UNIVERSITA' DEGLI STUDI DI
NAPOLI FEDERICO II

Scuola Politecnica e delle Scienze di Base

Corso di Laurea Magistrale in Ingegneria Informatica

Elaborato **Calcolo Numerico**

Fast Fourier Transform (FFT): Analisi ed Applicazioni

Anno Accademico 2020/21

Studenti:

Michele Maresca M63/1151

Vincenzo Riccardi M63/1146

Marco Carlo Feliciano M63/1136

Indice

Introduzione

Lo scopo dell'elaborato è quello di mostrare alcuni esempi di applicazione della Fast Fourier Transform.
Inizialmente sarà introdotta la Trasformata di Fourier nel dominio del tempo continuo (FT) e la sua versione discretizzata (DFT) al fine di poterla utilizzare nelle applicazioni.
Successivamente viene descritto come è effettuata la discretizzazione della Trasformata di Fourier mediante formule di quadratura.
Inoltre, sono introdotti l'algoritmo di Cooley e Tukey e l'algoritmo di Gentleman e Sande, ovvero gli algoritmi Fast Fourier Transform (FFT), per il calcolo della Trasformata Discreta di Fourier (DFT).
In conclusione, vengono descritte le applicazioni della FFT per l'elaborazione dei segnali multimediali e viene effettuato il confronto dei tempi di esecuzione della FFT rispetto alla DFT.

1. Dalla Trasformata di Fourier continua alla Trasformata Discreta di Fourier

Nel seguente capitolo daremo uno sguardo alla trasformata di Fourier nel tempo continuo e discreto toccando l'analisi numerica degli integrali per capire come si è giunti alla DFT.

1.1. Trasformata di Fourier tempo continuo

Cosa significa che un segnale è rappresentato nel dominio del tempo? Significa che la sua base matematica è un impulso temporale, e quindi ogni segnale è esprimibile mediante una combinazione lineare di impulsi temporali centrati in diversi punti e che presentano determinate ampiezze. In questo modo si ottiene una localizzazione perfetta nel tempo. Cosa significa rappresentare un segnale nel dominio della frequenza? Significa che la sua base matematica è un'onda sinusoidale, e quindi un segnale rappresentato in frequenza è esprimibile mediante una combinazione lineare di onde sinusoidali caratterizzate da una certa frequenza, una certa fase iniziale ed una certa ampiezza. In questo modo si ottiene una localizzazione perfetta in frequenza, ovvero sappiamo quali sinusoidi sono "contenute" in quel segnale, ma abbiamo perso l'informazione temporale. Siamo abituati a lavorare con segnali espressi nel domino del tempo, ma in alcuni casi ci interessa fare una "analisi spettrale" dei segnali, andando a vedere di quali sinusoidi il segnale da analizzare è composto. Lo strumento matematico che ci permette di passare dalla rappresentazione nel tempo alla rappresentazione in frequenza è la trasformata di Fourier. L'anti-trasformata ci permette, dato un segnale espresso nel domino della frequenza, di ritornare nel dominio del tempo. Entrambe le trasformazioni si esprimono in forma integrale.

$$X(f) = \int_{-\infty}^{+\infty} x(t) e^{-j2\pi f t} dt$$

$$x(t) = \int_{-\infty}^{+\infty} X(f) e^{j2\pi f t} df$$

1.2. Trasformata di Fourier tempo discreto

Per poter utilizzare la Trasformata di Fourier nelle applicazioni numeriche è necessaria una discretizzazione, viene introdotta allora la Trasformata Discreta di Fourier.

Considerando un segnale tempo discreto x(n), le due equazioni relative alla trasformata diretta e inversa sono:

$$X(v) = \sum_{n=-\infty}^{+\infty} x(n) e^{-j2\pi vn}$$

$$x(n) = \int_{1} X(v) e^{j2\pi vn}\, dv$$

Dove v è la frequenza misurata in numero di cicli/campione. La prima è detta equazione di analisi (Trasformata di Fourier diretta), essa permette di determinare il peso, in termini di ampiezza e fase, che le varie componenti sinusoidali hanno nella ricostruzione del segnale. La seconda è detta formula di sintesi (Trasformata di Fourier inversa), mostra come x(n) possa essere rappresentato come combinazione lineare di esponenziali complessi, con la frequenza che varia con continuità su intervalli di durata unitaria.

Poiché la Trasformata di Fourier è continua nella variabile v, risulta difficile l'analisi e l'elaborazione numerica in frequenza di un segnale tempo discreto.

È possibile però, definire la trasformata Discreta di Fourier diretta (DFT) e inversa (IDFT) per un segnale x(n) di durata N nel seguente modo:

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{\frac{-j2\pi kn}{N}} \quad k = 0, \dots, N-1$$

$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k) e^{\frac{j2\pi kn}{N}} \quad n = 0, \dots, N-1$$

In questo modo un segnale di durata finita N è rappresentato in frequenza esattamente con N campioni complessi.

Notare che, in generale, se consideriamo un segnale periodico di periodo T, la quantità $\Delta f = 1/T$ è detta frequenza fondamentale, ovvero è la minima distanza alla quale si devono trovare due frequenze per distinguerle nel campionamento del segnale; nel caso della DFT, abbiamo un intervallo di sample ΔT, e quindi si ha $T = N * \Delta T$, dove N è il numero di campioni. A tal punto la frequenza fondamentale la possiamo riscrivere come f_s / N, dove f_s è la frequenza di sample e dunque f_s / N è la frequenza più bassa che si può rappresentare effettuando su N campioni la DFT di un segnale periodico di periodo T; applicando il teorema di Shannon, la frequenza più alta che si può rappresentare è invece $f_s / 2$, detta Frequenza di Nyquist. Dunque in generale le frequenze rappresentabili variano nell'intervallo $[f_s / N, f_s / 2]$.

Un'altra osservazione che si può fare è che è possibile rappresentare una DFT come una trasformazione lineare, cioè come il prodotto di una matrice per un vettore.

Infatti, il calcolo degli N coefficienti della DFT esplicitandola, risulta:

$$X(0) = x(0) + x(1) + \ldots + x(N-1)$$

$$X(1) = x(0) + x(1)e^{-j2\pi/\mathrm{N}} + \ldots + x(N-1)e^{-j2\pi(\mathrm{N}-1)/\mathrm{N}}$$

$$\ldots = \ldots$$

$$X(N-1) = x(0) + x(1)e^{-j2\pi(\mathrm{N}-1)/\mathrm{N}} + \ldots + x(N-1)e^{-j2\pi(\mathrm{N}-1)(\mathrm{N}-1)/\mathrm{N}}$$

Detti $\boldsymbol{X} = [X(0), X(1), \ldots, X(N-1)]^{\mathbf{T}}$ e $\boldsymbol{x} = [x(0), x(1), \ldots, x(N-1)]^{\mathbf{T}}$ i vettori colonna, e **W** la seguente matrice:

$$\boldsymbol{W} = \begin{bmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & e^{-j2\pi/\mathrm{N}} & e^{-j4\pi/\mathrm{N}} & \ldots & e^{-j2\pi(\mathrm{N}-1)/\mathrm{N}} \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ 1 & e^{-j2\pi(\mathrm{N}-1)/\mathrm{N}} & e^{-j4\pi(\mathrm{N}-1)/\mathrm{N}} & \ldots & e^{-j2\pi(\mathrm{N}-1)(\mathrm{N}-1)/\mathrm{N}} \end{bmatrix}$$

Risulta che

$$\boldsymbol{X} = \boldsymbol{W}\boldsymbol{x}$$

Questa osservazione ci suggerisce immediatamente che la complessità della DFT è $O(n^2)$.

