CAPITOLO 2: Algoritmo Fast Fourier Transform (FFT)

L'algoritmo FFT (Fast Fourier Transform) è decisamente uno dei più utilizzati all'interno dell'elaborazione dei segnali digitali. Il suo scopo è quello di calcolare la trasformata discreta di Fourier (DFT) in maniera veloce.

2.1 La trasformata discreta di Fourier (DFT)

Precedentemente si è visto come la trasformata di Fourier di un segnale tempo discreto ad energia finita possa essere scritta come in (1.3.1).
Supponiamo di campionare $X(\omega)$ ad intervalli regolari. Dal momento che $X(\omega)$ è periodica di periodo 2π, sono necessari solo i campioni prelevati in un intervallo. Se il numero di campioni considerato è N, la spaziatura è pari a $\Delta\omega = 2\pi / N$. Sostituendo nella (1.3.1) il valore del campione k-esimo, si ottiene:

$$X\left(\frac{2\pi k}{N}\right) = \sum_{-\infty}^{\infty} x(n) e^{-j2\pi kn/N} \qquad k = 0, 1, ..., N-1 \tag{2.1.1}$$

L'espressione (2.1.1) è composta da un numero infinito di somme, ciascuna contenente N termini. Cioè:

$$X\left(\frac{2\pi k}{N}\right) = ... + \sum_{n=-N}^{-1} x(n) e^{-j2\pi kn/N} + \sum_{n=0}^{N-1} x(n) e^{-j2\pi kn/N} + \sum_{n=N}^{2N-1} x(n) e^{-j2\pi kn/N} + ... = \tag{2.1.2}$$

$$= \sum_{l=-\infty}^{\infty} \sum_{n=lN}^{lN+N-1} x(n) e^{-j2\pi kn/N}$$

Manipolando questa equazione, modificando n con n-lN nella serie più interna, si ottiene:

$$X\left(\frac{2\pi k}{N}\right) = \sum_{n=0}^{N-1} \left[\sum_{l=-\infty}^{\infty} x(n - lN) \right] e^{-j2\pi kn/N} \tag{2.1.3}$$

Il segnale:

$$x_p(n) = \sum_{l=-\infty}^{\infty} x(n - lN) \tag{2.1.4}$$

ottenuto come ripetizione del segnale $x(n)$ è chiaramente periodico di periodo N.
Il segnale $x_p(n)$ può essere perciò decomposto in serie di Fourier; dopo alcuni passaggi si ottiene:

$$x_p(n) = \frac{1}{N} \sum_{k=0}^{N-1} X\left(\frac{2\pi k}{N}\right) e^{j2\pi kn/N} \qquad n = 0, 1, ..., N-1 \tag{2.1.5}$$

La relazione (2.1.5) rappresenta la ricostruzione del segnale periodico $x_p(n)$ a partire dai campioni dello spettro $X(\omega)$. Per poter comunque ricostruire $x(n)$ a partire da $x_p(n)$ è opportuno che nel dominio del tempo non vi sia *aliasing*, cioè che il numero di campioni di cui è costituito $x(n)$ non superi il periodo temporale N di $x_p(n)$. Quindi, lo spettro di un segnale aperiodico tempo discreto con durata finita pari ad L può essere calcolato a partire dai suoi campioni alle frequenze $\omega_k = 2\pi k/N$, se $N \geq L$.
Il procedimento consiste quindi nel calcolare $x_p(n)$ mediante la (2.1.5), dopodichè:

$$x(n) = \begin{cases} x_p(n) & 0 \leq n \leq N\text{-}1 \\ 0 & \text{altrimenti} \end{cases} \tag{2.1.6}$$

Se quindi la durata di $x(n)$ è pari ad L campioni, dove $L \leq N$, per un singolo periodo si può scrivere:

$$x_p(n) = \begin{cases} x(n) & 0 \leq n \leq L\text{-}1 \\ 0 & L \leq n \leq N\text{-}1 \end{cases} \tag{2.1.7}$$

Questa relazione ci è utile, perchè il valore di $x(n)$ è lo stesso di $x_p(n)$, che a sua volta può essere calcolato mediante la (2.1.5) . Inoltre, l'operazione di *zero padding* (riempimento con zeri) non fornisce alcuna informazione aggiuntiva sullo spettro $X(\omega)$.
Dunque, se $x(n)$ è una sequenza di durata finita L, si può scrivere:

$$X(\omega) = \sum_{n=0}^{L-1} x(n) e^{-j\omega n} \qquad 0 \leq \omega \leq 2\pi \tag{2.1.8}$$

