CALCOLO NUMERICO

UNIVERSITÀ DEGLI STUDI
DI NAPOLI FEDERICO II

Alunni: *COPPOLA ADRIANO*
ANDREA OLANDESE

Corso di studi: Magistrale Ingegneria Informatica

A.A. 2020-2021

Docente: Luisa D'Amore

Sommario

Nella seguente documentazioni sono stati esplorati i concetti teorici delle trasformate di Fourier, approfondendo il tema della quadratura numerica e delle FFT, esaminati attraverso MATLAB e attraverso un elaborato.
L'elaborato sfrutta l'algoritmo FFT per manipolare due file audio, al fine di isolare una determinata parte del primo ed eliminare il rumore del secondo.

Indice

Capitolo I – DFT

L’analisi di segnali è caratterizzata dal passaggio dal dominio del tempo a quello della frequenza. Comunemente le informazioni in frequenza sono quelle più interessanti per i segnali audio e video, essendo più semplici da quantificare e manipolare.
In matematica si può fare ciò applicando alla funzione una trasformazione, in particolare per l’analisi frequenziale si usa la trasformata di Fourier.
Tale trasformazione deriva dalla serie di Fourier, essa ha come scopo il descrivere un segnale periodico continuo mediante la somma di funzioni periodiche molto elementari, ossia le sinusoidi. Con un numero sufficiente di queste funzioni è possibile costruire qualsiasi segnale con una buona accuratezza.

$$f(t) = a_0 + \sum_{n=1}^{\infty} a_n \cos(nw_0 t) + b_n \sin(nw_0 t)$$

In frequenza un segnale si può considerare come la somma delle sue componenti frequenziali, le quali possiamo analizzare e modificare.
La trasformata di Fourier di una funzione del tempo x(t) è definita come:

$$F[x(t)](\omega) = X(\omega) := \int_{-\infty}^{\infty} x(t) e^{-\mathbf{i}\omega t} dt, \qquad \omega \in \mathbb{R}$$

Essa ci fornisce una funzione che si trova nel dominio delle frequenze e che quindi evidenzia il contributo delle sinusoidi ad ogni frequenza.
L’integrale è indefinito in quanto teoricamente ci sarebbe bisogno di un

numero infinito di sinusoidi per descrivere un segnale reale.

L'utilizzo di un calcolatore implica memoria limitata, ergo dobbiamo lavorare nel tempo discreto e quindi con un numero finito di componenti frequenziali.

Dato un vettore f composto da un insieme di valori f_j che la funzione assume in istanti differenti ,ossia campioni, possiamo definire la trasformata discreta di Fourier come:

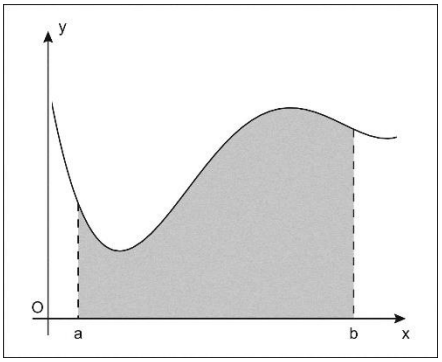
$$F_k = \sum_{j=0}^{N-1} f_j e^{-i\ 2\pi\ jk\ /N}$$

Come è possibile notare siamo passati da un integrale indefinito ad una serie finita, ciò evidenzia come quello che ricaviamo sia una approssimazione del segnale originale, calcolata mediante un processo noto come quadratura numerica.

Capitolo II – Quadratura Numerica

La quadratura numerica è un insieme di metodi atti a stimare il valore di un integrale senza dover calcolare la primitiva della funzione integranda. Nel calcolo numerico è fondamentale come operazione in quanto unico modo per calcolare l'approssimazione di un integrale.

Calcolare un integrale significa trovare l'area di un rettangoloide, ossia di una figura che è delimitata dalla curva di funzione, l'asse delle x e due perpendicolari ad esso tracciate dagli estremi dell'intervallo di integrazione.



Non essendo possibile calcolarlo precisamente, troviamo figure la cui area approssima sufficientemente quella del rettangoloide.

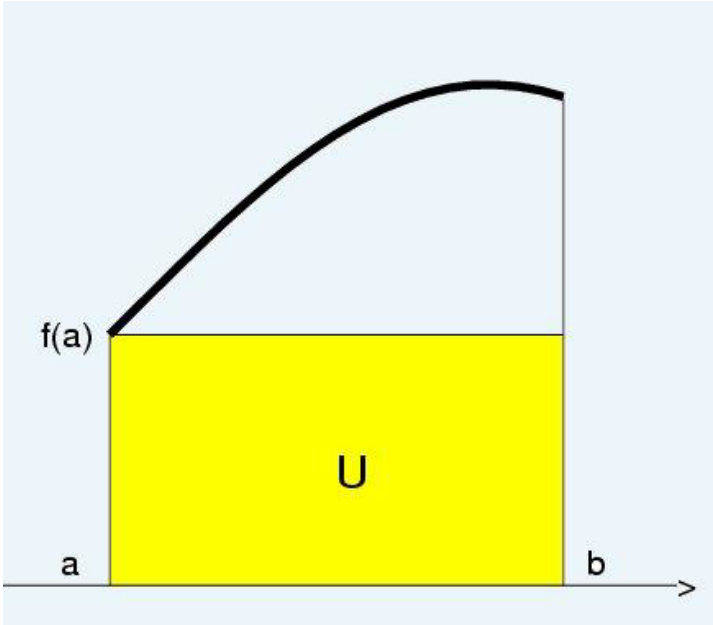
Di seguente sono illustrati alcuni dei metodi relativi alla quadratura numerica più comuni:

Formula Rettangolare

Area=f(a)(b-a)

Questa è una formula semplice che usa l'area del rettangolo, usando il valore a inizio intervallo come altezza.

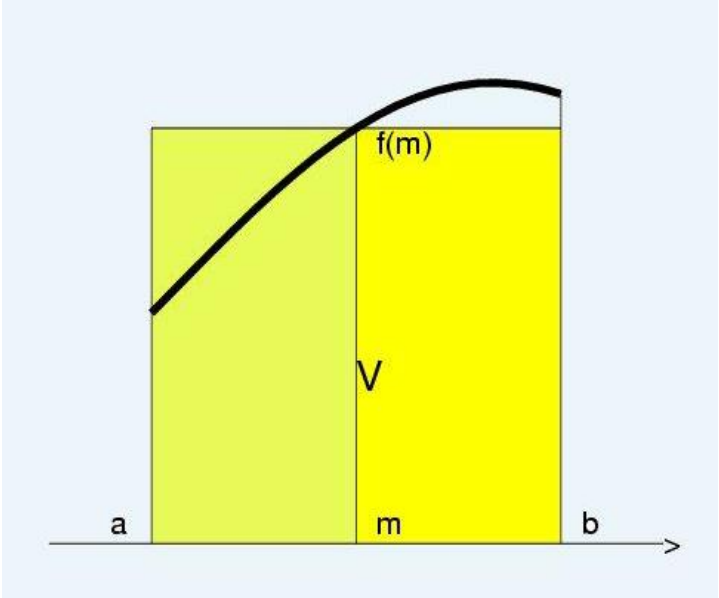
Si commette un errore abbastanza alto nel calcolo dell'integrale.



Formula del Punto Medio

Area=f(m)(b-a)

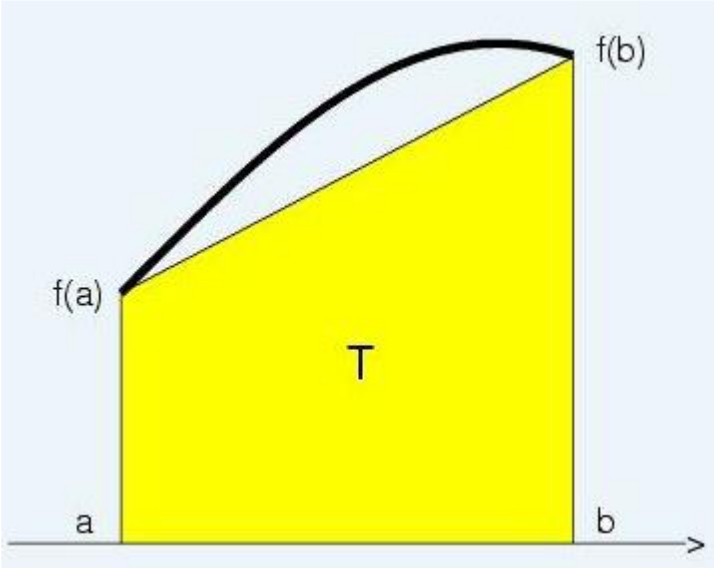
Simile alla formula di prima utilizza però come altezza il valore della funzione al punto medio dell'intervallo.



