UNIVERSITÀ DEGLI STUDI DI BRESCIA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Corso di laurea triennale in ingegneria elettronica e delle telecomunicazioni
Corso di laurea triennale in ingegneria informatica

Anno accademico 2015/2016

Implementazione di un algoritmo per la trasformazione di una sequenza basata sull'analisi iterativa di simmetrie locali

Laureandi:

Dennis Bontempi
Stefano Della Fiore

Relatore:

Prof. Riccardo Leonardi

Compendio

Nel seguente documento verrà presentata l'implementazione di un algoritmo finalizzato alla costruzione di una nuova trasformata, applicabile a sequenze discrete monodimensionali e bidimensionali, che si distingue da molte altre nell'approccio originale adottato per ridurre la ridondanza presente nel dominio di partenza.

Mentre molte trasformate, tra cui la *DCT* (alla base della compressione *JPEG*) e la *FFT*, si basano sull'approssimazione di un segnale mediante una somma di un numero finito di forme d'onda definite a priori, la nuova trasformata in questione, chiamata *"pari-dispari"* (*PD* in seguito), fa della sua semplicità virtù: essa trova la sua efficacia nella naturale presenza di proprietà simmetriche che caratterizzano porzioni di una vasta gamma di segnali. Per questo motivo la trasformata si basa, principalmente, sull'iterazione a più stadi dell'elementare scomposizione pari-dispari di una sequenza.

La breve trattazione teorica a riguardo lascia spazio alla presentazione di una nostra implementazione dell'algoritmo alla base della trasformazione, in linguaggio C, supportato da codice MatLab®. L'unione degli script nei due diversi linguaggi costituisce il mezzo utilizzato per test e confronti di seguito presentati e ampiamente commentati.

Il lavoro prende spunto dai precedenti studi del Professor Leonardi, dell'Ing. Guerrini e dell'Ing. Gnutti del dipartimento di Ingegneria dell'Informazione della nostra Università (degli Studi di Brescia).

Ringraziamenti

Un sentito grazie a tutte le persone speciali, conosciute in facoltà, che mi hanno accompagnato in questo percorso triennale di cui questa tesi è il culmine, in particolare Lorenzo, Mattia, Giulia, Leonardo, Roberto, sempre pronte ad ascoltarmi e aiutarmi.

Un ringraziamento a Stefano, che si è dimostrato la persona perfetta con cui condividere innumerevoli passioni e progetti.

Un ringraziamento a Rossana, senza la quale probabilmente mai avrei intrapreso il percorso di studi, a tutta la mia famiglia, per il loro costante supporto e le possibilità datemi, a tutti i miei già affermati amici come Marco, Alice, Pietro, Daniel e Alberto, per la loro insostituibile presenza e i loro consigli.

Un ultimo ringraziamento va a Sonia, punto di riferimento e compagna dal valore inestimabile, che da sempre mi sostiene (.. e mi sopporta).

Dennis Bontempi

Desidero innanzitutto ringraziare i miei genitori, Gianni e Loredana, per avermi permesso di frequentare l'università senza farmi mai mancare nulla, la mia compagna Veronica, per avermi sostenuto durante tutti questi anni di studi, spronandomi sempre nonostante le difficoltà.

Inoltre desidero ringraziare Umberto, Anna, Simona, Gloria, la nonna Rosa, per avermi accompagnato in questo percorso formativo, sostenendomi in ogni occasione.

Un ringraziamento dovuto a Dennis, con cui si è instaurata una costante collaborazione che ha portato e porterà ad una continua voglia di accrescere le mie conoscenze.

Infine, una dedica speciale a mia zia Carolina, che mi ha insegnato a lottare nonostante le avversità e che porterò sempre nel cuore, così come i nonni Amabile, Primo e Umberto.

Stefano Della Fiore

Inoltre, entrambi desideriamo ringraziare il nostro relatore, Professor Riccardo Leonardi, e allo stesso modo l'Ing. Alessandro Gnutti e l'Ing. Fabrizio Guerrini, che con il loro prezioso aiuto e insegnamento, hanno contribuito a rendere di grande valore il percorso intrapreso, nonché possibile la realizzazione dell'intero lavoro.

Indice

Introduzione

La **comunicazione** riveste da sempre un ruolo fondamentale nella società.
Nell'arco dei secoli, l'uomo ha più volte tentato, nelle maniere più svariate, di trovare una soluzione al problema che la **distanza** costituisce in questo campo: da quasi cent'anni, ormai, questo compito è affidato al settore dell'ingegneria dell'informazione.

Sin dalla sua nascita, storicamente associata alla figura di **C. E. Shannon** e al suo tanto celebre quanto innovativo lavoro *"A mathematical theory of communication"*, l'intera branca ha concentrato gran parte dei suoi sforzi nella **ricerca di una sempre più compatta rappresentazione del dato da trasmettere**: essa non solo consente di sfruttare al meglio la capacità limitata dei canali reali, ma permette anche un risparmio in termini di risorse da utilizzare nel processo di trasmissione stesso.