Campionando $X(\omega)$ ad intervalli regolari $\omega_k = 2\pi k/N$, con $k = 0, 1, 2, ..., N-1$, si ottiene:

$$X(k) \equiv X\left(\frac{2\pi k}{N}\right) = \sum_{n=0}^{L-1} x(n) e^{-j2\pi kn/N}$$

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-j2\pi kn/N} \qquad k = 0, 1, 2, ..., N-1 \tag{2.1.9}$$

La relazione (2.1.9) è chiamata *trasformata discreta di Fourier* (DFT) di $x(n)$.
La relazione che invece permette, a partire dai campioni $X(k)$, di trovare il segnale $x(n)$, cioè

$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X_k e^{j2\pi kn/N} \qquad n = 0, 1, 2, ..., N-1 \tag{2.1.10}$$

viene chiamata *trasformata discreta inversa di Fourier* (IDFT)

2.2 Algoritmo FFT

Sebbene la trasformata discreta di Fourier risulti una potentissima procedura di risoluzioni di molti problemi matematici, essa è caratterizzata da una complessità computazionale che è eccessiva quando si utilizza la DFT per risolvere problemi di grandi dimensioni. Tutto ciò ha portato alla ricerca di un algoritmo per il calcolo della DFT che avesse una complessità inferiore.
Nel 1965 l'ingegnere dell'IBM Cooley e il matematico Tukey giunsero alla determinazione dell'algoritmo noto come "*Decimation in time*", detto anche *Fast Fourier Transform* (FFT), che riduceva notevolmente la complessità computazionale della DFT e ne aumentava inoltre la stabilità.
L'FFT è semplicemente un algoritmo che consente di calcolare in modo molto rapido la trasformata discreta di Fourier DFT.
La strategia su cui si basano gli algoritmi FFT, detti "*divide and conquer*", è quella di ricondurre il calcolo di una DFT di lunghezza N al calcolo di una DFT di lunghezza inferiore.
Cerchiamo ora di capire quale sarebbe la complessità di un programma che calcolasse la DFT in maniera diretta.
Ricordando che la DFT è espressa in (2.1.9), il valore $X(k)$ può essere ricavato attraverso N moltiplicazioni complesse ed $N - 1$ addizioni complesse; essendo però una moltiplicazione complessa riconducibile a quattro moltiplicazioni tra numeri reali, ed essendo inoltre un'addizione complessa rappresentabile con due addizioni reali, il numero di operazioni complessivo risulta

essere di $8N - 2$ operazioni per ogni valore dell'indice k. L'indice k, inoltre, varia tra 0 ed $N - 1$, per cui si arriva a determinare un numero di operazioni pari a

$$N_{op}(N) = (8N - 2)N = 8N^2 - 2N \tag{2.2.1}$$

L'algoritmo FFT (o meglio la classe di algoritmi, visto che ne esistono più di uno) permette di abbattere notevolmente il carico computazionale appena descritto. In pratica per far ciò l'algoritmo utilizza due proprietà note del fattore $W_N = e^{-j2\pi/N}$, presente nel calcolo della DFT:

- Proprietà di simmetria: $W_N^{k+N/2} = -W_N^k$ (2.2.2)
- Proprietà di periodicità: $W_N^{k+N} = W_N^k$ (2.2.3)

Supponiamo di considerare il più semplice algoritmo di FFT, il *radix-2*. In questo caso l'ordine N della trasformata è (come si intuisce facilmente dal nome stesso) una potenza di due, cioè $N = 2^R$. La trasformata DFT può allora essere riscritta separando i valori di $x(n)$ con indice n pari da quelli con indice dispari, ottenendo:

$$X_k = \overbrace{\sum_{n=0}^{N/2-1} x(n) e^{-j\frac{2\pi kn}{N/2}}}^{P_k} + e^{-j\frac{2\pi k}{N}} \overbrace{\sum_{n=0}^{N/2-1} x(n+1) e^{-j\frac{2\pi kn}{N/2}}}^{D_k}, \quad k = 0, \ldots, N-1 \tag{2.2.4}$$