Formula Trapezoidale

$$\text{Area}=[f(a)+f(b)]\frac{(b-a)}{2}$$

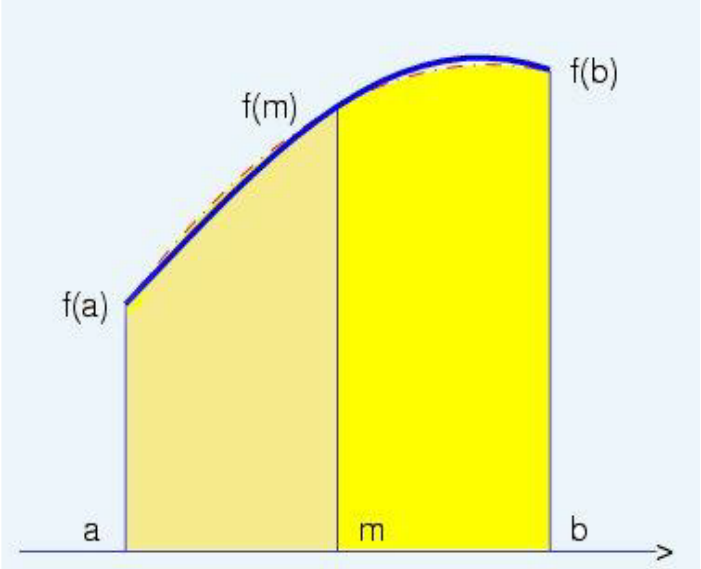
Questa formula utilizza l'area del trapezio per cercare di approssimare con più accuratezza il rettangoloide.



Formula di Simpson

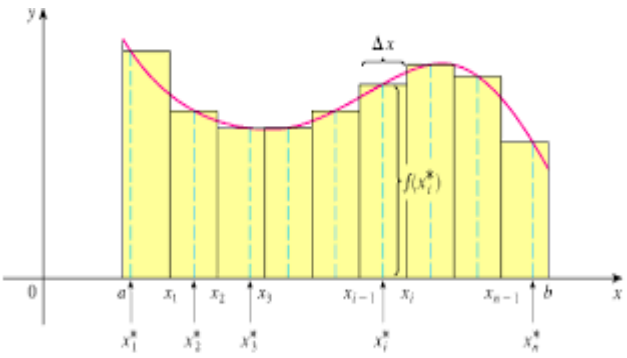
$$\text{Area}=[f(a)+4f(m)+f(b)]\frac{(b-a)}{6}$$

La formula di Simpson effettua una media pesata tra i valori di funzione agli estremi e al punto medio, dando un peso maggiore a quest'ultimo.



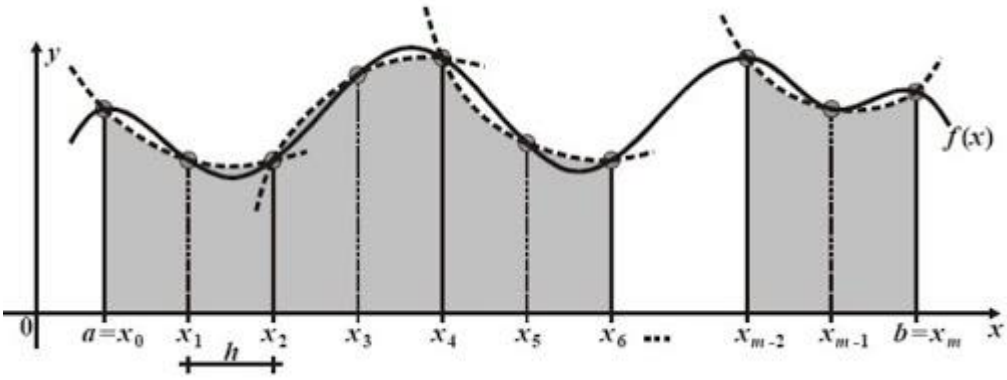
Somme di Riemann

$$\sigma_n = \sum_{i=0}^{n-1} f(t_i)(x_{i+1} - x_i)$$



Le somme di Riemann sono approssimazioni dell'area di principio simile a quello della formula del rettangolo, ma che tiene considerazione di diverse scelte possibili per l'altezza, ad esempio se poniamo $t_i=x_i$ otteniamo una somma inferiore di Riemann, se poniamo $t_i=x_{i+1}$ otteniamo una somma superiore di Riemann.
Per l'integrale secondo Riemann se il numero di intervalli tende ad infinito da queste somme otteniamo l'effettivo valore dell'integrale, premesso che la funzione sia integrabile.

Come si può notare ognuna di queste formule è una combinazione lineare di valori misurabili della funzione integranda, quindi calcolabili numericamente.
Il problema dell'utilizzo di queste formule è che si commette sempre un errore di discretizzazione.
Per ridurre l'errore di discretizzazione relativo ad una formula di quadratura la si può applicare su dei sottointervalli dell'intervallo di integrazione.



Si osserva che dividendo in due l'intervallo di integrazione e applicando la formula trapezoidale l'errore si riduce di circa 4 volte, viene naturale pensare che continuando a ridurre l'ampiezza dei sotto intervalli si ottenga una riduzione esponenziale dell'errore.

Effettivamente l'utilizzo di formule composte con sottointervalli sempre più piccoli comporta le stesse conclusioni delle somme di Riemann, poiché si dimostra che una formula trapezoidale composta è appunto decomponibile in somme di Riemann, quindi se il numero di sottointervalli tende all'infinito si ottiene una valutazione precisa dell'integrale della funzione.

Di seguente un esempio in MATLAB della funzione trapezoidale composta.

```
function [Int,Err,Iflag] = trapez_comp(a,b,Tol,f,Maxsum)
%TRAPEZ_COMP calcola l'integrale definito nell'intervallo [a,b] di f
%   con la formula trapezoidale composita a meno di una tolleranza
%   Tol ed effettuando un limite massimo di somme Maxsum.
%   L'output è composto dall'integrale, l'approssimazione dell'errore
%   e un flag che indica se abbiamo raggiunto Maxsum somme

m = 2;                          %Numero iniziale di segmenti
h = (b-a)/m;                    %Grandezza di ogni segmento
x = (a+b)/m;                    %Punto medio dell'intervallo
oldint = h*(f(a)+f(b));         %Integrale calcolato precedentemente,
                                %   in questo caso su un unico trapezio
Int = oldint/2 + h*f(x);        %Integrale tramite formula trapezoidale
                                    %composta con due trapezi
nsum = 3;                       %Numero corrente di somme effettuate
Err = abs(Int-oldint)/3;        %Stima dell'errore di discretizzazione della
                                %   formula con m sottointervalli

while Err > Tol && nsum < Maxsum
   m = 2*m;                     %Raddoppiamo il numero di intervalli...
   h = h/2;                     %...si dimezza la grandezza degli intervalli...
                                %...e riduciamo di circa 4 volte l'errore
   sum = 0;                     %Valore temporaneo della somma

   for k = 1:2:m-1              %Valutiamo la funzione nei punti non
                                %   non ancora valutati
        x = a + k*h;
        sum = sum + f(x);
        nsum = nsum + 1;
   end
   oldint = Int;
   Int = oldint/2 + h*sum;  %oldint viene dimezzato proprio come h
   Err = abs(Int-oldint)/3;
end

if nsum < Maxsum
    Iflag = 0;                  %Flag che indica se abbiamo superato il
                                %   numero massimo di somme
else
    Iflag = 1;
end
end
```

```
%Esempio di applicazione della formula trapezoidale composita
%tramite la funzione trapez_comb, usata per calcolare l'integrale
%della curva f(x)=sin(x) nell'intervallo [0,pi]

a = 0; b = pi; Tol = 10^(-5); Maxsum = 5000;
f = @(x) sin(x);

[Int, Err, Iflag] = trapez_comp(a,b,Tol,f,Maxsum);

Int
Err
Iflag
```

```
>> esempio_trapez

Int =

    2.0000


Err =

   6.2749e-06


Iflag =

     0
```

Generalmente l'errore è valutato come: E(f)=I(f)-Q(f)

I(f) è la funzione che si sta approssimando con Q(f) quindi non è nota e di conseguenza dobbiamo scegliere una strada diversa per valutare E(f). Per farlo ci serviamo di una coppia di formule innestate, ossia di due formule in cui l'insieme di nodi di una delle formule è contenuto in quello dell'altra.

Ad esempio considerando una formula trapezoidale ad m nodi T_m, possiamo tenere conto di T_{2m} per trovare una stima dell'errore. Si dimostra che l'errore di discretizzazione relativo a una formula che divide l'intervallo in 2m nodi è pari a:

$|E_{2m}[f]|=|T_m[f]-T_{2m}[f]|/3$

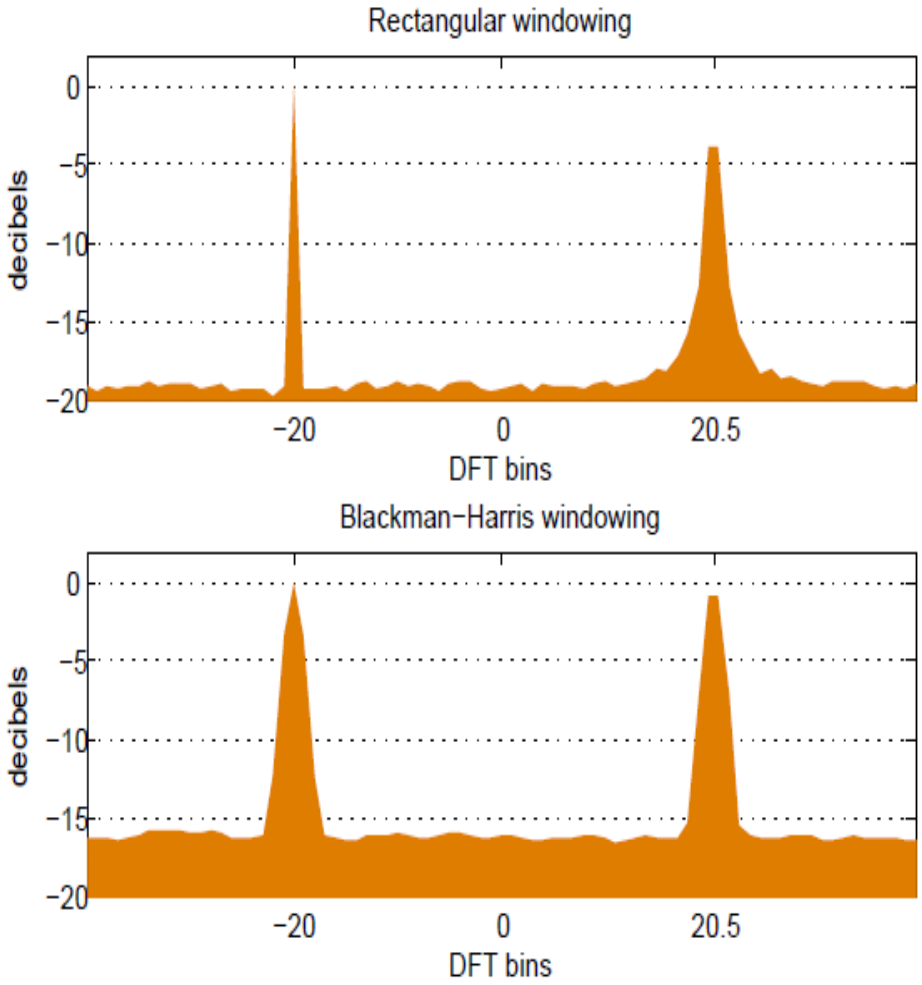
Generalmente in Matlab viene impostato un valore di tolleranza per l'errore tale da essere considerato accettabile.

Di seguente vi è un esempio in Matlab che mostra un algoritmo che calcola l'integrale di una funzione.

```
function [Int,Err,Iflag] = trapez_comp(a,b,Tol,f,Maxsum)
%TRAPEZ_COMP calcola l'integrale definito nell'intervallo [a,b] di f
%   con la formula trapezoidale composita a meno di una tolleranza
%   Tol ed effettuando un limite massimo di somme Maxsum.
%   L'output è composto dall'integrale, l'approssimazione dell'errore
%   e un flag che indica se abbiamo raggiunto Maxsum somme

m = 2;                          %Numero iniziale di segmenti
h = (b-a)/m;                    %Grandezza di ogni segmento
x = (a+b)/m;                    %Punto medio dell'intervallo
oldint = h*(f(a)+f(b));         %Integrale calcolato precedentemente,
                                %   in questo caso su un unico trapezio
Int = oldint/2 + h*f(x);        %Integrale tramite formula trapezoidale
                                    %composta con due trapezi
nsum = 3;                       %Numero corrente di somme effettuate
Err = abs(Int-oldint)/3;        %Stima dell'errore di discretizzazione della
                                %   formula con m sottointervalli

while Err > Tol && nsum < Maxsum
    m = 2*m;                    %Raddoppiamo il numero di intervalli...
    h = h/2;                    %...si dimezza la grandezza degli intervalli...
                                %...e riduciamo di circa 4 volte l'errore
    sum = 0;                    %Valore temporaneo della somma

    for k = 1:2:m-1             %Valutiamo la funzione nei punti non
                                %   non ancora valutati
        x = a + k*h;
        sum = sum + f(x);
        nsum = nsum + 1;
    end
    oldint = Int;
    Int = oldint/2 + h*sum;  %oldint viene dimezzato proprio come h
    Err = abs(Int-oldint)/3;
end

if nsum < Maxsum
    Iflag = 0;                  %Flag che indica se abbiamo superato il
                                %   numero massimo di somme
else
    Iflag = 1;
end
end
```

Capitolo III – Window Leakage

Il passaggio da trasformata di Fourier a DFT comporta anche l'errore di troncamento dovuto all'approssimazione dell'integrale da -∞ a +∞ ad uno tra –τ e + τ ossia un errore dovuto al **windowing leakage.** Il fenomeno di windowing leakage corrisponde alla moltiplicazione della funzione per una funzione detta window function, la quale limita temporalmente il segnale ad un determinato intervallo, ciò comporta che le armoniche ad una determinata frequenza si spalmino su altre frequenze, allontanandosi dal caso ideale, cioè di armoniche nulle in tutte le frequenze tranne la propria.



Di fianco viene mostrato come l'utilizzo di una window function diversa può influenzare il leakage.

La quadratura numerica introduce un errore che limita la banda della funzione, risultando in un segnale a tempo e banda limitata. L’unico segnale così caratterizzato è il segnale identicamente nullo, quindi è impossibile eseguire una trasformata discreta senza commettere uno di questi due errori.

Capitolo IV – FFT

A livello computazionale il calcolo di una DFT comporta numerose operazioni.

Le somme della formula $F_k = \sum_{j=0}^{N-1} f_j e^{-i\,2\pi\,jk\,/N}$ forniscono le componenti di un vettore risultante dal prodotto della matrice di Fourier e il vettore della funzione.

$$\mathbf{F}_N = \begin{bmatrix} \omega_N^{0\cdot 0} & \omega_N^{0\cdot 1} & \dots & \omega_N^{0\cdot (N-1)} \\ \omega_N^{1\cdot 0} & \omega_N^{1\cdot 1} & \dots & \omega_N^{1\cdot (N-1)} \\ \vdots & \vdots & \ddots & \vdots \\ \omega_N^{(N-1)\cdot 0} & \omega_N^{(N-1)\cdot 1} & \dots & \omega_N^{(N-1)\cdot (N-1)} \end{bmatrix} = \left[\omega_N^{jk}\right] \qquad j,k = 0,1,\dots,N-1$$

Matrice di Fourier

Il calcolo diretto di un prodotto matrice-vettore ha complessità computazionale pari a $O(N^2)$. Comunemente il valore di N si attesta a minimo 10^3 e le applicazioni della DFT possono anche essere con funzioni 2D o 3D.

Un metodo importante per la riduzione del numero di calcoli da eseguire è lo schema “butterfly”.

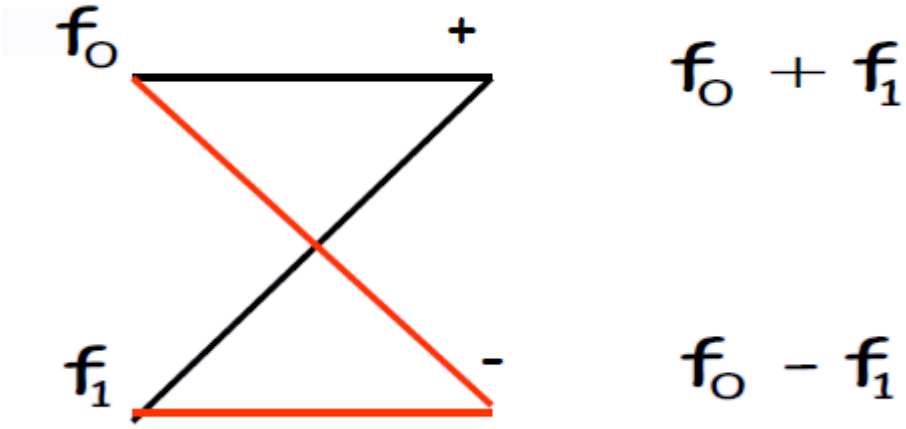
Essendo il vettore di valori da calcolare una combinazione lineare di valori del vettore f, si può riordinare le operazioni tramite schemi di somma e sottrazione, passando dal calcolo di una DFT complessa a quello di n DFT più semplici. Ad esempio, il calcolo di una DFT di lunghezza 2 richiederebbe normalmente 2^2 operazioni, ma rielaborando tramite uno schema di somma e sottrazione si può arrivare al risultato con solo 2 operazioni, preelaborando i valori della matrice di Fourier.

$$F_0 = f_0 w_2^0 + f_1 w_2^0$$
$$F_1 = f_0 w_2^0 + f_1 w_2^1$$

Calcolando i valori di w_2 con la formula $w_2=e^{-i2\pi/2}$ si ottiene

$$F_0 = f_0 + f_1$$
$$F_1 = f_0 - f_1$$

Queste operazioni possono essere schematizzate come segue



La forma di tale schema ricorda quella di una farfalla da cui il nome. Tale schema è utilizzato dall'algoritmo pubblicato nel 1965 da Cooley e Turkey, l'algoritmo FFT o Fast Fourier Transform. Questo algoritmo ha ridotto la complessità a $N\log_2(N)$.

Tenendo in considerazione il caso in cui N è una potenza di 2 pari a 2^τ, la FFT fattorizza la matrice N x N in τ matrici (sempre N x N) tali da minimizzare il numero di moltiplicazioni e addizioni complesse. Il calcolo diretto richiede N^2 moltiplicazioni complesse e N(N-1) addizioni complesse, con questo metodo eseguiamo solo $N\tau/2$ moltiplicazioni complesse e $N\tau$ addizioni complesse.