Varie **tecniche di compressione** si basano sull'utilizzo di trasformate che, lavorando in domini differenti da quello originale, permettono di concentrare gran parte dell'informazione in moli di dati di dimensione minore rispetto a quelli di partenza.

Poiché questi processi consistono solitamente nell'**eliminazione della ridondanza** presente nel segnale originale, nella ricerca di nuove trasformate adatte allo scopo è di primaria importanza **l'approccio adottato per l'individuazione di pattern ricorrenti** che caratterizzano l'intero dato o parte di esso.

La trasformata protagonista del documento, chiamata per ovvie ragioni *pari-dispari* (*PD*), punta al raggiungimento di questo obbiettivo sfruttando la **ricorrenza di simmetrie locali in segnali monidimensionali**.

Di seguito, dopo aver introdotto, le nozioni matematiche necessarie alla presentazione del lavoro (capitolo 1), esporremo, pur mantenendo una certa generalità, i passaggi algoritmici necessari al calcolo della trasformata *PD* (capitolo 2), base sul quale poggiano i capitoli seguenti.

Il terzo, infatti, presenta l'implementazione vera e propria in linguaggio C, insieme all'esposizione delle scelte fatte e degli aspetti tecnici dello script responsabile della trasformazione e dell'anti-trasformazione del dato.

Nel capitolo 4, al lettore viene introdotto il codice MatLab® di supporto, del quale ci siamo serviti per graficare i risultati ottenuti, aprendo le porte a considerazioni, analisi e confronti.

A tal proposito, ci siamo serviti di altre due trasformate per valutare effettivamente le capacità compattanti e non solo:

- La prima, data la sua importanza nel campo nonchè il suo enorme utilzzo, sarà proprio la *Fast Fourier Transform* (*FFT*), che computa il passaggio dal dominio temporale a quello delle frequenze.
- L'altra trasformata accostata alla *PD*, invece, è la *Discrete Cosine Transform* (*DCT*), alla base dell'arcinota compressione JPEG delle immagini.

I risultati delle varie elaborazioni fatte, nonchè prove dell'efficienza della trasformata, sono contenuti nel capitolo 5, suddiviso come segue:

- Nelle prime sezioni, vengono presentati graficamente i prodotti del lavoro sinergico tra i due script, insieme con alcune possibili interpretazioni agli esiti dei molteplici test eseguiti;
- L'ultima sezione del capitolo, e quindi del lavoro, è stata dedicata all'esposizione di come diverse scelte implementative impattino sui risultati, garantendo prestazioni differenti. In particolare analizzeremo graficamente le variazioni che un diverso ordinamento dei coefficienti del segnale trasformato, prima dell'azzeramento che precede la ricostruzione, produce nei risultati.

Capitolo 1

Nozioni di base

Per tutta la durata del documento, parleremo di segnali riferendoci nello specifico a **segnali a tempo discreto**. In particolare:

Definizione 1 (***Segnale a tempo discreto***)**.**
Chiamiamo sequenza o segnale a tempo discreto la successione di valori:

$$x[n] \in \mathbb{R}, \qquad n \in \mathbb{Z}$$

Inoltre:

Definizione 2 (***Energia di un segnale a tempo discreto***)**.**
Si dice energia di un segnale a tempo discreto la quantità definita come:

$$E_x = \sum_{n=-\infty}^{\infty} |x[n]|^2$$

Data una sequenza $x[n]$, si dimostra facilmente come essa possa sempre essere ottenuta dalla somma di due sequenze [1], una pari ed una dispari:

$$x[n] = x_p[n] + x_d[n]$$

ovvero tali per cui:

$$x_p[n] = x_p[-n], \qquad x_d[n] = -x_d[-n]$$

dove le suddette possono essere calcolate a partire da:

$$x_p[n] = \frac{x[n] + x[-n]}{2} \qquad x_d[n] = \frac{x[n] - x[-n]}{2}$$

Si noti che il prodotto scalare, definito come:

$$< \bar{x}_p, \bar{x}_d >^{1} = \sum_{n=-\infty}^{\infty} x_p[n] \cdot x_d[n]$$

risulta essere nullo, poichè sommatoria da $-\infty$ a ∞ di una sequenza dispari, ottenuta dal prodotto di una pari ed una dispari. Per questo motivo x_p ed x_d si dicono ortogonali tra loro. Perciò, dalla definizione sopra riportata di energia di un segnale a tempo discreto:

$$E_x = \sum_{n=-\infty}^{\infty} |x[n]|^2 = \sum_{n=-\infty}^{\infty} |x_p[n] + x_d[n]|^2 =$$

$$\sum_{n=-\infty}^{\infty} |x_p[n]|^2 + \sum_{n=-\infty}^{\infty} 2 \cdot |x_p[n] \cdot x_p[n]| + \sum_{n=-\infty}^{\infty} |x_p[n]|^2$$

da cui:

$$E_x = E_p + E_d$$

Dalla definizione di segnale pari e dispari precedentemente fornita, risulta ovvio come le sequenze risultino perfettamente rappresentabili sfruttando solamente l'informazione relativa alla metà del supporto totale, come riportato graficamente in **figura 1.1**.
È quindi possibile risalire alla sequenza di partenza $x[n]$ a partire da una nuova sequenza definita come:

$$\hat{x}[n] = \begin{cases} x_p[n], & n \geq 0 \\ x_d[n], & n < 0 \end{cases}$$

rappresentabile come riportato in **figura 1.2**, per esempio procedendo come segue:

$$x_p[n] = \hat{x}[n] \cdot \varepsilon^2[n] + \hat{x}[-n] \cdot \varepsilon[-n-1]$$
$$x_d[n] = \hat{x}[n] \cdot \varepsilon[-n-1] - \hat{x}[-n] \cdot \varepsilon[n-1]$$