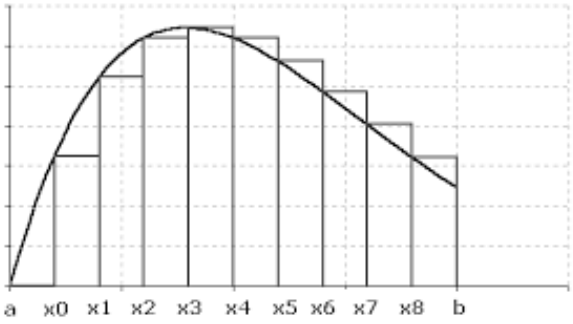
1.3. Integrazione numerica

Il calcolo della Trasformata di Fourier, come precedentemente accennato, richiede il calcolo di un integrale.

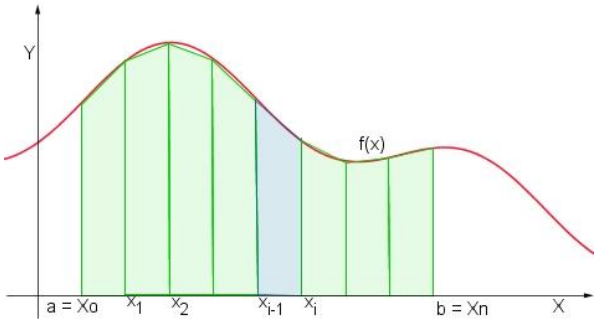
L'approccio più naturale per il calcolo di un integrale è quello di approssimare l'integrale continuo $I[f] = \int_a^b f(x)\ dx$ con combinazioni lineari della funzione integranda *f(x)*. Le formule che realizzano tali approssimazioni sono dette formule di quadratura.

Alcuni esempi notevoli sono:

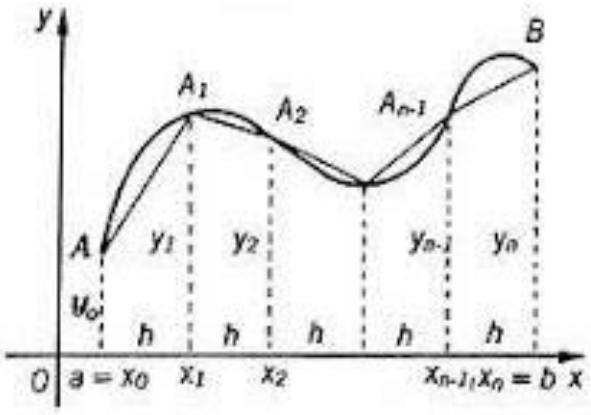
- Il metodo dei rettangoli



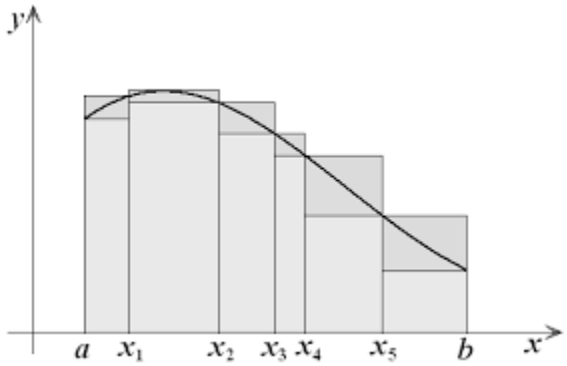
- Il metodo dei trapezi



- Il metodo di Simpson



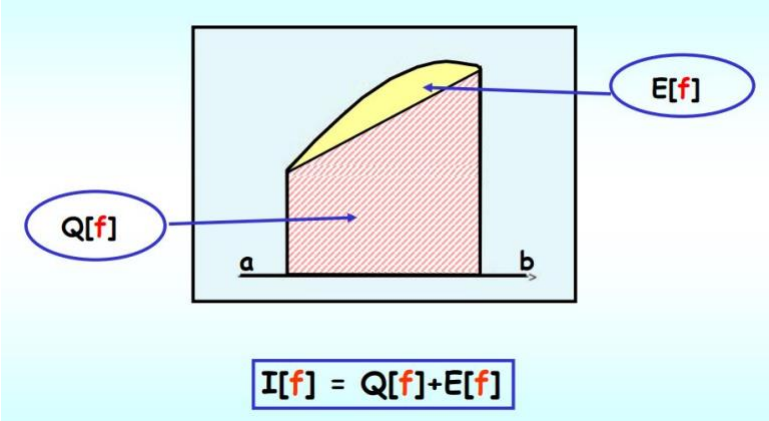
- Le somme di Riemann



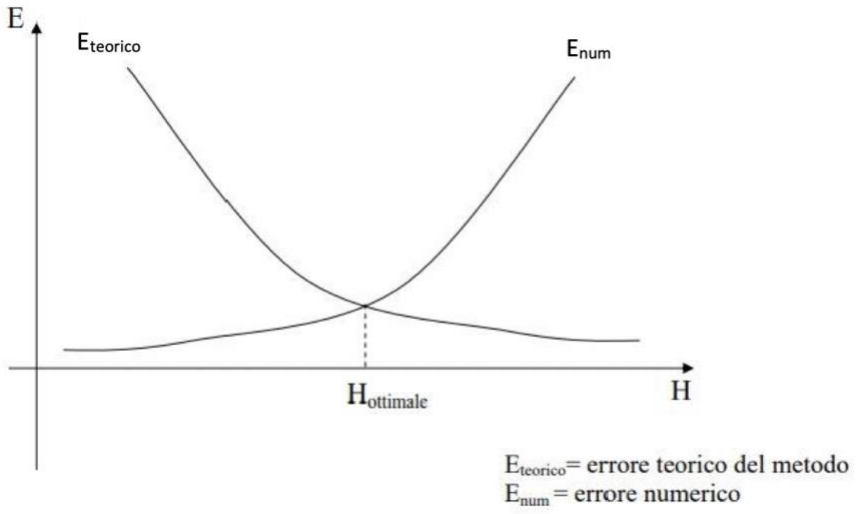
Per citare i principali metodi.

Come già detto, esse sono tutte combinazioni lineari della funzione integranda $Q[f] = A_1 f(x_1) + A_2 f(x_2) + \cdots + A_n f(x_n) \cong I[f]$, dove i coefficienti A_i sono detti pesi, mentre le ascisse x_i sono dette nodi. A tal punto vorremmo essere in grado di valutare l'errore che si commette usando una determinata formula di integrazione numerica Q[f]. La differenza E[f] = I[f] - Q[f] è detta errore di discretizzazione della formula di quadratura Q[f]. È facile farne un'interpretazione geometrica, considerando il concetto di

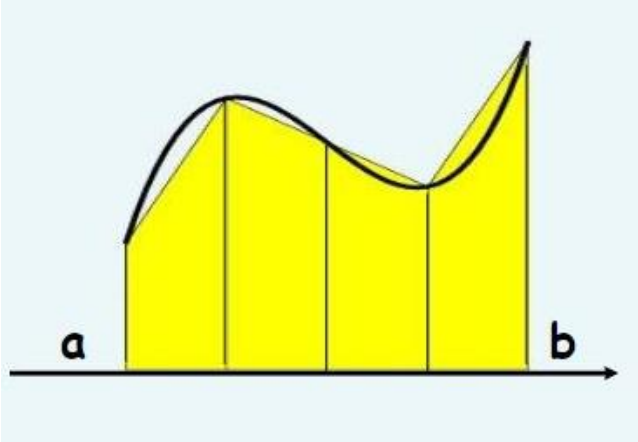
area. È chiaro che l'errore che si commette può avere segno positivo o negativo.