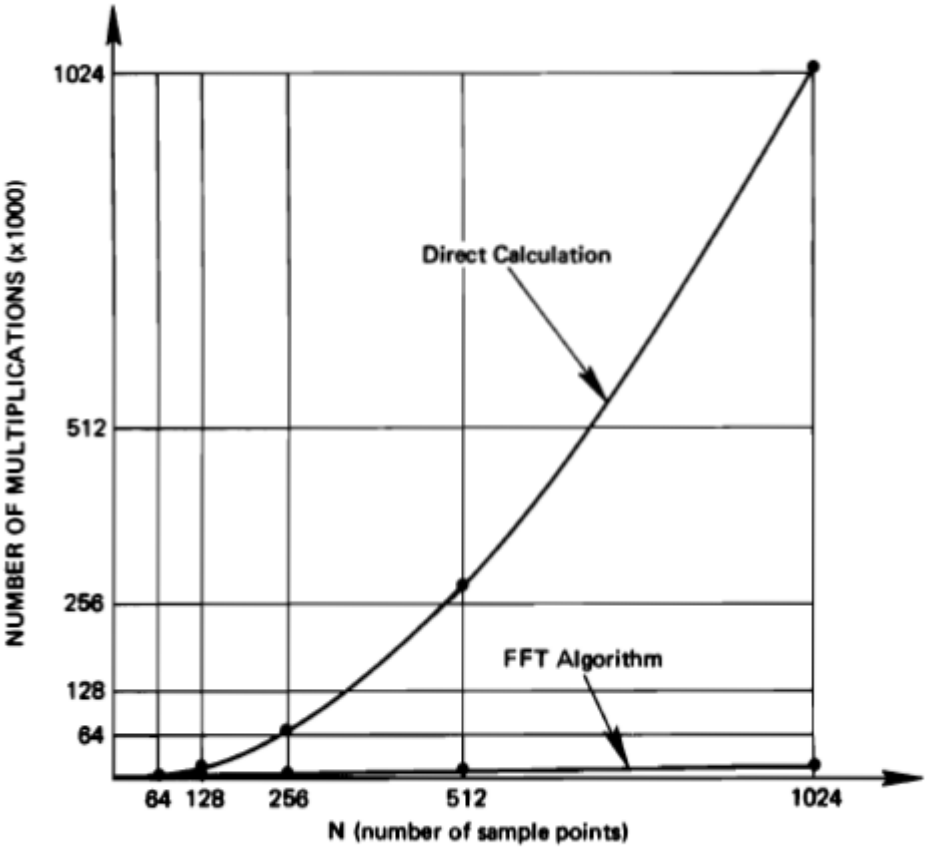
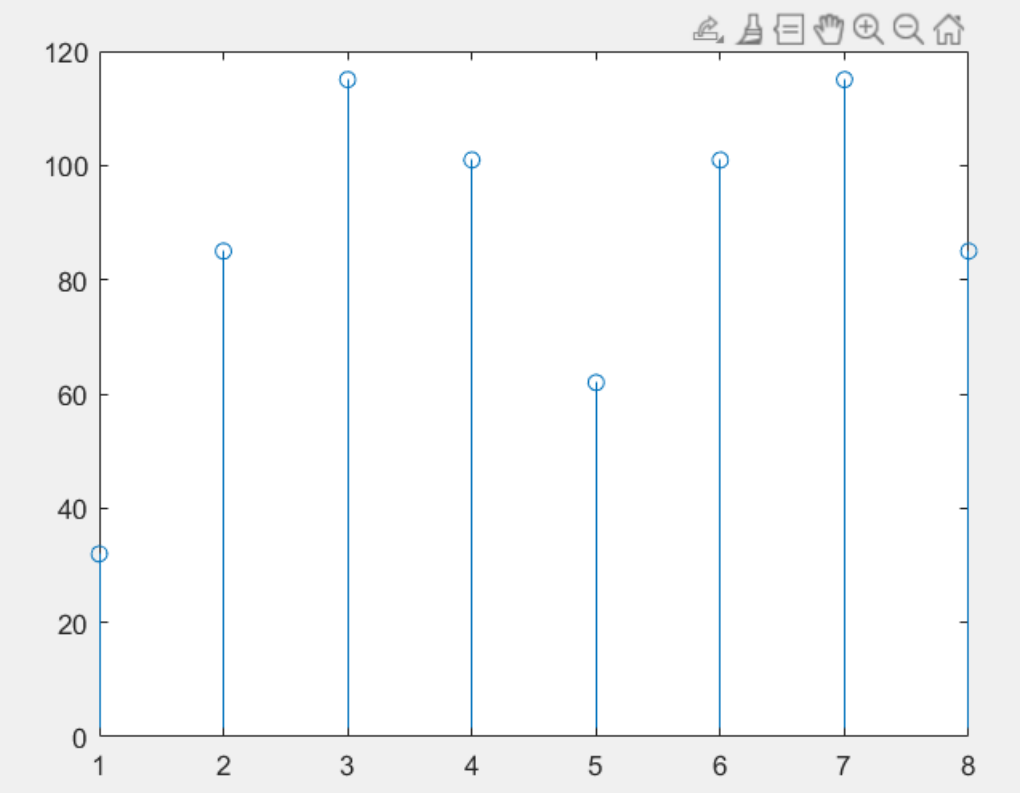


Grafico che illustra come varia il numero di moltiplicazioni in base a se viene applicata la FFT o si esegue il calcolo diretto.

L'algoritmo FFT per casi in cui N è pari a potenza di 2 viene detto radix-2.

```
m = 3;
N = 2^m;
%Applichiamo l'algoritmo FFT -radix2
%Invece di calcolare una DFT di lunghezza N
%l'algoritmo calcolerà N/2 DFT di lunghezza 2

x = randi([-50, 50], N, 1);
y = fft(x);
inverse_y = ifft(y);
```

```
y =                    x =        inverse_y =

   1.0e+02 *

                          32          32.0000
   0.3200 + 0.0000i       41          41.0000
   0.5082 - 0.6815i      -38         -38.0000
   1.0500 + 0.4700i       42          42.0000
  -0.1282 - 1.0015i       13          13.0000
  -0.6200 + 0.0000i      -41         -41.0000
  -0.1282 + 1.0015i      -22         -22.0000
   1.0500 - 0.4700i        5           5.0000
   0.5082 + 0.6815i
```



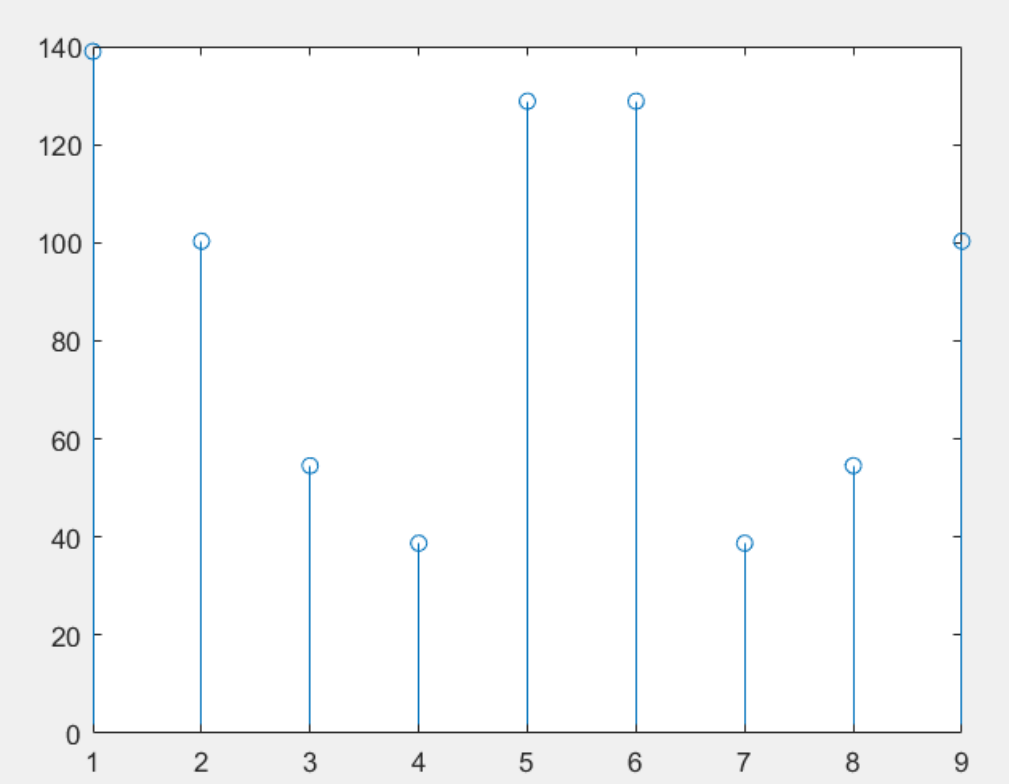
Esistono molte varianti di FFT che tengono conto di casi diversi, ad esempio se N è una potenza di 3 si può utilizzare radix-3.

```
m = 2;
N = 3^m;
%Applichiamo l'algoritmo FFT -radix3
%Invece di calcolare una DFT di lunghezza N
%l'algoritmo calcolerà 3^(m-1) DFT di lunghezza 3

x = randi([-50, 50], N, 1);
y = fft(x);
inverse_y = ifft(y);
stem(abs(y))
%Non specificando l'intervallo di rappresentazione
%viene automaticamente usato l'intervallo [1, length(y)]
```

```
y =

   1.0e+02 *

  -1.3900 + 0.0000i
   0.3239 - 0.9494i
   0.5224 - 0.1570i
   0.2300 - 0.3118i
   0.7437 + 1.0522i
   0.7437 - 1.0522i
   0.2300 + 0.3118i
   0.5224 + 0.1570i
   0.3239 + 0.9494i
```

```
x =

    25
   -11
    16
   -33
    21
   -47
   -23
   -46
   -41
```