[1] La notazione qua utilizzata allude alla possibilità di trattare le sequenze come vettori, e lo spazio delle sequenze come spazio vettoriale (si veda [1])

[2] $\varepsilon := \begin{cases} 1, & n \geq 0 \\ 0, & n < 0 \end{cases}$

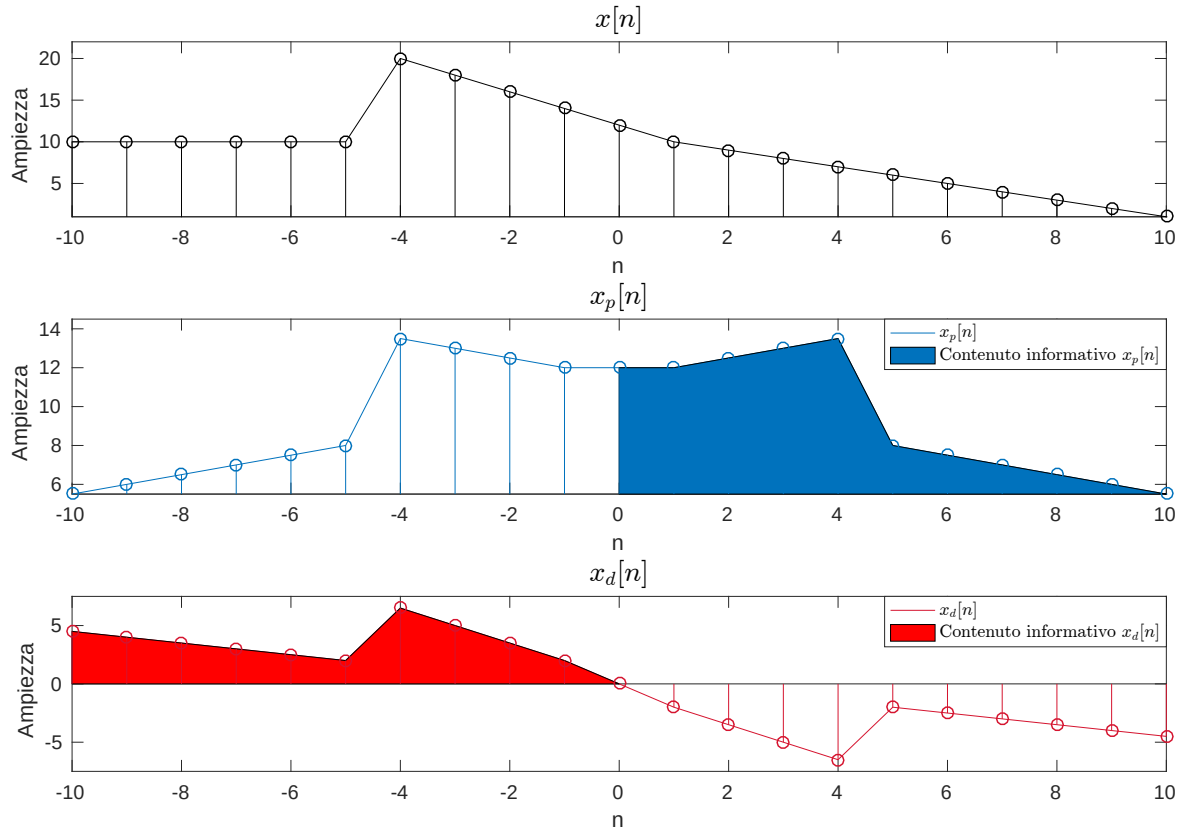


Figura 1.1: *Scomposizione della sequenza $x[n]$.*

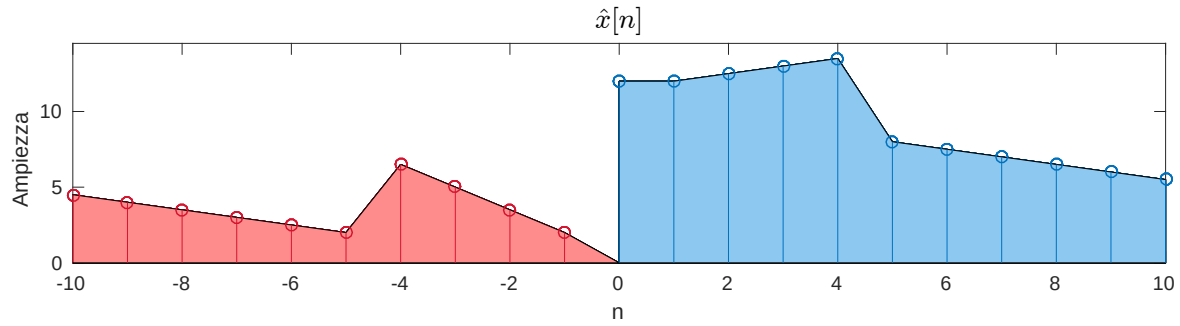


Figura 1.2: *Nuova sequenza $\hat{x}[n]$.*

Si prenda ora in considerazione il caso in cui le sequenze considerate siano definite su un supporto finito L con **asse di simmetria** $\frac{\mathbf{L+1}}{\mathbf{2}}$. La definizione precedentemente fornita diventa:

Definizione 3 (***Segnale a tempo discreto di supporto L***)**.**
Dato un numero $L \in \mathbb{N}^+$ (intero, maggiore di zero), chiamiamo sequenza di lunghezza L o segnale a tempo discreto di supporto L la successione di valori:

$$x[n] \in \mathbb{R}, \qquad n = 1, 2, 3, \ldots, L$$

Si noti come, in questo particolare caso, le sequenze pari e dispari possano essere calcolate a partire da:

$$x_p[n] = \frac{x[n] + x[L+1-n]}{2} \qquad x_d[n] = \frac{x[n] - x[L+1-n]}{2}$$

Anche stavolta è possibile definire una sequenza $\hat{x}[n]$ dalla quale è possibile risalire a quella di partenza:

$$\hat{x}[n] = \begin{cases} x_p[n], & 1 \leq n \leq \left\lfloor \frac{L+1}{2} \right\rfloor \\ x_d[n], & L - \left\lfloor \frac{L}{2} \right\rfloor < n \leq L \end{cases}$$

Esempio 1 (***Calcolo di $\hat{x}[n]$ - sequenza con L pari***)**.**
Data la sequenza a supporto finito, con $L = 4$, $x[n] = \{1, 4, -2, 3\}$, ovvero tale per cui:

$$x[1] = 1, \qquad x[2] = 4, \qquad x[3] = -2, \qquad x[4] = 3$$

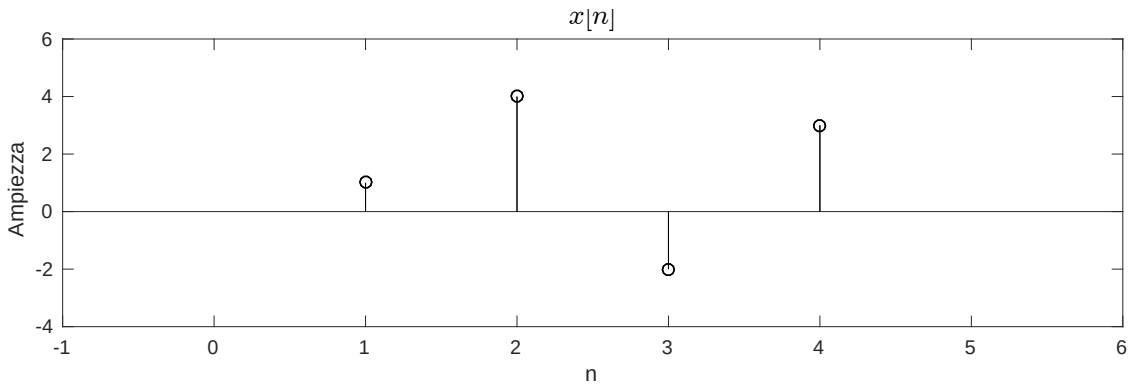


Figura 1.3: *Sequenza $x[n]$, $L = 4$.*

*graficata in **figura 1.3**, è possibile calcolare parte pari e parte dispari, riportate in nella **figura 1.4**:*

$$x_p[n] = \frac{x[n] + x[L+1-n]}{2} = \{2, 1, 1, 2\} \quad x_d[n] = \frac{x[n] - x[L+1-n]}{2} = \{-1, 3, -3, 1\}$$