Mediante degli esempi, si nota velocemente che raddoppiando il numero di nodi d'integrazione, l'errore di discretizzazione della formula si riduce di circa quattro volte. Nella pratica però non si può aumentare di troppo il numero di nodi, perché poi si aumenterebbe l'errore numerico. Di seguito è ripotato un grafico che aiuta per sommi capi a comprendere come l'errore teorico e l'errore numerico si muovono al variare dei nodi chiamati H nella rappresentazione, l'obbiettivo è trovare un numero di nodi tali che l'errore numerico e teorico siano ben bilanciati.



Una formula di integrazione abbastanza semplice ma che permette di ottenere un errore di discretizzazione che converge a 0 al crescere del numero di nodi (a meno di errori di calcolo numerico) è la formula trapezoidale composita.



In pratica si suddivide l'intervallo [a, b] in m sotto intervalli di uguale ampiezza $h = \frac{(b-a)}{m}$, ottenendo quindi m + 1 nodi $t_0, t_1 \dots t_n$. L'area di ciascun sotto-intervallo è pari all'area di un trapezio avente come due vertici "*superiori*" i nodi del sotto intervallo (con le relative ordinate), e quindi nella pratica per semplicità di implementazione si considera un rettangolo avente altezza pari alla media delle ordinate dei due nodi del sotto-intervallo. Avendo fatto questa osservazione sull'implementazione, è facile scrivere la formula:

$$T_m = \frac{h}{2} * (f(t_0) + f(t_1)) + \frac{h}{2} * (f(t_1) + f(t_2)) + \cdots + \frac{h}{2} * (f(t_{m-1}) + f(t_m)) =$$

$$= h \sum_{1}^{m-1} f(t_j) + h * \frac{f(t_m) + f(t_0)}{2}$$

Si può dimostrare che la famiglia Tm[f] è convergente, ovvero dato $E_m = I[f] - T_m[f]$, si ha che:

$$\lim_{m \to \infty} E_m[f] = 0$$

C’è solo una incognita che resta indeterminata: l’errore di discretizzazione è stato definito come $E[f] = I[f] - Q[f]$, ma *I[f]* non è noto (altrimenti non ci porremmo il problema dell’integrazione numerica). Dunque, vorremmo determinare una stima calcolabile dell’errore di discretizzazione $E_m[f]$, facendo riferimento alla formula trapezoidale composita $T_m[f]$. Abbiamo che:

$$E_m = I[f] - T_m[f]$$

$$E_{2m} = I[f] - T_{2m}[f]$$

$$\rightarrow E_{2m} - E_m = T_m[f] - T_{2m}[f]$$

Precedentemente abbiamo detto che raddoppiando il numero di nodi l’errore diventa circa un quarto, dunque possiamo porre $E_m = 4 * E_{2m}[f]$. Sostituendo questa espressione in quella precedentemente trovata e passando ai valori assoluti, si ha:

$$|T_m[f] - T_{2m}[f]| = |4 * E_{2m}[f] - E_{2m}[f]| = |3 * E_{2m}[f]|$$

E dunque abbiamo ottenuto una stima calcolabile dell’errore di discretizzazione che si commette usando la formula d’integrazione trapezoidale composita:

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

1.4. Discretizzazione Fourier con formula di quadratura (FT to DFT)

In questa sezione ci poniamo il problema di come passare dalla trasformata di Fourier tempo continuo alla Trasformata Discreta di Fourier, sfruttando le basi di integrazione numerica del paragrafo precedente.

Per iniziare consideriamo l'integrale nel tempo continuo:

$$\int_{-\infty}^{+\infty} f(t)e^{-2\pi i\omega t}dt$$

dove $f : R \rightarrow R$ è una funzione per cui tale integrale esiste ed è finito. Al variare di ω l'integrale definisce una funzione:

$$F : \omega \rightarrow \int_{-\infty}^{+\infty} f(t)e^{-2\pi i\omega t}dt$$

a cui si dà il nome di Trasformata di Fourier della funzione *f*. Sia $T \in R - [\infty]$, si ha:

$$F(\omega) = \int_{-\infty}^{+\infty} f(t)e^{-2\pi i\omega t}dt = \sum_{k=-\infty}^{+\infty} \int_{kT}^{(k+1)T} f(t)e^{-2\pi i\omega t}dt =$$

$$= \sum_{k=-\infty}^{+\infty} \int_{0}^{T} f(t+kT)e^{-2\pi i\omega(t+kT)}dt = \int_{0}^{T} \sum_{k=-\infty}^{+\infty} f(t+kT)e^{-2\pi i\omega(t+kT)}dt$$

Se introduciamo la funzione f_p, periodica di periodo *T*, definita come:

$$f_p(t) = \sum_{k=-\infty}^{+\infty} f(t+kT), \quad t \geq 0$$

segue che:

$$F(\omega) = \int_{-\infty}^{+\infty} f(t)e^{-2\pi i\omega t}dt = \int_0^T f_p(t)e^{-2\pi i\omega t}dt$$

ovvero la Trasformata di Fourier di *f* coincide con la Trasformata di Fourier di f_p.

Supponiamo di voler valutare la funzione *F(ω)* per $\omega = \omega_n = n/T$, $n = 0, 1, 2, \ldots$:

$$F(\omega_n) = \int_0^T f_p e^{-2\pi i\omega_n t}dt$$

e discretizziamo l'integrale con la formula di quadratura trapezoidale composita su N punti. Posto h = T/N si ottiene:

$$F(\omega_n) = h\sum_{k=0}^{N-1} f_p(kh)e^{-2\pi ikh\frac{n}{T}} = h\sum_{k=0}^{N-1} f_p(kh)e^{-\frac{2\pi}{N}ikn}$$

Considerati i vettori $F = \{F(\omega_n)\}_{n=0,1,\ldots,N-1}$, *e* $f_p = \{f_p(kh)\}_{k=0,1,\ldots,N-1}$, segue che

$$F = DFT[f_p]$$

ovvero, la Trasformata discreta di Fourier, costruita a partire dal vettore f_p, fornisce un'approssimazione dei valori assunti dalla Trasformata di Fourier della funzione f_p in un insieme di punti equidistanziati. In particolare, se f è nulla al di fuori dell'intervallo [0, T], allora $f_p(v) = f(v)$, $v \in [0, T]$ e la $DFT[f_p]$ *(che coincide con DFT[f])* fornisce un'approssimazione della Trasformata di Fourier di *f*. In altre parole, la Trasformata di Fourier e la DFT nei punti ω_n coincidono a meno di un errore. L'errore dipende dall'errore di discretizzazione, introdotto dalla formula di quadratura utilizzata, e dall'errore di troncamento della formula di quadratura trapezoidale, introdotto dall'aver assunto $f_p = f$, ovvero la funzione f nulla al di fuori dell'intervallo [0, T]. Per quanto riguarda l'errore di discretizzazione, tale quantità tende a zero al tendere a zero di h.