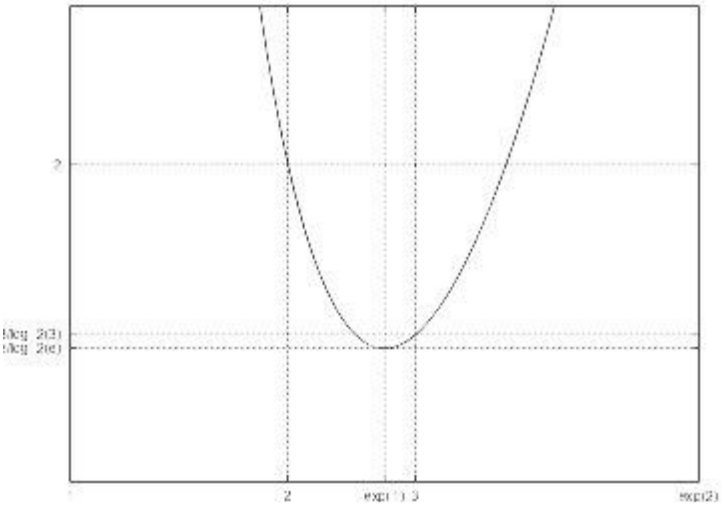
```
inverse_y =

   25.0000
  -11.0000
   16.0000
  -33.0000
   21.0000
  -47.0000
  -23.0000
  -46.0000
  -41.0000
```



Dai valori della y notiamo nel caso con radix-2 che attorno alla frequenza di Nyquist, ossia metà della frequenza di campionamento, la DFT ha modulo con simmetria pari, ossia i valori equidistanti hanno parte reale uguale e parte immaginaria con il segno opposto.
Per radix-3 valgono le stesse valutazioni, manca però un valore alla frequenza di Nyquist che si trova al centro della simmetria.

In base al radix vi è un diverso livello di efficienza, ossia per ogni versione si effettua un numero diverso di calcoli.

In generale per radix-r sappiamo che ha complessità è $O(Nr\log_r(N))$.
$Nr\log_r(N) = N\log_2(N)\frac{r}{\log_2(r)}$ quindi per sapere per quale r l'algoritmo FFT esegue meno operazioni dobbiamo cercare il valore di minimo per $\frac{r}{\log_2(r)}$.

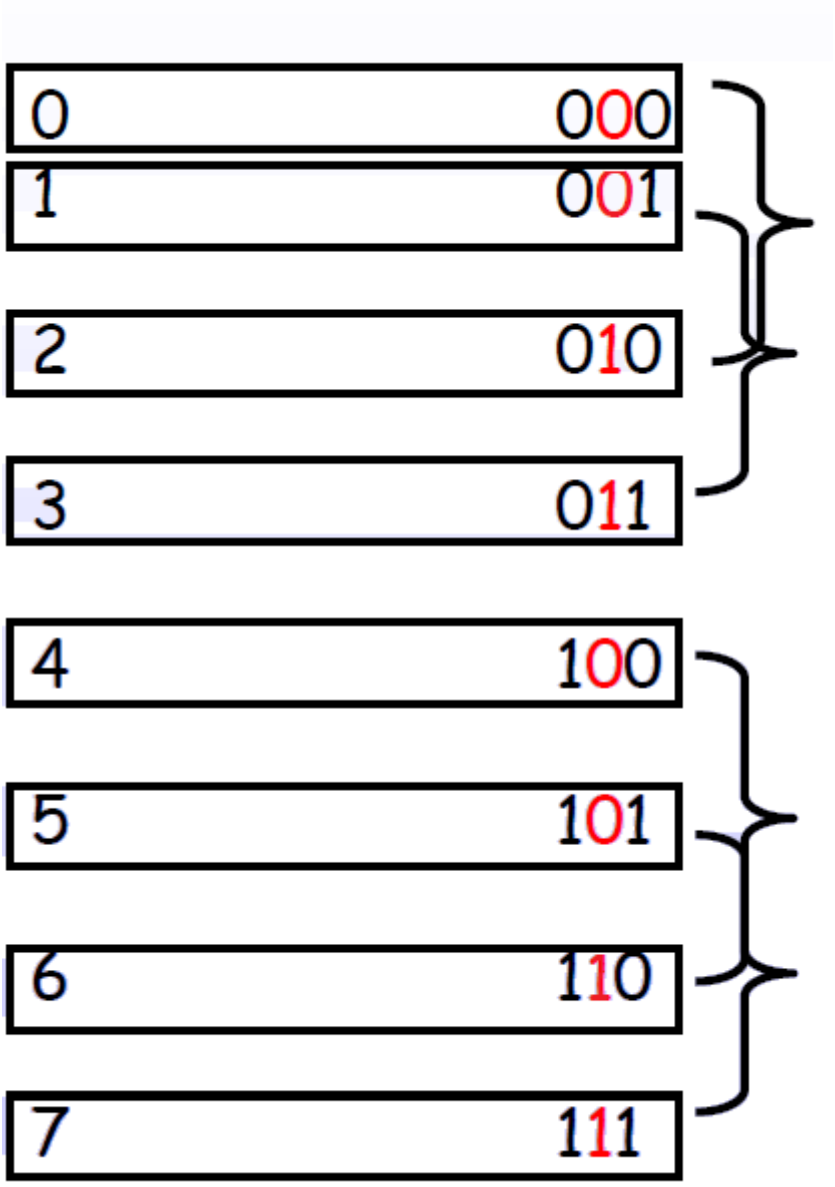


Considerata la funzione $y=\frac{r}{\log_2(r)}$, notiamo che assume valore intero minimo in 3, quindi possiamo considerare radix-3 come il più efficiente tra gli algoritmi radix.

Può capitare che bisogna effettuare una DFT di lunghezza N che non è pari alla potenza di un numero intero primo ma al prodotto tra due numeri primi. In tal caso viene eseguita un FFT Mixed-radix, ossia in cui il processo non è definito da un singolo valore fissato. Con radix-3 possiamo dividere 1 DFT di

lunghezza 9 in 3 DFT di lunghezza 3, cosa che non possiamo fare nel caso N=12. In tal caso possiamo dividere la DFT in 4 DFT di lunghezza 3, sulle quali possiamo applicare radix-3, o in 3 DFT di lunghezza 4, sulle quali possiamo applicare radix-2.

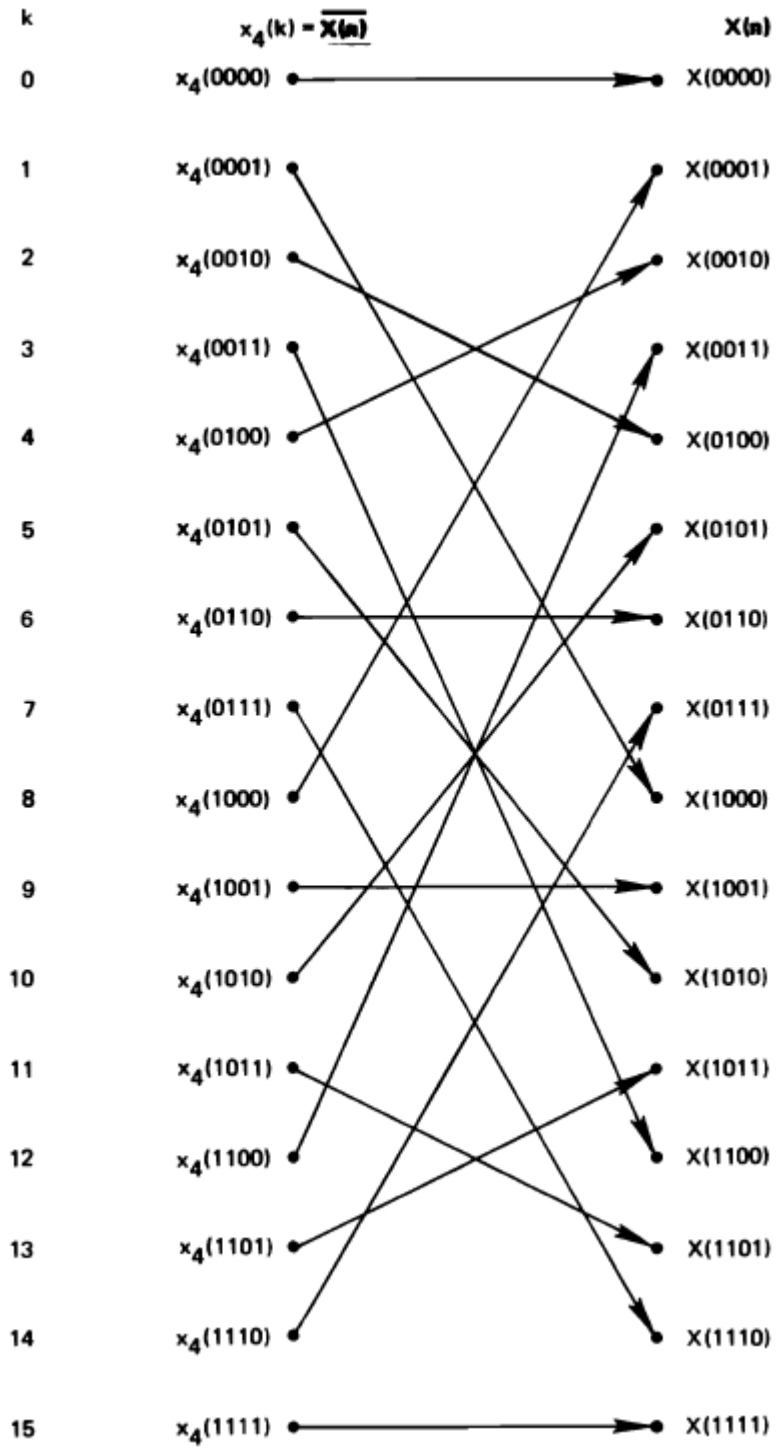
Un modo che ha l'algoritmo radix-2 per trovare le coppie da inserire in uno schema butterfly, senza dover fare un grosso lavoro di preelaborazione, è quello di valutare la differenza nei bit. Considerando 8 valori di f, in base alla rappresentazione binaria posso trovare con facilità le coppie, ad esempio il valore 0, in binario 000, e il valore 4, in binario 100, differiscono solo per il primo bit quindi considero come prima coppia f_0 e f_4. I vari passaggi si eseguono spostando il bit preso come riferimento, al primo passaggio si considera il primo bit, al secondo si considera il bit successivo e così via.