La prima sommatoria rappresenta la DFT degli $N/2$ campioni di $x(n)$ con n pari, mentre la seconda rappresenta la DFT degli $N/2$ campioni con n dispari. Risulta subito evidente quindi come la DFT di ordine N possa essere calcolata a partire dal calcolo di due DFT di ordine $N/2$.
Calcoliamo ora il numero di operazioni necessarie per risolvere la trasformata. Si ottiene:

$$N_{FFT}(N) = N_{FFT}\left(\frac{N}{2}\right) + N_{FFT}\left(\frac{N}{2}\right) + 6N + 2N \tag{2.2.5}$$

dove i primi due termini corrispondono alla complessità di calcolo delle due trasformate di ordine $N/2$; il terzo termine corrisponde alla complessità nel calcolo di un esponenziale complesso (6 operazioni reali), il quarto dalla somma con P_k (è la somma tra due numeri complessi, quindi occorrono 2 operazioni reali).
Il termine (2.2.5) può essere riscritto come:

$$N_{FFT}(N) = 2\left[N_{FFT}\left(\frac{N}{2}\right) + 4N \right] \tag{2.2.6}$$

Ma il metodo di scomposizione utilizzato sopra è ancora valido (siano nel caso di *N* potenza di due); l'idea è quindi quella di iterare il procedimento, fino a calcolare la quantità $N_{FFT}(1)$.
Al secondo passo si hanno sottosequenze tutte di lunghezza *N*/4: il calcolo di una trasformata di ordine *N*/2 comporta quindi una complessità:

$$N_{FFT}\left(\frac{N}{2}\right) = 2\left[N_{FFT}\left(\frac{N}{4}\right) + \frac{4N}{2} \right] \tag{2.2.7}$$

che sostituita nella espressione di partenza porta ad una complessità :

$$N_{FFT}(N) = 4N_{FFT}\left(\frac{N}{4}\right) + 2 \cdot 8N \tag{2.2.8}$$

Continuando ad iterare il procedimento come descritto, si arriva ad avere:

$$N_{FFT}(N) = N \cdot N_{FFT}\left(\frac{N}{N}\right) + \log_2 N \cdot 8N \tag{2.2.9}$$

dove $N_{FFT}(1)$ comporta una sola moltiplicazione ($X_0 = x(0)e^{j0}$) per cui si ricava:

$$N_{FFT}(N) = 6N + 8N \log_2 N \cong 8N \log_2 N \tag{2.2.10}$$

La complessità dell'algoritmo FFT è quindi notevolmente inferiore a quella per il calcolo diretto della DFT. Per fare un esempio, consideriamo un fattore *N* pari a 1024. Il rapporto tra il numero di operazioni necessario nei due casi è:

$$\frac{N_{TDF}(N)}{N_{FFT}(N)} = \frac{8N^2}{8N \log_2 N} = \frac{N}{\log 2N} \cong 100 \tag{2.2.11}$$

Questo significa che la complessità nel calcolo diretto della DFT è 100 volte maggiore che utilizzando la FFT!
In effetti, benchè la DFT fosse già ben conosciuta negli anni ’60, è solo con l’avvento dell’algoritmo FFT che la trasformata discreta di Fourier ha potuto essere implementata su calcolatore, all’epoca troppo poco potenti.

2.3 Esempi

In questo paragrafo si proporranno alcuni esempi sull’utilizzo della FFT , fornendo anche spiegazioni sull’utilizzo di tale algoritmo in Matlab.

All’interno di Matlab, l’algoritmo è gia implementato nella funzione $y = fft(x,n)$ (la funzione accetta anche altri parametri, ma a noi non interessano), dove:

- x è il segnale d’ingresso
- n è il numero di punti utilizzati del calcolo della DFT; se n è maggiore del numero di campioni del segnale, si effettua lo zero padding su x, mentre se n è minore x viene tagliato
- y rappresenta la DFT del segnale d’ingresso x

La funzione inversa della *fft*, ossia quella che calcola il segnale a partire dalla DFT, è la $y = ifft(x,n)$, dove:

- x rappresenta la DFT del segnale calcolata attraverso la FFT
- n è il numero di punti utilizzati del calcolo della IDFT
- y rappresenta la IDFT del segnale di x

In questo caso, se il segnale iniziale è reale, l’operazione $ifft(fft(x))$ può dare alcune componenti complesse, quindi è conveniente considerarne la sola parte reale.