2. Algoritmo FFT

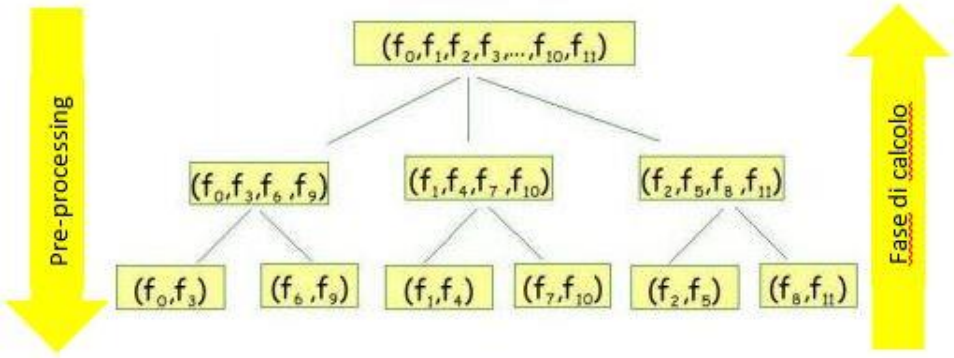
Il calcolo diretto della DFT ha una complessità dell'ordine di $O(n^2)$, poichè per ciascun componente del vettore DFT bsisogna calcolare N somme, in quanto $T(n) = N * [N\ molt. + (N-1) somme] = O(n^2)$. Ciascuna somma prevede, in sostanza, la valutazione di N esponenziali complessi ed N operazioni complesse. Per venire incontro alle esigenze di rapidità per il calcolo della trasformata di Fourier sono state messe a punto versioni più specializzate rispetto alla DFT ovvero le Fast Fourier Transform (FFT). L'approccio di base è il *divide et impera* che permette di ricondurre il calcolo di una DFT di lunghezza N = r * q a quello di r DFT di lunghezza q ottenendo una riduzione della complessità di tempo da $O(N^2)$, a $O(N\log(N))$.
Nel seguente paragrafo tratteremo due algoritmi di FFT:

- L'algoritmo di Cooley e Tukey;
- L'algoritmo di Gentleman e Sande.

2.1. Cooley e Tukey

L'algoritmo di Cooley-Tukey si basa sulla decomposizione ricorsiva del vettore da trasformare (di lunghezza N) fino ad ottenere un insieme di N vettori di lunghezza 1, la cui trasformazione è banale (del tipo $F_1 = f_1$). Il principio guida dell'algoritmo di Cooley-Tukey è, come anticipato, il "*divide et impera*", che prevede di raggruppare le somme delle componenti del vettore in input in base alla posizione dei campioni, in modo da minimizzare il numero complessivo di operazioni da svolgere, riducendo la complessità ad un $O(N \log_2 N)$. In questo modo otteniamo le cosiddette butterfly.
Esistono vari di criteri di divisione del vettore di partenza: si può optare per una divisione per due del vettore ed in questo caso avremo la variante Radix 2 o una divisone per qualunque altro numero fino ad avere addirittura radici miste.
Nel caso Cooley e Tukey radix-2 si ha che una DFT di lunghezza N è divisa in due DFT di lunghezza N/2 ed applicando ricorsivamente il procedimento

partendo da 2 DFT di lunghezza N/2 si ottengono 4 DFT di lunghezza N/4, poi 8 di lunghezza N/8 e così via fino ad avere una schiera di DFT a dimensione minima che rappresentano una coppia di campioni, come mostrato in figura. Dunque, se il numero degli elementi interni al vettore N è una potenza di due, $N = 2^S$, è possibile ottenere N/2 DFT di lunghezza 2, che sono banalmente risolvibili ciascuna con una somma. Si ottiene al termine della prima fase di decomposizione detta di "p*re-processing"* un albero binario. La fase dell'algoritmo che risale l'albero chiudendo le chiamate ricorsive ed unendo le soluzioni dei singoli sotto-problemi è chiamata *"fase di calcolo*".



Per capire meglio come avviene la divisione della DFT e la ricostruzione della soluzione possiamo rifarci allo pseudocodice dell'algoritmo:

```
procedure FFT(A)
    Input: An array of complex values which has a size of
2^m for m ≥ 0.
    Output: An array of complex values which is the DFT
of the input
    N := A.length
    if N = 1 then return A
    else
        W_N := e^(2πi/N)
        W := 1
        A_even := (A_0, A_2, ...., A_N−2)
        A_odd := (A_1, A_3, ...., A_N−1)
        Y_even := FFT(A_even)
        Y_odd := FFT(A_odd)
        for j:=0 to N/2 − 1 do
            Y[j] = Y_even[j] + W * Y_odd[j]
            Y[j + N/2] = Y_even[j] - W * Y_odd[j]
            W := W * W_N
        return Y
```

Dove notiamo che ad un primo passo la DFT di lunghezza N è scissa in due DFT di lunghezza N/2, di cui una con le componenti di indice pari e l'altra con le componenti di indice dispari, le quali vengono poi ricombinate a due a due.

La stessa idea, come anticipato, è applicabile anche per altri tipi di radix, ad esempio in radix-3 si decompone una DFT di lunghezza N in N/3 DFT di lunghezza 3. Nel caso mixed radix, invece, l'idea è di decomporre prima rispetto ad una radice, e poi eventualmente rispetto all'altra radice: se N = r1 * r2, devo risolvere r1 DFT di lunghezza r2, oppure r2 DFT di lunghezza r1. Si evince che l'algoritmo di CT è modulare, ovvero ciascuna DFT di lunghezza r2 la posso risolvere anche con un altro algoritmo di FFT, o comunque nel caso generale l'algoritmo mi porta a fare una sorta di scomposizione in fattori primi.

2.2. Gentlemen e Sande

Una strategia leggermente diversa da quella alla base dello schema di Cooley e Tukey sottende la versione dell'algoritmo FFT radix-2 introdotta da Gentleman e Sande. Tale variante si basa sul decomporre ripetutamente il vettore DFT da calcolare. Si osserva che le componenti con indice pari della DFT si possono esprimere come una DFT di lunghezza *N*/2 costruita a partire da un vettore ottenuto combinando le componenti del vettore di input che distano di *N*/2. Analogo ragionamento si fa per le componenti con indice dispari della DFT da calcolare.

Applicando la proprietà di periodicità dell'esponenziale complesso si ha ($w_N = e^{2*pi*i/N}$):

$$F(k) = \sum_{j=0}^{N-1} f(j) w_N^{-jk} = \sum_{j=0}^{N/2-1} f_j w_N^{-jk} + \sum_{i=0}^{N/2-1} f(j+N/2) w_N^{-(j+N/2)k}$$

Si ha:

$$F(k) = \sum_{j=0}^{N/2-1} [f(j) w_N^{-jk} + f(j+N/2) w_N^{(-j+N/2)k}] = \sum_{j=0}^{N/2-1} [f(j) + (-1)^k f(j+N/2)] w_{N/2}^{-jk/2}$$

L'ultima eguaglianza sussiste in quanto $w_N^{-N/2} = -1$ e $w_N^{-jk} = w_N^{-jk/2}$.
Consideriamo le componenti con indice pari:

$$F(2k) = \sum_{j=0}^{N/2-1} [f(j) + f(j+N/2) w_{N/2}^{-jk}], \quad k = 0, \ldots, N/2-1$$

E quelle con indice dispari:

$$F(2k+1) = \sum_{j=0}^{N/2-1} [(f(j) - f(j+N/2)) w_N^{-j}] w_{N/2}^{-jk}, \quad k = 0, \ldots, N/2-1$$

Introdotti i vettori $\{y_j\}_{j=0,\ \ldots,\ N/2\ -\ 1}$ e $\{z_j\}_{j=0,\ \ldots,\ N/2\ -\ 1}$ di componenti

$$y_j = f_j + f_{j+N/2}, \quad z_j = (f_j - f_{j+N/2})w_N^{-j}, \quad j = 0, \ldots, N/2 - 1$$