Nel caso a fianco esplorato avremo 3 iterazione trattandosi di 2^3 valori, quindi ci saranno 3 bit da considerare, uno per iterazione.

L'utilizzo della FFT comporta anche un passaggio finale ossia l'unscrambling. I risultati trovati dalle operazioni viste sopra non sono in ordine rispetto al vettore di output desiderato, per risolvere ciò bisogna convertire la posizione di un determinato nodo in binario, per poi invertire i bit di esso. Il numero binario ottenuto indicherà il numero del nodo con il quale verrà scambiato. Per fare un esempio il nodo f_0 non cambia di posto in quanto 000 inverso rimane lo stesso mentre nodo f_1 si scambia con f_4 in quanto 001 al contrario è 100. Con questa ultima operazione il vettore di output ottenuto rappresenta una trasformata

discreta del segnale di partenza.

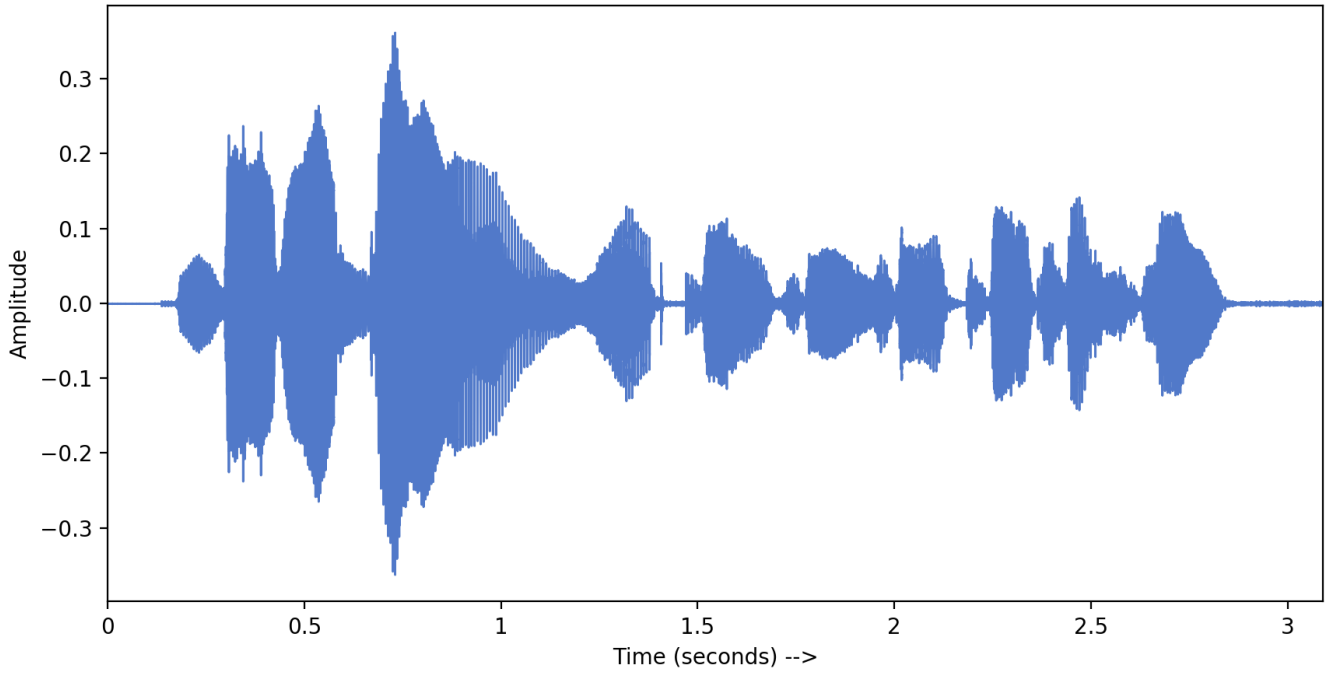


Esempio di Scrambling con N=16

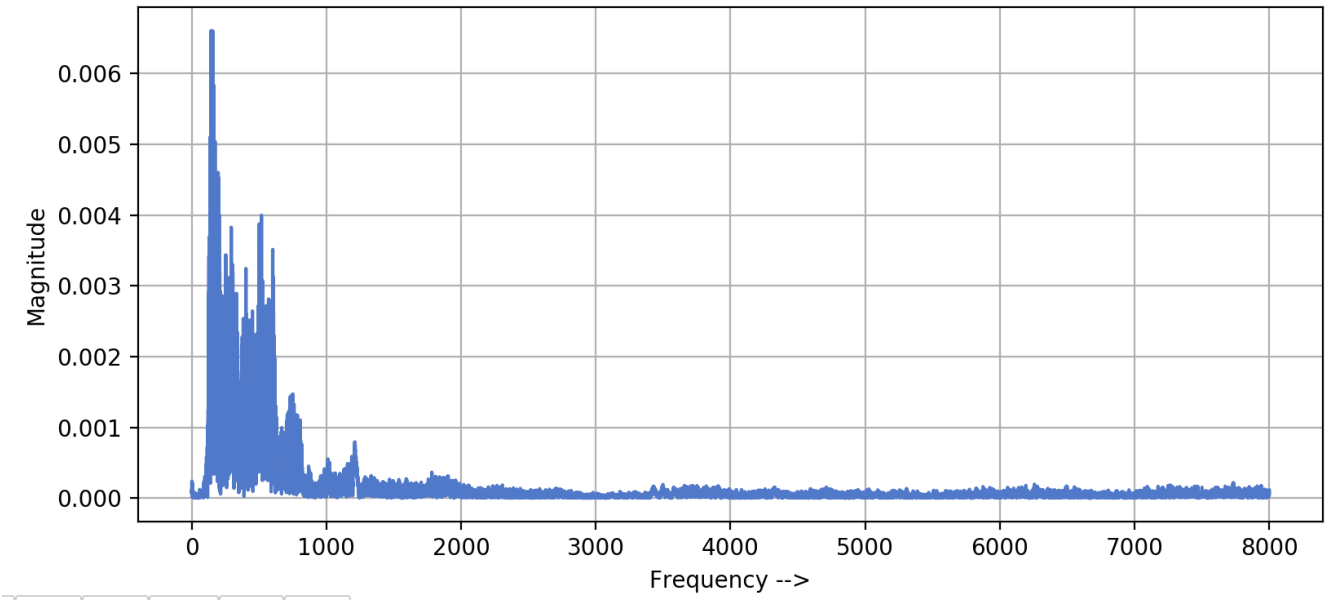
Capitolo V – Applicazioni di FFT su segnali audio

I calcolatori moderni presentano grandi possibilità per la manipolazione di file audio, tutto ciò grazie all'utilizzo delle DFT, poiché lavorare in frequenza è fondamentale per i segnali audio. Dall'elaborazione real-time di programmi come Ableton a semplici editor come Audacity, la trasformata di Fourier è un passaggio alla base di quasi tutte le funzionalità da essi offerte.

Un segnale audio è considerato come una somma di sinusoidi, ognuna ad una determinata frequenza e con una propria ampiezza.