Il rumore viene generato sotto Matlab con la funzione *awgn*; un esempio di chiamata per questa funzione è:

$$x = awgn(x, SNR, \text{'measured'})$$

dove:

- x è il segnale da sommare con rumore gaussiano bianco a media nulla

- SNR è il rapporto segnale rumore da impostare espresso in dB
- 'measured' è la stringa che passata come parametro indica alla funzione di calcolare la potenza di *x* prima di sommare il rumore

Consideriamo il segnale

$$x(t) = 2\sin(0.1t) + 3\sin(0.05t) + 5\sin(0.2t) + \sin(0.4t)$$

rappresentato in figura. Il segnale è scelto appositamente complesso per dimostrare le potenzialità dell'algoritmo FFT.

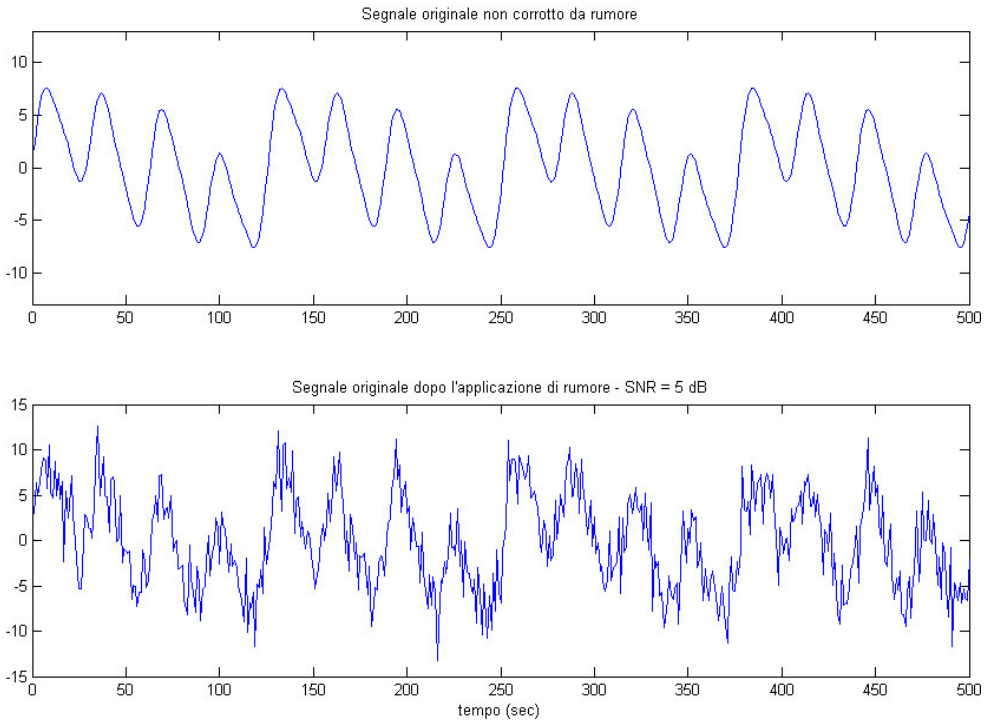

Figura 2.1. Segnale *x*(*t*) prima e dopo l'applicazione del rumore

Come appare evidente, il segnale corrotto da rumore (SNR = 5dB) sembra non aver più niente a che vedere con il segnale originario. Vogliamo vedere se applicando l'algoritmo FFT è possibile estrapolare qualche informazione utile. I test vengono eseguiti al variare del numero di campioni della FFT; in particolare si utilizzeranno rispettivamente 128, 256, 512 campioni.