I due vettori di lunghezza $N/2$, $\{F_{2k}\}_{k=0,\ \ldots,\ N/2}$ e $\{F_{2k+1}\}_{k=0,\ \ldots,\ N/2}$ si possono riguardare come DFT di lunghezza $N/2$ costruite rispettivamente su $\{y_j\}_{j=0,\ \ldots,\ N/2\ -\ 1}$ e $\{z_j\}_{j=0,\ \ldots,\ N/2\ -1}$.

$$F = DFT[f] = (\{F_{2k}\}, \{F_{2k+1}\})_{k=0,\ldots,N/2} = (DFT[y], DFT[z])$$

Al primo passo, l'algoritmo di Gentleman e Sande calcola i due vettori $[y]$ e $[z]$. Consideriamo ciascuna DFT e separiamo le componenti con indice pari da quelle con indice dispari. I due vettori così costruiti, di lunghezza $N/4$, sono esprimibili in termini di DFT di lunghezza $N/4$:

$$DFT[y] = \sum_{j=0}^{N/2-1} [f(j) + f(j + N/2)w_{N/2}^{-jk}] = \sum_{j=0}^{N/2-1} y_j w_{N/2}^{-jk} =$$

$$= \sum_{j=0}^{N/4-1} (y_j + y_{j+N/4})w_{N/4}^{-jk} + \sum_{j=0}^{N/4-1} [(y_j - y_{j+N/4})\omega_{N/2}^{-j}]w_{N/4}^{-jk}$$

Se consideriamo le componenti con indice pari del vettore $DFT[y]$:

$$DFT[y](2k) = \sum_{j=0}^{N/4-1} (y_j + y_{j+N/4})\omega_{N/4}^{-jk} \quad k = 0, \ldots, N/4 - 1$$

E:

$$DFT[y](2k+1) = \sum_{j=0}^{N/4-1} (y_j - y_{j+N/4})\omega_{N/2}^{-j}\omega_{N/4}^{-jk} \quad k = 0, \ldots, N/4 - 1$$

Introdotti i vettori di lunghezza $N/4$, $\{y'\}_{j=0,\ \dots\ ,\ N/4\ -\ 1}$ e $\{y''\}_{j=0,\ \dots\ ,\ N/4\ -\ 1}$, di componenti:

$$y'_j = (y_j + y_{j+N/4}), \quad y''_j = (y_j - y_{j+N/4}) w_{N/2}^{-j}$$

Segue che:

$$\{DFT[y]\} = \{DFT[y'], DFT[y'']\}$$

Analogamente, per il vettore $DFT[z]$ contenente le componenti con indice dispari della DFT di f:

$$DFT[z](k) = \sum_{j=0}^{N/2-1} [(f_j - f_{j+N/4}) w_{N/2}^{-j}] w_{N/4}^{-jk} = \sum_{0}^{N/2-1} z_j w_{N/2}^{-jk} =$$

$$= \sum_{j=0}^{N/4-1} [(z_j + z_{j+N/4}) w_{N/2}^{j}] w_{N/4}^{jk} + \sum_{j=0}^{N/4-1} [(z_j - z_{j+N/4}) w_{N/2}^{-j}] w_{N/4}^{-jk}, \quad k = 0, N/4 - 1$$

Introdotti i vettori $\{z'\}_{j=0,\ \dots,\ N/4\ -\ 1}$ e $\{z''\}_{j=0,\ \dots,\ N/4\ -\ 1}$ di componenti:

$$z'_j = z_j + z_{j+N/4}, \quad z''_j = (z_j - z_{j+N/4}) w_{N/2}^{-j}$$

Segue che:

$$DFT[z] = \{DFT[z'], DFT[z'']\}$$

Al secondo passo, l'algoritmo di Gentleman e Sande calcola i vettori *y', y'', z'* e *z''* e il calcolo della DFT di lunghezza *N* viene ricondotto a quello di 4 DFT di lunghezza *N*/4:

$$F = DFT[f] = (DFT[y], DFT[z]) = \\ = (DFT[y'], DFT[y'']), (DFT[z'], DFT[z''])$$

Proseguendo in questo modo, dopo $m = \log(N)$ passi, l'algoritmo di Gentleman e Sande costruisce *N*/2 vettori di lunghezza 2. Ciascuna coppia è una DFT e fornisce una coppia di componenti della DFT del vettore *f*. A differenza della variante di Cooley e Tukey, si nota che in questa strategia le DFT calcolate in un certo passo sono ottenute utilizzando il vettore costruito al passo precedente, per cui l'algoritmo di Gentleman e Sande prevede inizialmente una fase di calcolo, per un certo numero di passi dipendente dalla dimensione del vettore *f*, e poi, una volta arrivati a calcolare le *N*/2 DFT di lunghezza 2, è necessaria una fase di riordinamento del vettore DFT, dal momento che tale vettore risulta nell'ordine *scrambled*, ovvero secondo l'ordinamento indotto dal *bit reversal*. Questa variante è anche nota in letteratura come *decimazione nelle frequenze*, perché la decomposizione viene effettuata sulle componenti del vettore DFT, e nell'analisi armonica tale vettore fornisce informazioni sulle frequenze contenute nel vettore dato. Nelle implementazioni pratiche, le butterflies sono costruite guardando la rappresentazione binaria degli indici delle componenti del vettore di cui calcolare la DFT. Infatti, si nota che al primo passo gli indici delle componenti da accoppiare differiscono solo nel bit più significativo, al secondo passo solo nel secondo bit più significativo, e così via (considerando N potenza di 2). In generale, se consideriamo come sottostringa di bit "attiva" al passo j-esimo come quella costituita dagli N – j bit meno significativi (quindi per j=0 tutta la stringa, per j=1 ho tolto il bit più significativo considerato all'iterazione precedente), allora gli indici delle componenti da accoppiare differiscono sempre nel bit più significativo di quella che abbiamo definito come sottostringa attiva.

2.3. Analisi complessità famiglie FFT

In generale, quindi, gli algoritmi FFT effettuano il calcolo di una DFT di lunghezza N combinando opportunamente DFT di lunghezza inferiore. In base alla fattorizzazione del parametro N si distinguono essenzialmente tre tipologie di algoritmi FFT.

1. Se $N = r_1 r_2$, e r_1 e r_2 sono primi tra loro, gli algoritmi FFT calcolano r_1 DFT di lunghezza r_2 (algoritmi FFT con fattori primi (Prime Factor Algorithms (PFA))). La complessità di questi algoritmi è:

$$T(N) = O(N(r_1 + r_2))$$

2. Se $N = r_1^p \, r_2^q$ gli algoritmi FFT calcolano p DFT di lunghezza r e q DFT di lunghezza r (algoritmi a radici miste (FFT mixed radix). Tipicamente, la radice in questo caso è un numero primo sufficientemente piccolo, ad esempio r = 3, 5. La complessità di questa classe dalgoritmi è:

$$T(N) = O(p \log_{r1} N + q \log_{r2} N)$$

3. Se $N = r^p$ gli algoritmi FFT calcolano p DFT di lunghezza r (algoritmi *radix-r*). In particolare, tra gli algoritmi radix-r, i più efficienti sono quelli del tipo radix-3: ciò deriva dal fatto che in questi casi le funzioni trigonometriche assumono valori uguali a +1, -1 e 0. Tuttavia, nella pratica, per come sono fatti i calcolatori, sui quali si possono effettuare moltiplicazioni e divisioni per 2 mediante shift dei registri, gli algoritmi più utilizzati sono quelli del tipo radix-2, a cui appartengono anche quelli in cui r è una potenza di 2 come gli algoritmi radix-4 e radix-8 La complessità di tempo $T(N)$ di questi algoritmi è:

$$T_{DFT}(N) = log_2 N \cdot \left[\frac{N}{2} \cdot \underbrace{T_{DFT}(2)}_{2} \right] = O(N log_2 N)$$