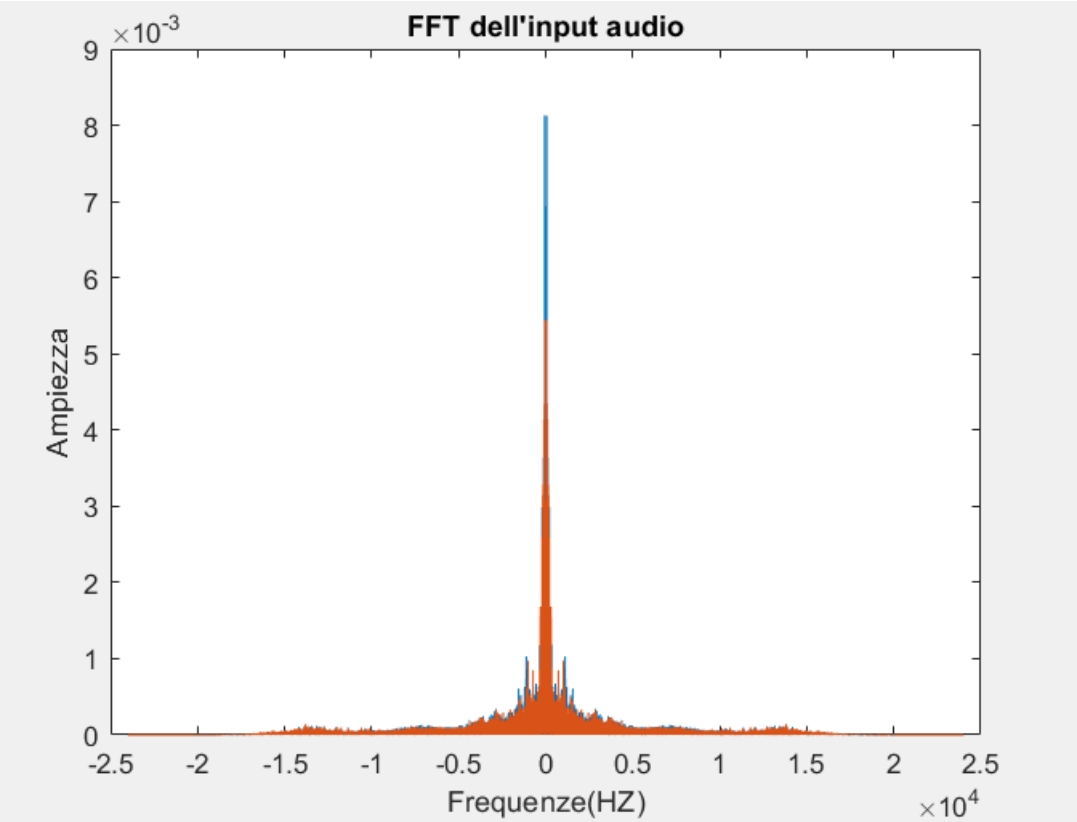


Analizzandolo in frequenza, nel caso discreto, queste sinusoidi si esprimono con una determinata ampiezza in un bin, un intervallo frequenziale che dipende dalla frequenza di campionamento.



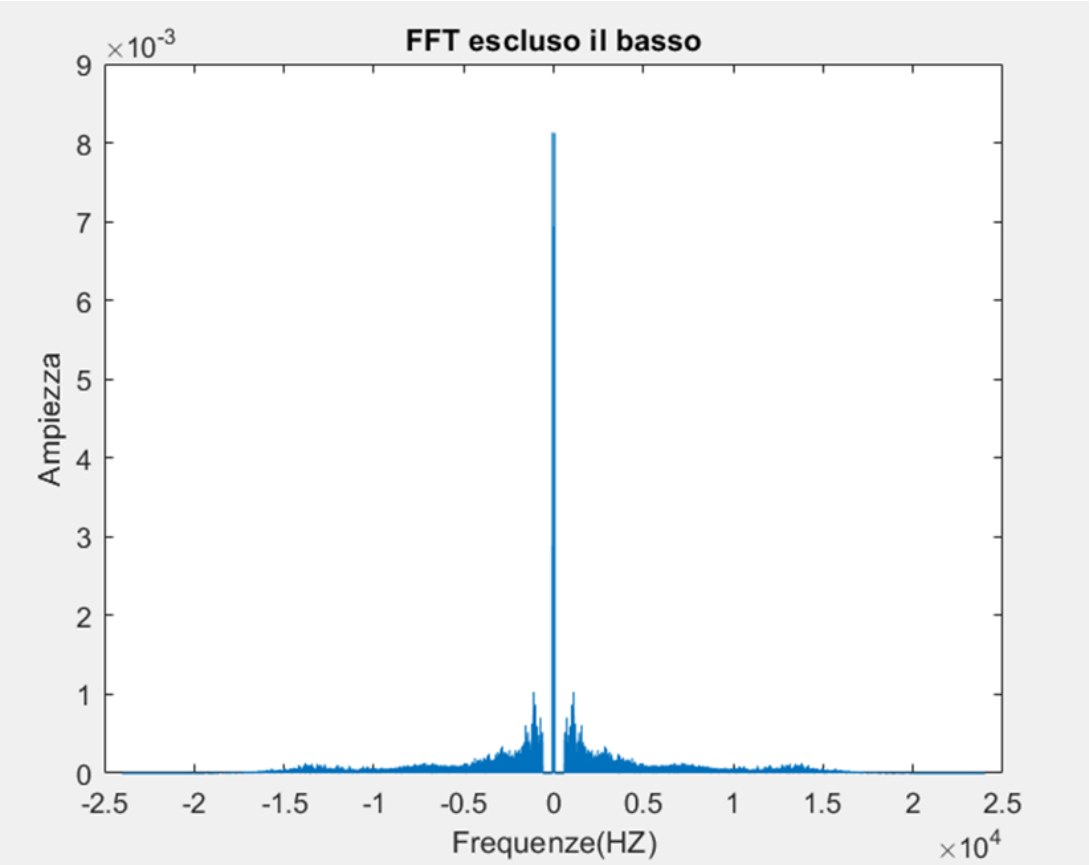
Le frequenze audio sono suddividibili in tre macrocategorie: bassi, medi e alti. Le frequenze audio si susseguono attraversando in ordine queste categorie consentendoci di visualizzarle in ordine e manipolarle per ottenere degli effetti. Ad esempio amplificando le frequenze audio a sinistra dello spettro del segnale, ossia la sua rappresentazione in frequenza, otteniamo un amplificazione del basso, con effetti uditivi evidenti. Gli strumenti di equalizzazione non fanno altro che modificare i vari bin di frequenza per accentuare o attenuare determinate componenti dell'audio. Per fare ciò possiamo ad esempio moltiplicare la trasformata per una funzione definita a tratti, andando a moltiplicare ogni bin per il valore che si desidera. Nel caso dell'elaborato sono state sfruttate le potenzialità di MATLAB per isolare un determinato strumento da un traccia musicale.

L'elaborato esegue la FFT del segnale d'ingresso, un file audio di una canzone, facendoci ottenere lo spettro.

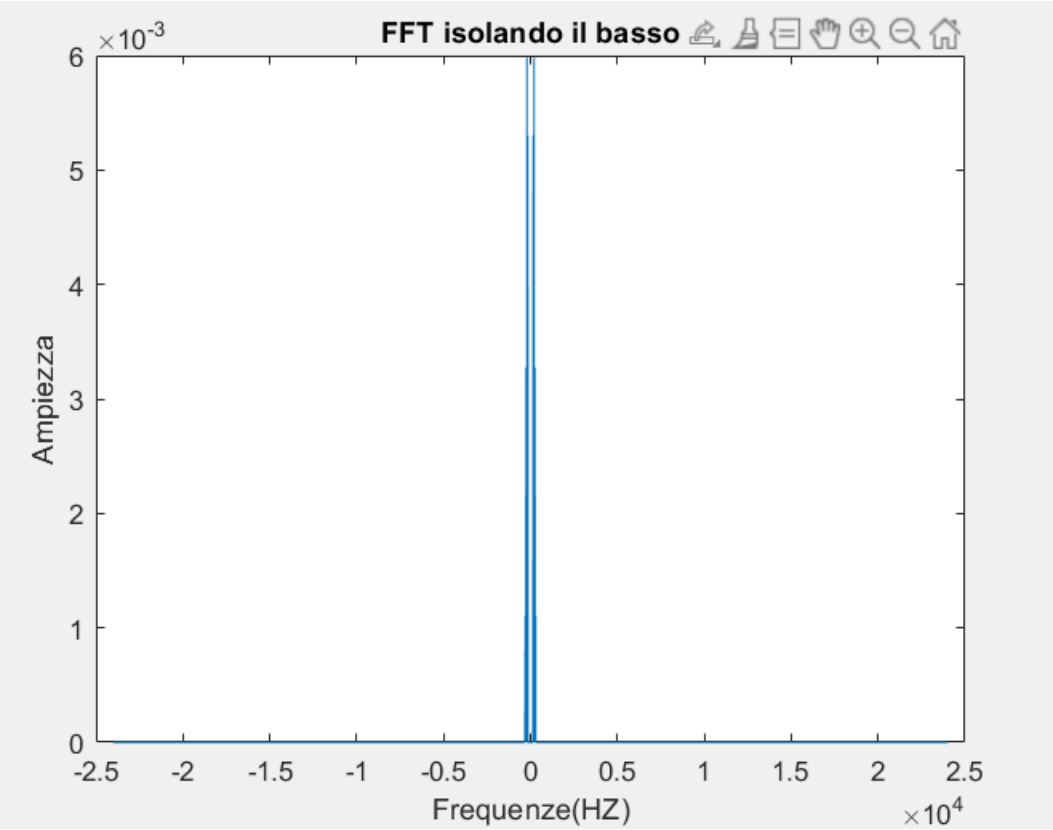


Come si può osservare il grafico presenta la frequenza 0 Hz al centro, mostrandoci anche come la FFT sia pari.

I due colori mostrano i contenuti informativi dei due canali audio, essendo il file d'origine in stereo. Si potrà notare nei prossimi grafici solo il colore blu in quanto è stato preso solo un canale per i calcoli.



In questo spettro possiamo osservare la modifica apportata alla trasformata del segnale, in cui, con una condizione in logica booleana, si è stabilito quale intervallo di frequenza lasciare inalterato e quale annullare. Le frequenze tra 60 e 600Hz, identificabili come frequenze che contengono l'audio del basso, sono state annullate, in modo da privare la canzone della parte di basso.



Qua invece è mostrato il caso in cui isoliamo il basso, cioè prendiamo solo frequenze tra 150 Hz e 300 Hz.