*da cui, come chiaro dalla **figura 1.5**:*

$$\hat{x}[n] = \{2, 1, -3, 1\}$$

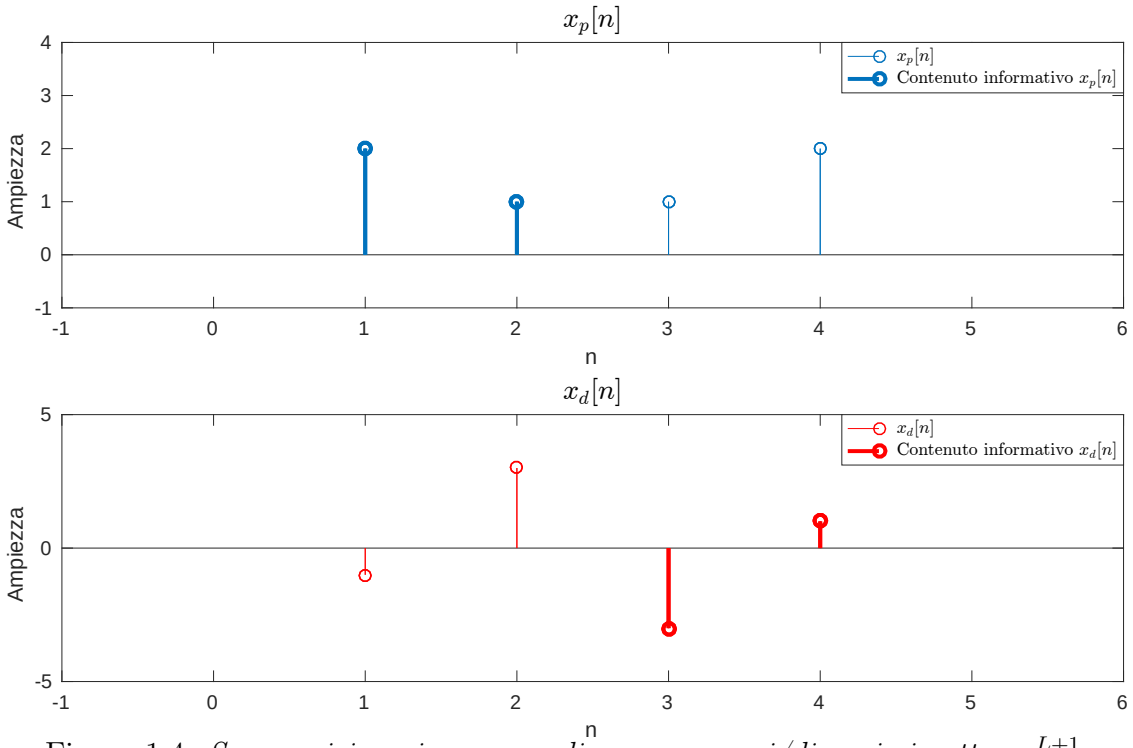


Figura 1.4: *Scomposizione in somma di sequenze pari/dispari rispetto a* $\frac{L+1}{2}$.

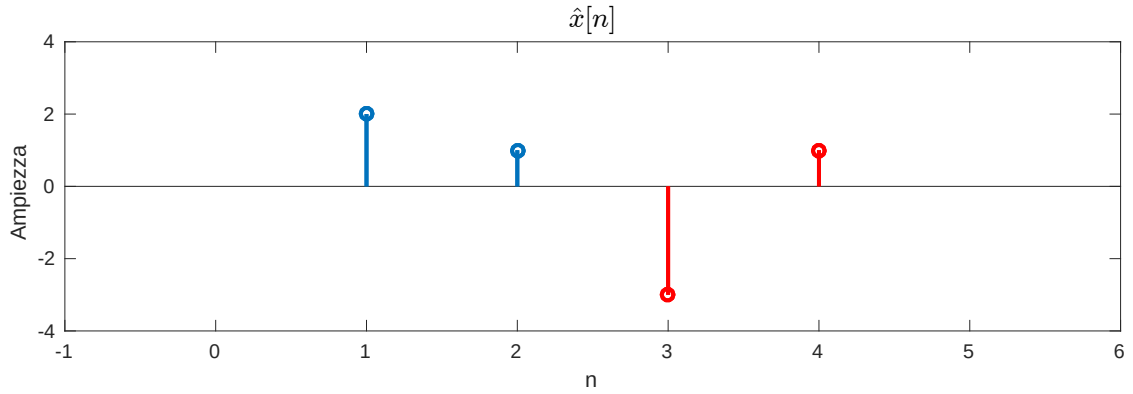


Figura 1.5: *Nuova sequenza* $\hat{x}[n]$.

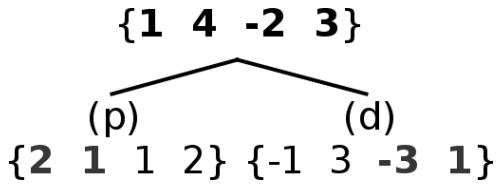


Figura 1.6: *Schema ad albero della scomposizione in* $x_p[n]$ *e* $x_d[n]$ *della sequenza*

Esempio 2 (***Calcolo di $\hat{x}[n]$ - sequenza con L dispari***)**.**
Data la sequenza a supporto finito, con $L = 5$, $x[n] = \{0,\ -2,\ 4,\ -4,\ -4\}$, *é possibile calcolare parte pari e parte dispari:*

$$x_p[n] = \frac{x[n] + x[L+1-n]}{2} = \{-2, -3, 4, -3, -2\}$$

$$x_d[n] = \frac{x[n] - x[L+1-n]}{2} = \{2,\ 1,\ 0,\ -1,\ -2\}$$

da cui:

$$\hat{x}[n] = \{-2,\ -3,\ 4,\ -1, 2\}$$

Generalizzando ulteriormente, è possibile scomporre una sequenza di L campioni rispetto ad un **asse di simmetria n_0 arbitrario** tale per cui:

$$k \in \mathbb{N} \wedge k \in [2, 2 \cdot L], \qquad n_0 = \frac{k}{2}$$

utilizzando le seguenti:

$$x_p[n; n_0] = \frac{x[n] + x[2 \cdot n_0 - n]}{2} \qquad x_d[n; n_0] = \frac{x[n] - x[2 \cdot n_0 - n]}{2}$$

dopo aver effettuato un padding (alla sinistra della sequenza, sempre rispettando $n \in \mathbb{N}$) di $L - n_0$ campioni. Se prendiamo, a scopo esplicativo, la sequenza dell'esempio 1, posto $n_0 = 1$ otteniamo ciò che è rappresentato in **figura 1.7**:

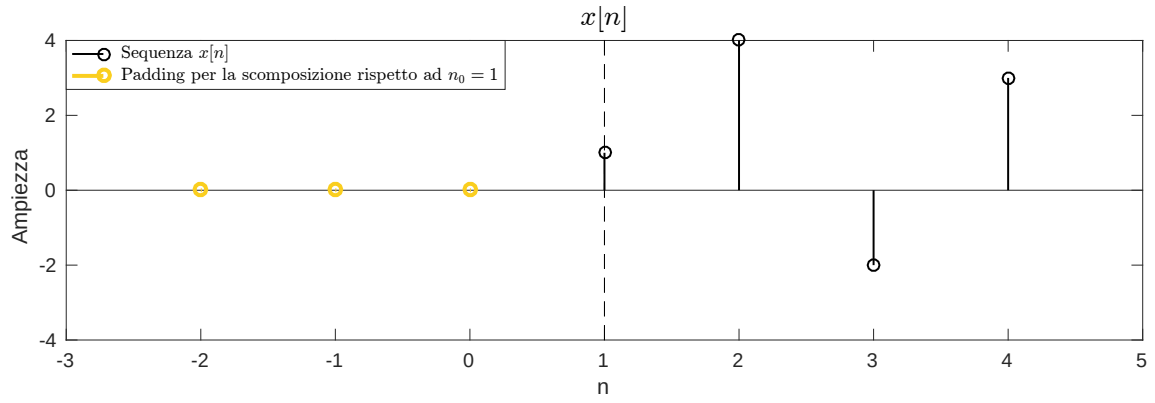


Figura 1.7: *Nuova sequenza con padding di* $L - n_0 = 3$ *samples*

Possiamo poi procedere con la scomposizione secondo il metodo sopra esposto.
Si noti come, seguendo le definizioni precedentemente presentate, ogni sequenza $x[n]$ a supporto finito L risulti definita solamente su numeri naturali maggiori di 0, mentre $n_0 \in \mathbb{Q}$. Per questo motivo l'asse di simmetria può cadere sia su un campione (e.g. segnale con L dispari, a simmetria centrale) sia nel mezzo di due campioni (e.g. segnale con L pari, a simmetria centrale).

Capitolo 2

Analisi dell'algoritmo

Nell'ambito della teoria dei segnali, l'obbiettivo comune a tutte le trasformate è la proiezione in un nuovo spazio nel quale la rappresentazione e la manipolazione di un segnale (o di una sequenza) risulti più conveniente (per esempio che sia in grado di redistribuire gran parte dell'energia originale su pochi coefficienti).

Altro requisito fondamentale è l'invertibilità della trasformata stessa, ovvero l'esistenza di un operatore inverso che garantisca la possibilità di tornare alla rappresentazione di partenza. Sfruttando i concetti introdotti nel **capitolo 1** e applicando l'algoritmo descritto in seguito, è possibile arrivare a definire due nuove trasformate, che basano la loro efficienza sull'eventuale esistenza di simmetrie locali all'interno di segnali monodimensionali.

2.1 Trasformata a scomposizione centrale

L'algoritmo in questione consta di due principali passaggi, atti poi ad essere reiterati:

1. Si scompone la sequenza in oggetto in x_p e x_d;
2. Si itera il procedimento sulla parte informativa delle nuove sequenze ottenute, ovvero sui $\left\lfloor \frac{L+1}{2} \right\rfloor$ campioni di x_p e gli ultimi $\left\lfloor \frac{L}{2} \right\rfloor$ campioni di x_d.

Il processo porta alla costruzione di un albero binario (come si può notare dalle **figure 2.1, 2.2, 2.3 e 2.4**, annesse agli esempi che seguono), ovvero una struttura, descrivibile mediante la classica notazione a nodi ed archi, nella quale ogni nodo risulta essere di grado massimo 2 (nel nostro caso, esattamente 2).

L'albero ha come nodo radice la sequenza di partenza di lunghezza L, e come nodi finali L sequenze di lunghezza unitaria (in seguito chiamate foglie) che se concatenate forniscono il segnale trasformato.

Per far si che l'energia del segnale $x[n]$ venga mantenuta dopo la trasformazione, inoltre, occorre moltiplicare ogni $\hat{x}[n]$ ottenuto (i.e. ogni livello) per il fattore $\sqrt{2}$.