Esiste una stretta relazione tra l'algoritmo radix-2 e gli algoritmi radix-r, infatti la complessità di tali algoritmi è data da:

$$T(N) = O(N r log_r N) = O(N r \frac{log_2 N}{log_2 r}) = O(\frac{r}{log_2 r} \cdot N log_2 N)$$

Il fattore di proporzionalità $\frac{r}{\log_2 r}$ assume il suo valore minimo per $r = e$; infatti:

$$\frac{d}{dr}\frac{r}{log_2 r} = \frac{log_2 r - log_2 e}{(log_2 r)^2} \geq 0 \iff r \geq e$$

Ed ha nel punto $r = 3$ un valore minore di quello che assume per $r = 2$:

$$\frac{3}{log_2 3} \simeq 1.893, \quad \frac{2}{log_2 2} = 2$$

Quindi l'algoritmo FFT radix-3 è quello che ha la complessità computazionale minima; nonostante ciò, ripetiamo che l'algoritmo radix-2 è di gran lunga il più utilizzato, sia perché la maggior parte dei calcolatori ha un sistema aritmetico floating point in base 2, sia perché con questa scelta alcune potenze w_N^k sono semplificate, (abbiamo visto che il numero complessivo di esponenziali da calcolare è uguale alla metà della lunghezza della sequenza di cui si vuole la FFT).

3. Applicazioni pratiche FFT

Nel seguente capitolo troviamo la FFT applicata in quattro esempi:

- Analisi e filtraggio di un segnale monodimensionale, con riferimento ad un segnale audio;

- Analisi di un segnale bidimensionale, ovvero visualizzazione spettro di ampiezza e fase di un'immagine e confronto dell'immagine originale con le immagini ottenute con spettro di ampiezza e spettro di fase separatamente;

- Analisi e filtraggio di un segnale bidimensionale, attuando un progetto di un filtro in frequenza con l'obbiettivo di ripulire dal rumore sinusoidale applicato all'immagine, con visualizzazione grafica del risultato e confronto dell'errore quadratico medio ottenuto dalla coppia immagine originale ed immagine rumorosa con quello ottenuto dalla coppia immagine originale ed immagine ripulita.

- Confronto complessità attraverso il calcolo dei tempi di esecuzione della funzione di libreria FFT di matlab e DFT creata ad hoc come prodotto matriciale.

Esercizio 1 - FFT su segnale 1D, analisi e filtraggio

Lettura file audio

```
path = 'PinkPanther30.wav';
[x, FS] = audioread(path);
```

Visualizzazione nel tempo e nella frequenza del segnale audio

```
info = audioinfo(path);
t = 0:seconds(1/FS):seconds(info.Duration);
t = t(1:end-1);
figure(1)
subplot(2,2,1); plot(t,x);
xlabel('time'); ylabel('audio Signal'); title('Diagramma temporale prima');
```

calcolo la FT di x(n) tramite FFT

```
X = fft(x);
```

traccio l'istogramma

```
[M, N] = size(t);
freq = (0:N/2) * FS/(N);
```

grafico istogramma

```
subplot(2,2,2); stem(freq, abs(X(1:N/2 +1)),'b');
xlabel('frequency'); ylabel('Amplitude'); title('Istogramma delle frequenze
prima');
```

Filtraggio in frequenza

```
H = ones(1, N);
H(freq > 600) = 0;
```

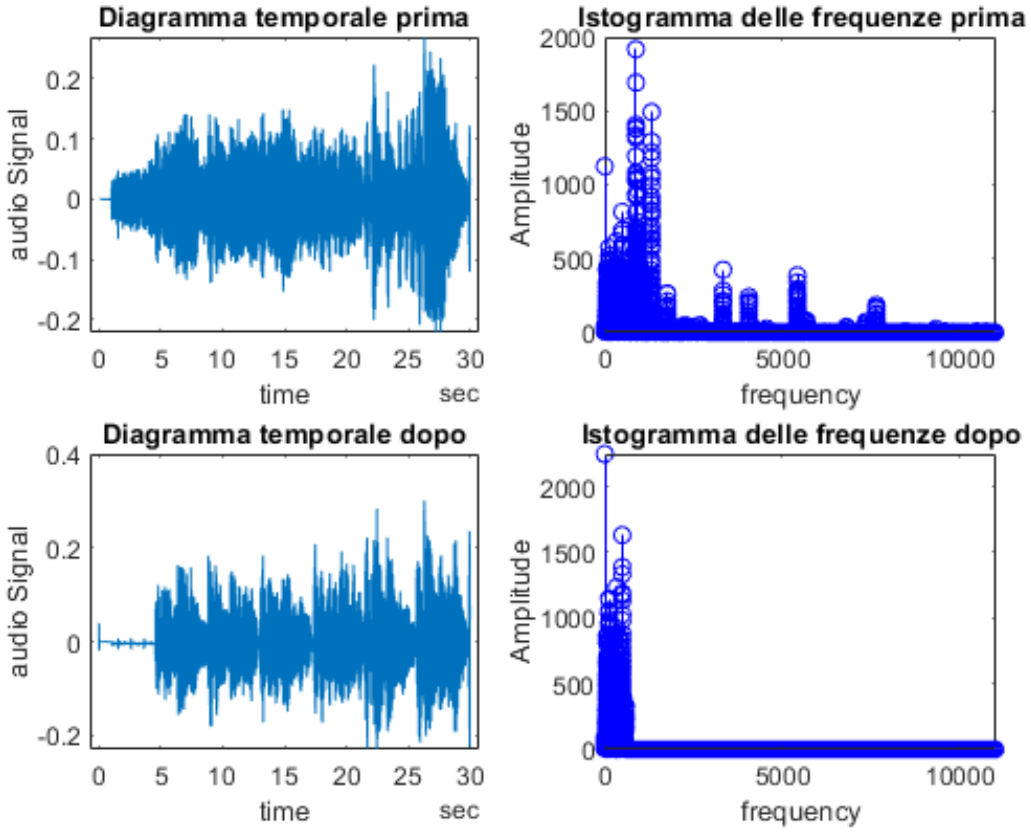
isolo i suoni bassi

copia la prima metà del vettore H nella seconda, per simmetria

```
 H(N/2 +1 : N) = fliplr(H(1:N/2));
 X = X .* (H');
 X = X .* 2; %sempre per la simmetria
 subplot(2,2,4); stem(freq, abs(X(1:N/2 +1)),'b');
 xlabel('frequency'); ylabel('Amplitude'); title('Istogramma delle frequenze
dopo');
```

Ritorno nel dominio del tempo dopo il filtraggio

```
 x = real(ifft(X));
 subplot(2,2,3); plot(t, x);
 xlabel('time'); ylabel('audio Signal'); title('Diagramma temporale dopo');
```



Riproduzione e salvataggio dell'audio ottenuto

```
 sound(x, FS); % riproduce l'audio alla frequenza corretta
 audiowrite('PinkPanther30_bassi.wav', x, FS)
```

