\frac{s}{\omega_c} + 1} \qquad \text{oppure} \qquad H_2\left(\frac{s}{\omega_c}\right) = \frac{1}{(\frac{s}{\omega_c})^2 + b\frac{s}{\omega_c} + 1}.$$

Illustriamo ora come costruire un circuito a componenti attive che realizza il filtro con funzione di trasferimento

$$H_2\left(\frac{s}{\omega_c}\right) = \frac{1}{(\frac{s}{\omega_c})^2 + b\frac{s}{\omega_c} + 1}. \tag{3.1}$$

A questo scopo si consideri il circuito (a) in Figura 3.11. Esso è composto da elementi passivi (condensatori e resistenze) e da un amplificatore operazionale non-invertente che permette di amplificare il segnale V_2 di un fattore A, controllato dalle resistenze R_1 e R_2:

$$U = AV_2 \qquad \text{con} \qquad A = \frac{R_1 + R_2}{R_1} > 1.$$

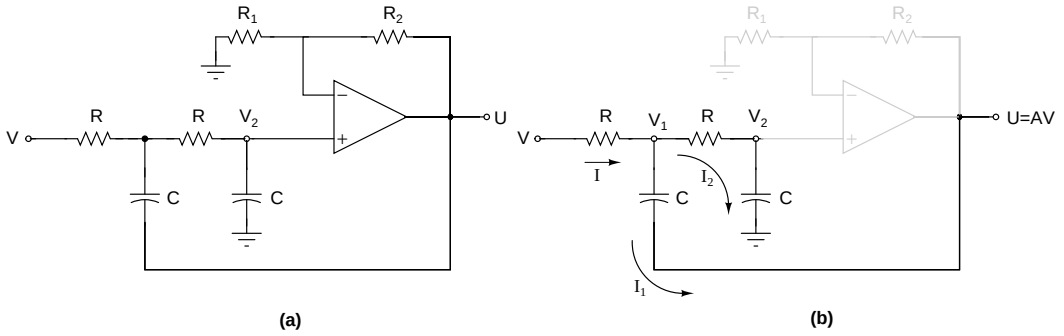


Figura 3.11 (a) Filtro di Butterworth di ordine 2. (b) Sottocircuito.

Analizziamo ora il sottocircuito (b) in Figura 3.11; con V, V_1, V_2, U denotiamo le trasformate di Fourier delle tensioni nei punti indicati e con I, I_1, I_2 le trasformate di Fourier delle correnti nei cammini indicati. Poiché $I = I_1 + I_2$ risulta:

$$V = R(I_1 + I_2) + V_1. \tag{3.2}$$

Analogamente:

$$V_1 = RI_2 + V_2, \qquad V_1 = \frac{1}{sC} I_1 + U,$$
$$V_2 = \frac{1}{A} U, \qquad V_2 = \frac{1}{sC} I_2.$$

Queste ultime quattro equazioni permettono di esprimere V_1, V_2, I_1, I_2 in termini di U; sostituendo nella (3.2) si ottiene infine:

$$V = [(RCs)^2 + (3 - A)RCs + 1]U.$$

Il circuito realizza dunque un filtro con funzione di trasferimento:

$$\frac{1}{(RCs)^2 + (3 - A)RCs + 1}.$$

Per ottenere il filtro con la funzione di trasferimento desiderata (3.1) basta scegliere R, C ed A tali che:

$$RC = \frac{1}{\omega_c}, \tag{3.3}$$
$$A = 3 - b. \tag{3.4}$$

Il metodo sopra esposto del tutto generale. In particolare:

- Esso può essere applicato per la realizzazione di altri filtri, ad esempio quelli di Chebyschev, semplicemente considerando le fattorizzazioni dei polinomi di Chebyschev in fattori di primo e secondo grado a coefficienti reali.
- Esso può essere facilmente adattato alla costruzione di filtri passaalto, osservando che la funzione di trasferimento di un filtro passaalto con frequenza di taglio ω_c pu essere ottenuta dalla funzione di trasferimento del filtro passabasso con la stessa frequenza di taglio, operando la sostituzione $\frac{\omega}{\omega_c} \to \frac{\omega_c}{\omega}$. In particolare, un circuito passalto con funzione di trasferimento H_2 può essere ottenuto scambiando le resistenze R con i condensatori C nel circuito in Figura 3.11.

- Per quanto riguarda infine la realizzazione di filtri passabanda, un filtro passabanda con frequenza di taglio $\omega_1 < \omega_2$ può essere ottenuto ponendo in cascata un filtro passabasso con frequenza di taglio ω_2 e un filtro passaalto con frequenza di taglio ω_1.

Esempio 3.3.3.

Realizzazione di un filtro di Butterworth passabasso di ordine 4 con frequenza di taglio $f_0 = 1\ kHz$.

Ricordiamo (vedi Tabella 3.2) che il polinomio di Butterworth di ordine 4 è:

$$B_4(s) = (s^2 + 0.765s + 1)(s^2 + 1.848s + 1).$$

Il filtro sarà ottenuto ponendo in cascata i due circuiti aventi funzioni di trasferimento:

$$\frac{1}{(\frac{s}{\omega_c})^2 + 0.765\frac{s}{\omega_c} + 1}, \qquad \frac{1}{(\frac{s}{\omega_c})^2 + 1.848\frac{s}{\omega_c} + 1}, \qquad \text{con } \omega_c = 2\pi f_0.$$

Tali circuiti sono del tipo mostrato in Figura 3.11. Per la (3.4), gli amplificatori operazionali dei due circuiti devono avere guadagni $A' = 3 - 0.765 = 2.235$, $A'' = 3 - 1.848 = 1.152$.

Poiché $A' = \frac{R'_1 + R'_2}{R'_1}$ e $A'' = \frac{R''_1 + R''_2}{R''_1}$, fissati arbitrariamente $R'_1 = R''_1 = 10\ k\Omega$, si ricava $R'_2 = 12.35\ k\Omega$ e $R''_2 = 1.52\ k\Omega$. Per la (3.3), i valori di R e C nei due circuiti devono verificare $RC = \frac{1}{2\pi 10^3}$ sec. Fissato arbitrariamente $C = 0.1\mu F$, si ottiene $R = 1.6\ k\Omega$.

Il circuito finale, ottenuto ponendo in cascata i due circuiti precedenti, è mostrato in Figura 3.12.

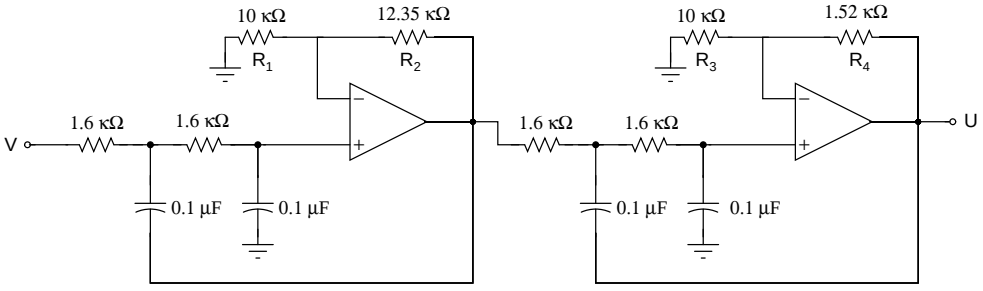


Figura 3.12 Filtro di Butterworth passabasso di ordine 4.