Esempio 3 (***Trasformazione - sequenza con supporto L pari***)**.**
Data la sequenza a supporto finito, con $L = 4$, $x[n] = \{1,\ 4,\ -2,\ 3\}$, *iterando i passaggi sopra esposti, si ottiene la trasformata della sequenza:*

$$X[n] = \{3,\ -1,\ -2,\ 2\}$$

seguendo lo schema:

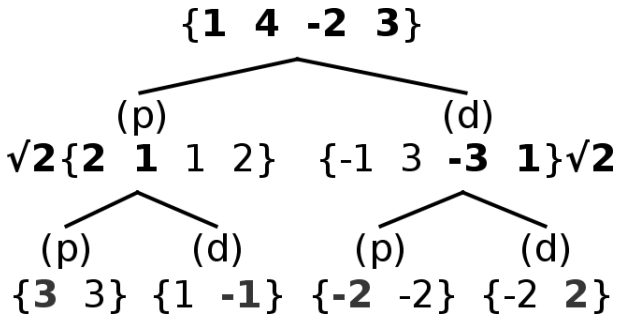


Figura 2.1: *Schema della scomposizione iterata sulla sequenza con L pari*

Esempio 4 (***Trasformazione - sequenza con supporto L dispari***[1])**.**
Data la sequenza a supporto finito, con $L = 5$, $x[n] = \{0,\ -2,\ 4,\ -4\ \ -4\}$, *iterando i passaggi sopra esposti, si ottiene la trasformata della sequenza:*

$$X[n] = \{-\frac{5}{\sqrt{2}} + 2,\ -\frac{\sqrt{2}}{2} - 2,\ 2 + 2 \cdot \sqrt{2},\ -3,\ -1\}$$

seguendo lo schema:

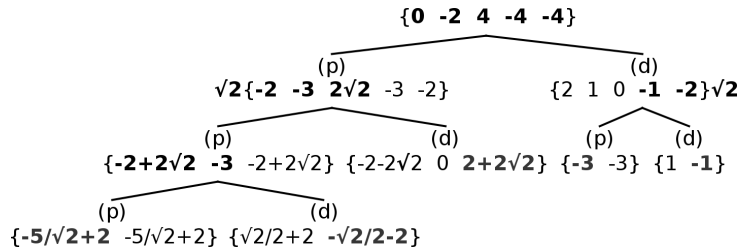


Figura 2.2: *Schema della scomposizione iterata sulla sequenza con L dispari*

Per giudicare la bontà della trasformata, iniziamo valutando la sua capacità di concentrare l'energia della sequenza di partenza di lunghezza L in pochi campioni della sequenza trasformata. In questi termini, la trasformata in questione risulta essere estremamente performante per segnali costanti (i.e. simmetrici scomponibili in somma di sequenze a loro volta simmetriche), oppure con forte simmetria rispetto all'asse $\frac{L+1}{2}$, come dimostrano gli esempi che seguono.

[1]Casistica ampiamente sviluppata nella **sezione 2.2**

Esempio 5 (***Trasformazione di una sequenza costante con supporto L pari, potenza di* 2**)**.**
Data la sequenza a supporto finito, con $L = 8$, $x[n] = \{1,\ 1,\ 1,\ 1,\ 1,\ 1,\ 1,\ 1\}$, *iterando i passaggi sopra esposti, si ottiene la trasformata della sequenza:*

$$X[n] = \{2 \cdot \sqrt{2},\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0\}$$

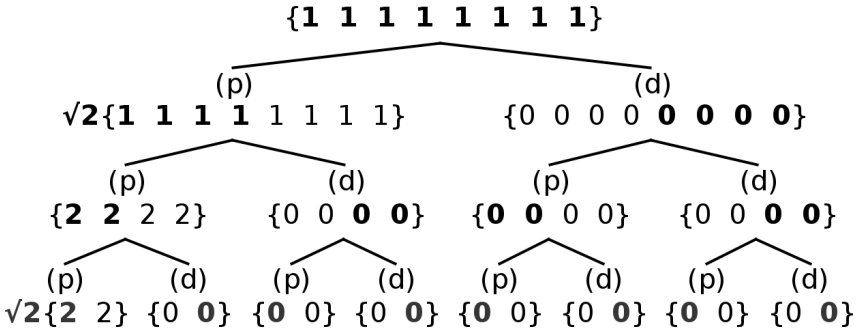


Figura 2.3: *Schema della scomposizione iterata sulla sequenza costante* ($L = 8$)

Come si osserva in ***figura 2.3****, la simmetria rispetto ad* $\frac{L+1}{2} = 4$ *di* $x[n]$ *e di ogni parte informativa di* $x_p[n]$ *e* $x_d[n]$ *fa si che il segnale trasformato concentri tutta l'energia in un solo campione.*

Esempio 6 (***Trasformazione di una sequenza simmetrica rispetto ad*** $\frac{L+1}{2}$**, con *L pari, potenza di* 2**)**.**
Data la sequenza a supporto finito, con $L = 8$, $x[n] = \{2,\ 4,\ 2,\ 0,\ 0,\ -2,\ -4,\ -2\}$, *iterando i passaggi sopra esposti, si ottiene la trasformata della sequenza:*

$$X[n] = \{0,\ 0,\ 0,\ 0,\ -4 \cdot \sqrt{2},\ -2 \cdot \sqrt{2},\ -2 \cdot \sqrt{2},\ 0\}$$