Esercizio 2 - FFT su segnale 2D, solo analisi

Lettura immagine

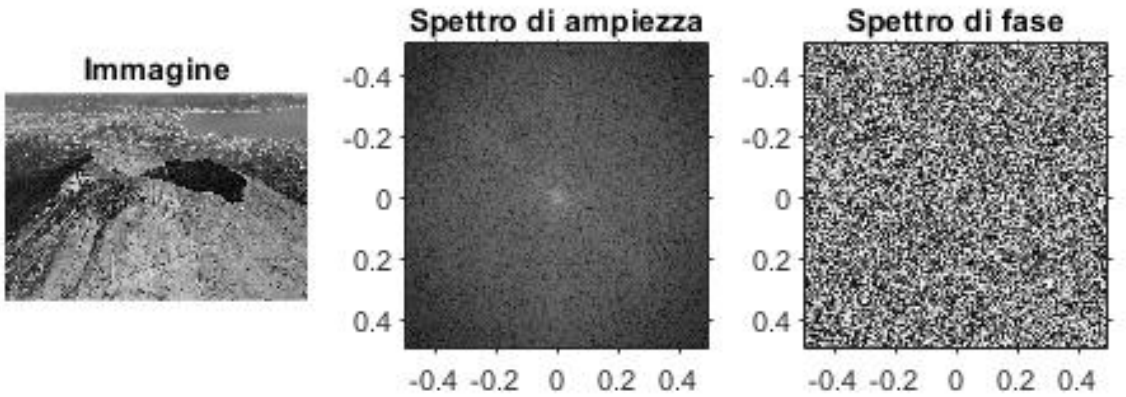
```
x = imread('vesuvio_cratere.jpg');
x = double(rgb2gray(x)) ./ 255;
```

Trasformata immagine

```
[M, N] = size(x);
X = fft2(x, M, N);
Xf = fftshift(X);
m = -0.5:(1/M):(0.5-1/M);
n = -0.5:(1/N):(0.5-1/N);
```

Visualizzazione di spettri di ampiezza e di fase rispetto all'immagine

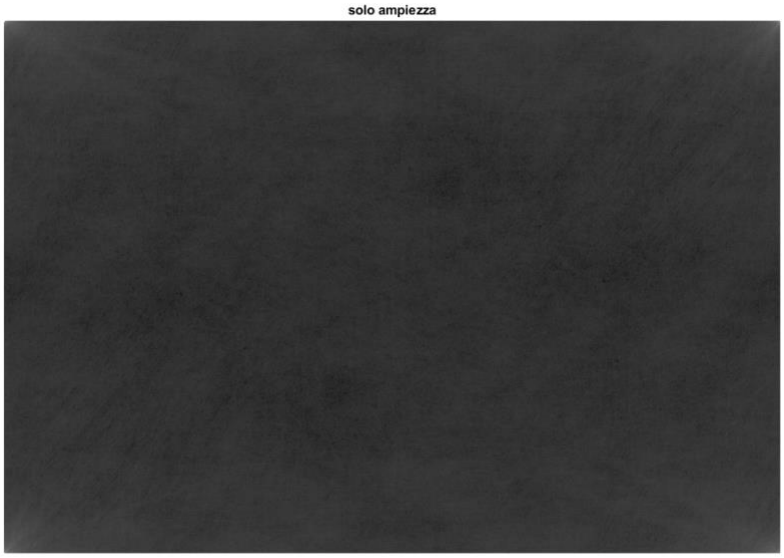
```
figure();
subplot(1,3,1); imshow(x); title('Immagine');
subplot(1,3,2); imshow(log(abs(Xf)+1), [], 'XData', n, 'YData', m);
axis on; title('Spettro di ampiezza');
subplot(1,3,3); imshow(angle(Xf), [], 'XData', n, 'YData', m);
axis on; title('Spettro di fase');
```



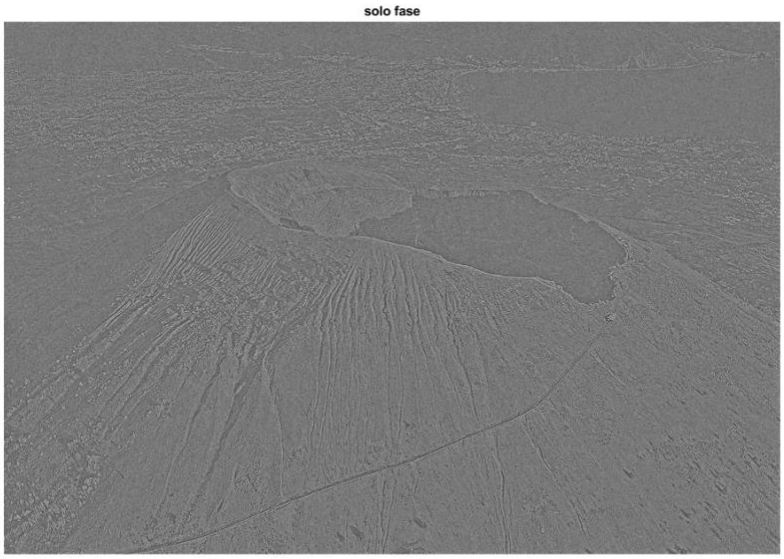
Ricostruzione immagine solo con ampiezza e solo con fase e confronto con l'originale

```
X1 = abs(X);
x1 = ifft2(X1);
X2 = X ./ abs(X); % oppure exp(1j .* angle(X));
x2 = ifft2(X2);
figure(); imshow(x); title('originale');
```

```
figure(); imshow((x1-min(x1(:))).^0.3, []); title('solo ampiezza');
```




```
figure(); imshow(x2, []); title('solo fase');
```



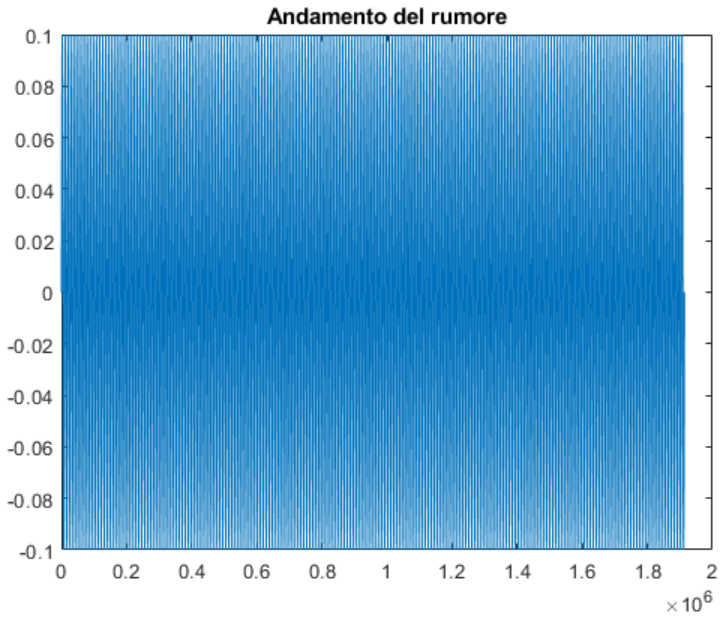
Esercizio 3 - FFT su segnale 2D: aggiunta rumore, analisi e filtraggio

Lettura immagine

```
x = imread('Diego-Armando-Maradona.jpg');
x = double(rgb2gray(x)) ./ 255;
```