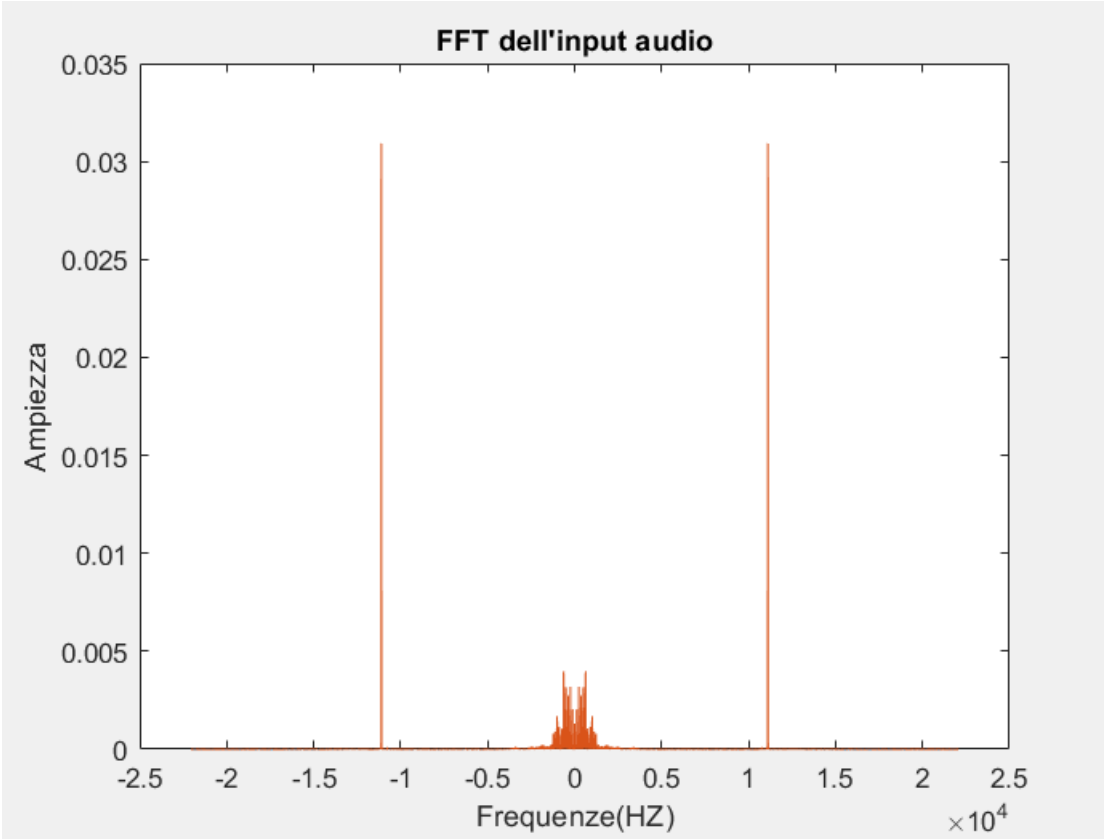
Di seguito è mostrato il codice utilizzato per queste operazioni.

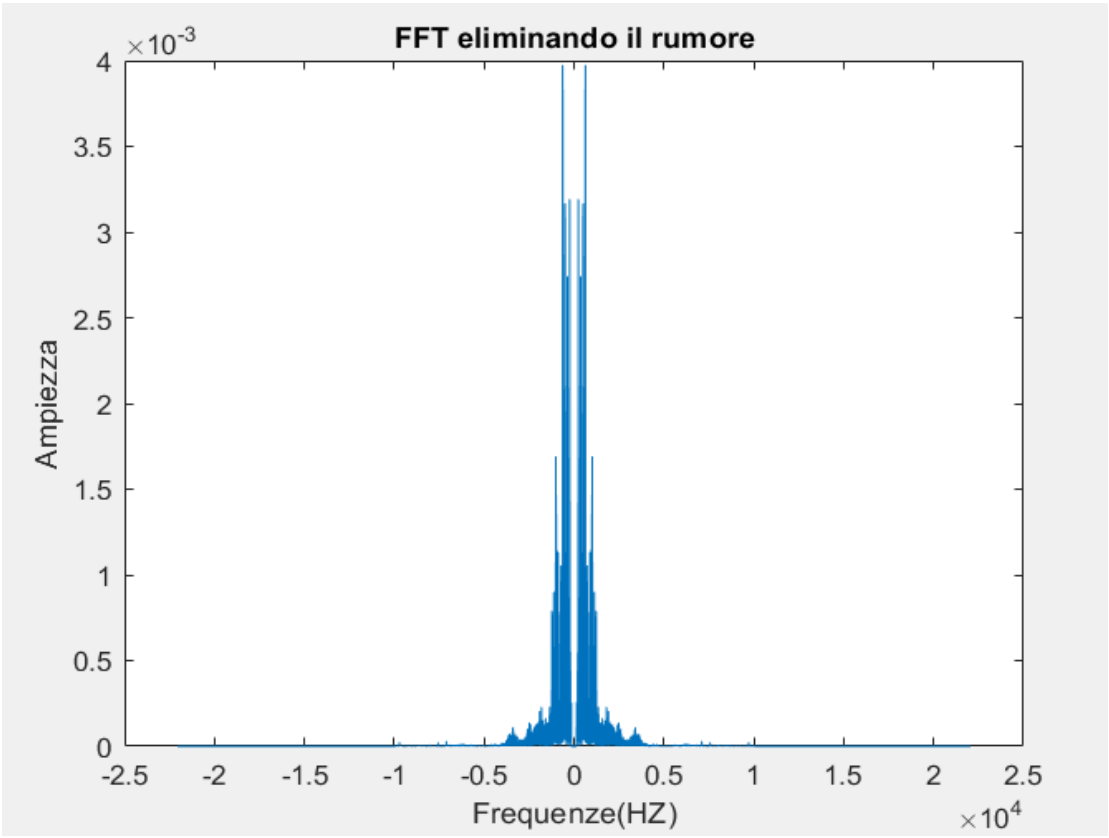
```
% Questo esempio mira a dimostrare come manipolare un segnale nel
% dominio della frequenza per isolare alcune frequenze dalle altre.
% Queste tecniche sono molto utili in ambito musicale per isolare
% la voce o gli strumenti di una canzone. Qui vogliamo isolare l'iconica
% linea di basso della canzone Another One Bites The Dust dei Queen.
% Lo script crea due player, uno per ascoltare solo il basso, l'altro
% per ascoltare il brano senza basso.
% Usage: play(player_ins) per ascoltare il brano senza basso
%        stop(player_ins) per interrompere la riproduzione
%        ------------------------------------------------
%        play(player_bass) per ascoltare solo il basso
%        stop(player_bass) per interrompere la riproduzione


[audio_in,Fs] = audioread('Another One Bites The Dust.mp3');

%df è il range minimo di frequenze
n = length(audio_in);
df = Fs/n;

%Scegliamo i valori di frequenze che comporranno le ascisse del grafico
frequenze = -Fs/2:df:Fs/2-df;

%Applichiamo la trasformata di Fourier
FFT_audio_in = fftshift(fft(audio_in)/length(fft(audio_in)));

%Plotting della trasformata
figure
plot(frequenze,abs(FFT_audio_in));
title("FFT dell'input audio");
xlabel('Frequenze(HZ)');
ylabel('Ampiezza');

%Range di frequenze per isolare il basso ed il resto del brano
freq_taglio_inf_ins = 60;
freq_taglio_sup_ins = 600;

freq_taglio_inf_bass = 150;
freq_taglio_sup_bass = 300;

%Tramite operazioni booleane applichiamo un filtro passa-banda
val_ins = abs(frequenze)<freq_taglio_sup_ins & abs(frequenze)>freq_taglio_inf_ins;
val_bass = abs(frequenze)<freq_taglio_sup_bass & abs(frequenze)>freq_taglio_inf_bass;

FFT_ins = FFT_audio_in(:,1);        %Prendiamo un solo canale (essendo stereo)
FFT_bass = FFT_audio_in(:,1);
FFT_ins(val_ins) = 0;
FFT_bass(~val_bass) = 0;
FFT_bass(val_bass) = FFT_bass(val_bass)*2;

%Plotting dei segnali dopo l'applicazione del filtro
figure
plot(frequenze,abs(FFT_ins));
title("FFT escluso il basso");
xlabel('Frequenze(HZ)');
ylabel('Ampiezza');
```

```
figure
plot(frequenze,abs(FFT_bass));
title("FFT isolando il basso");
xlabel('Frequenze(HZ)');
ylabel('Ampiezza');

%Applichiamo la trasformata inversa di Fourier
FFT_a = ifftshift(FFT_audio_in);
FFT_a11 = ifftshift(FFT_ins);
FFT_a31 = ifftshift(FFT_bass);

%Creiamo i segnali nel dominio del tempo
s1 = ifft(FFT_a11*length(fft(audio_in)));
s3 = ifft(FFT_a31*length(fft(audio_in)));

%Creiamo i player
player_ins = audioplayer(s1, Fs);
player bass = audioplayer(s3, Fs);
```

Un’ultima applicazione delle FFT che è stata esplorata è la cancellazione del rumore, nel caso del file provato vi era in sottofondo un fastidioso rumore ad alta frequenza, il quale simula un fischietto per cani. Eliminando le componenti frequenziali più alte si può ridurre considerevolmente il rumore ottenendo una perdita di contenuto informativo accettabile.

Di seguente sono mostrati i due spettri del segnale prima e dopo la cancellazione del rumore.


FFT dell'input audio
Ampiezza
0.035
0.03
0.025
0.02
0.015
0.01
0.005
0
-2.5
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2
2.5
Frequenze(HZ)
×10^4


FFT eliminando il rumore
×10^-3
Ampiezza
4
3.5
3
2.5
2
1.5
1
0.5
0
-2.5
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2
2.5
Frequenze(HZ)
×10^4