I risultati ottenuti sono evidenziati in tabella:

Frequenze reali (rad/sec)	**Frequenze stimate (rad/sec) con punti FFT = 128**	**Frequenze stimate (rad/sec) con punti FFT = 256**	**Frequenze stimate (rad/sec) con punti FFT = 512**
0.05	0.0503	0.0491	0.0491
0.1	0.1005	0.0982	0.0982
0.2	0.2011	0.1963	0.1963
0.4	0.4021	0.3927	0.4050

Sebbene ad occhio nudo non si riesca più nemmeno ad associare il segnale rumoroso a quello originario, mediante la FFT è possibile ricostruire il segnale ripulito dal rumore.
Utilizzando un numero di punti per la FFT pari alla lunghezza del segnale (in modo da avere la IDFT pari alla lunghezza del segnale originario), in questo caso 500, la ricostruzione è quasi perfetta:

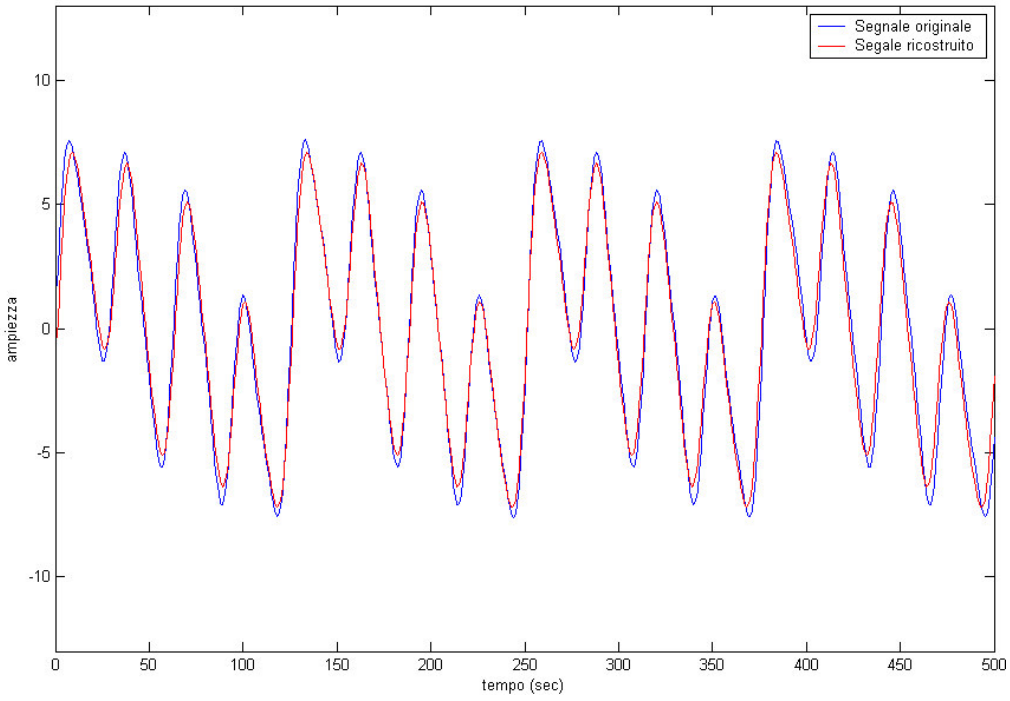

Figura 2.2. Sovrapposizione del segnale originale e quello ricostruito

Appare evidente come la FFT possa essere utilizzata con successo in molti campi dove si richiede la cancellazione del rumore su un segnale anche fortemente corrotto.

In pratica, per arrivare al risultato della figura sopra, sono possibili almeno due strade:

1) La prima possibilità consiste nel percorrere i seguenti passi:
   a) calcolare la FFT ad *N* punti del segnale di partenza, ottenendo l'uscita *y*
   b) calcolare lo spettro, valutando per ogni campione la potenza come *y*.* conj(*y*) / *N*
   c) conoscendo il numero di componenti frequenziali *p* di cui è composto il segnale di partenza (*p* componenti), identificare le frequenze corrispondenti ai *p* picchi in potenza più elevati
   d) nel vettore *y*, annullare tutte le componenti tranne quelle relative alle frequenze ottenute
   e) calcolare la IDFT di *y* come *ifft*(*y*), ottenendo il segnale ricostruito

2) La seconda strada percorribile consiste in:
   a) come nel punto 1-a, calcolare la FFT ad *N* punti del segnale di partenza, ottenendo l'uscita *y*
   b) come nel punto 1-b, valutare la potenza spettrale
   c) impostare una soglia *s*, il cui valore dipende dall'influenza del rumore sul segnale
   d) per il vettore *y*, azzerare tutte le componenti che presentano potenza inferiore a $s*P_{max}$, dove P_{max} è la potenza massima
   e) calcolare il segnale ricostruito mediante la funzione *ifft*(*y*)

Entrambe le possibilità sono state implementate all'interno di questa tesi.