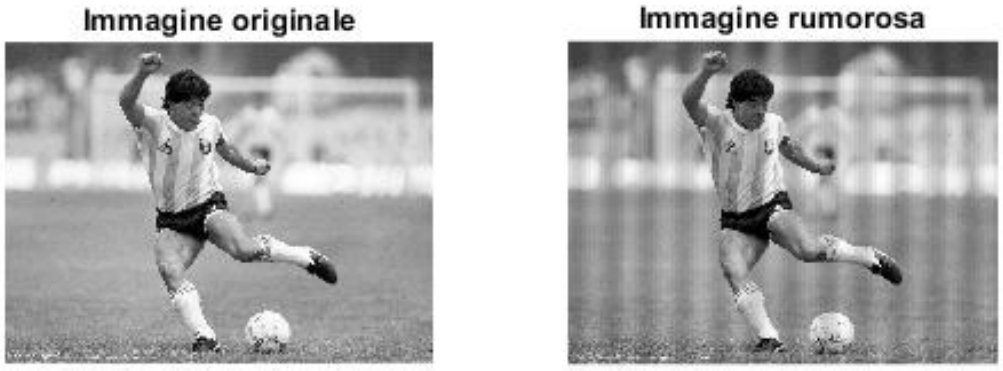
Aggiunta del rumore sinusoidale e visualizzazione

```
d = size(x(:));
freq = 200;
n_x = linspace(0, d(1), d(1)); % potrei metterli tra 0 ed 1 ma così riduco gli errori di calcolo
alfa = 0.1;
n = alfa .* sin(2 * pi / d(1) * freq * n_x);
figure(); plot(n_x, n); title('Andamento del rumore');
```




```
n = reshape(n, size(x));
y = x + x .* n;
y = y + abs(min(y(:)));
y = y ./ max(y(:));
figure();
subplot(1,2,1); imshow(x); title('Immagine originale');
```

```
 subplot(1,2,2); imshow(y); title('Immagine rumorosa');
```

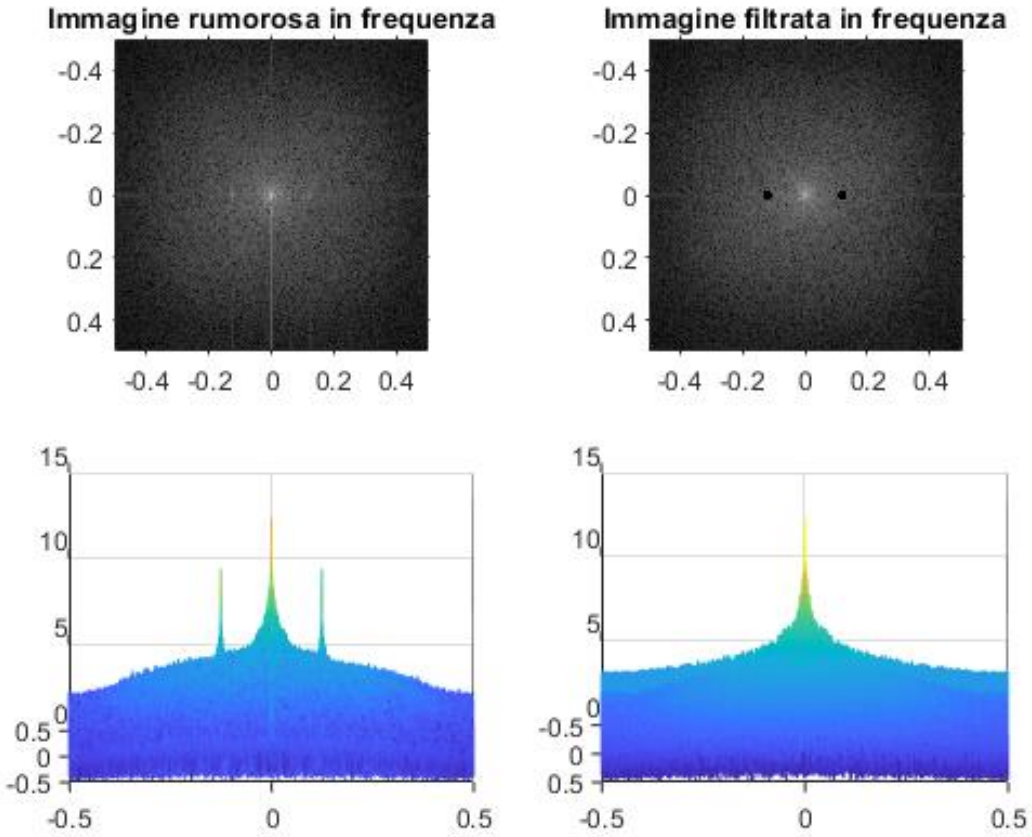


Analisi in frequenza

```
 [M, N] = size(y);
 Y = fftshift(fft2(y,M,N));
 n = (-0.5):1/N:(0.5-1/N);
 m = (-0.5):1/M:(0.5-1/M);
 [l, k] = meshgrid(n, m);
 figure();
 subplot(2,2,1); imshow(log(abs(Y)+1), [], 'XData', n, 'YData', m);
axis('on');
 title('Immagine rumorosa in frequenza');
 subplot(2,2,3); mesh(l, k, log(abs(Y)+1));
```

Filtraggio in frequenza e tuning del filtro

```
 dist = @(l, k, l0, k0) sqrt( (l-l0).^2 + (k-k0).^2 );
 d1 = dist(l, k, 0.12, 0);
 d2 = dist(l, k, -0.12, 0);
 th = 0.015;
 H = (d1 > th) & (d2 > th);
 subplot(2,2,2); imshow(log(abs(Y).*H + 1), [], 'XData', n, 'YData', m);
axis('on');
 title('Immagine filtrata in frequenza');
 subplot(2,2,4); mesh(l, k, log(abs(Y).*H+1));
```




```
 F = Y .* H;
```

Ritorno nel dominio del tempo e visualizzo il risultato

```
f = real(ifft2(ifftshift(F)));
figure();
imshow(f, []); title('Immagine filtrata');
```

Valutazione dell'errore quadratico medio

```
f = f + abs(min(f(:)));
f = f ./ max(f(:));
mse_noisy = immse(x, y)
```

```
mse_noisy = 0.0055
```

```
mse_filtered = immse(x, f)
```

```
mse_filtered = 0.0012
```

Esercizio 4 - Confronto complessità FFT e DFT

Confrontiamo i tempi di esecuzione della FFT con quella della DFT calcolata come prodotto matriciale. Il test sarà fatto su istanze di dimensione variabile, di seguito il codice.

Calcolo del vettore delle dimensioni, ogni elemento è una potenza di due.

```
format long
N=zeros(1,11);
for i=1:11
    N(i) = 2^(i+2);
end
```

Inizializzazione dei vettori dei tempi: t1 misura i tempi della FFT, t2 misura i tempi della DFT.

```
trasf_prod=@(x,W) W*x;
t1=zeros(1,length(N));
t2=zeros(1,length(N));
```

Di seguito il ciclo for usato per il calcolo degli N tempi di esecuzione, nel ciclo for ci serviamo di un segnale x pari ad una finestra rettangolare di N/4 campioni, dunque di dimensione variabile ad ogni ciclo.

```
for i=1:length(N)
    n=[0:N(i)-1];
    k=[0:N(i)-1];
    x=zeros(N(i));
    L=N(i)/4;  %  25% di N(i)
    x(1:L)=1;
    W=zeros(N(i),N(i));
    W=exp(-1i*2*pi*k'*n/N(i));
X=W*x;
    X1=fft(x);
% Impulso rettangolare di durata L
    t1(i)=timeit(@() fft(x));
    t2(i)=timeit(@() trasf_prod(x,W));
end
```

Infine, i grafici a confronto.

```
figure;plot(N,t1);
hold on;
plot(N,t2);legend('FFT','DFT');
```