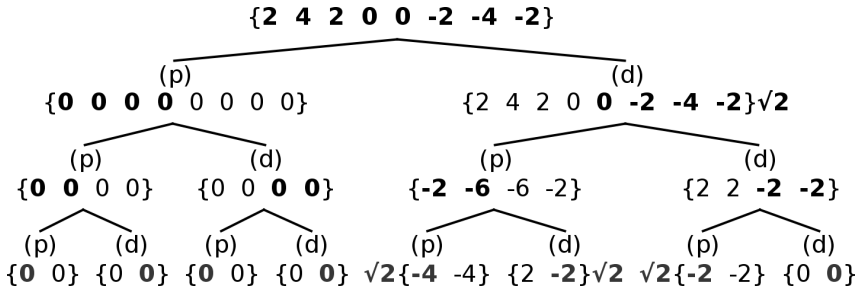


Figura 2.4: *Schema della scomposizione iterata sulla sequenza simmetrica dispari*

La simmetria rispetto ad $\frac{L+1}{2} = 4$ *di* $x[n]$ *fa si che il segnale trasformato concentri tutta l'energia in pochi campioni, pur distando dal caso ottimo presentato nell'esempio precedente (a causa della mancata simmetria centrale delle sottosequenze).*

La capacità della trasformata di concentrare l'energia in pochi campioni si rivela fondamentale per applicazioni quali la compressione dei dati, dove è di primaria importanza la bontà con la quale l'antitrasformata é in grado di ricostruire il segnale originale partendo da un numero limitato di campioni.

Un esempio, classico nella teoria dei segnali, é fornito dalla trasformata di Fourier (implementata nel discreto come Fast Fourier Transform - FFT).

Basandosi sulle intuizioni dell'omonimo creatore, secondo il quale "ogni segnale di periodo T puó essere espresso come somma di (eventualmente infinite) sinusoidi di frequenza f, eventualmente sfasate di una quantitá ϕ":

$$x(t) = a_1 \cdot s_1(t) + a_2 \cdot s_2(t) + ... + a_k \cdot s_k(t)$$

Essa permette di ricostruire segnali periodici a partire da relativamente pochi campioni, ordinando le sinusoidi in base al coefficiente di Fourier a_i e ricostruendo il segnale con le prime n utilizzando l'operazione inversa alla trasformata.

Allo stesso modo, possiamo valutare l'efficienza della trasformata presentata in questa sezione eliminando (i.e. assegnando valore nullo) $k < L$ delle L foglie totali (quelle associate ad energia minore), costruendo l'albero a partire dal fondo e confrontando la ricostruzione con quello di partenza. Per chiarezza, riprendiamo alcune delle sequenze precedentemente oggetto di esempi, indicando con apice barrato le foglie tagliate.

Esempio 7 (***Anti-trasformazione della sequenza dell'esempio 3***)**.**
Azzerando 1 *foglia su* 4 *(*$k = 1$*), si ottiene la sequenza ricostruita:*

$$x_r[n] = \{1,\ 3,\ 0,\ 2\}$$

a fronte di:

$$x[n] = \{1,\ 4,\ -2,\ 3\}$$

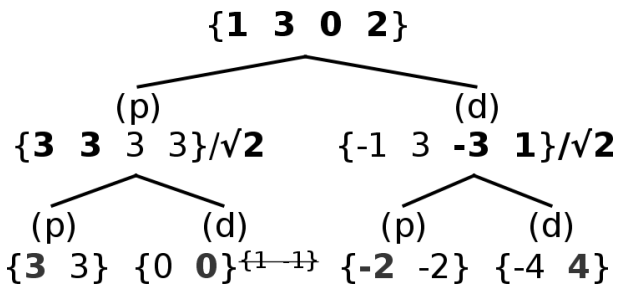


Figura 2.5: *Ricostruzione della sequenza a partire dal* 75% *dei campioni*

Esempio 8 (***Anti-trasformazione della sequenza dell'esempio 4***)**.**
Azzerando 3 *foglie su* 5 *(*$k = 3$*), si ottiene la sequenza ricostruita:*

$$x_r[n] = \{1 - \sqrt{2},\ 1,\ 0,\ -3,\ -2 + \sqrt{2}\}$$

a fronte di:

$$x[n] = \{0,\ -2,\ 4,\ -4,\ -4\}$$

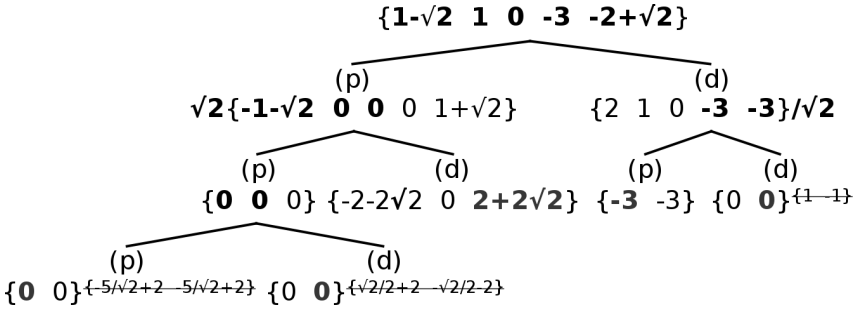


Figura 2.6: *Ricostruzione della sequenza a partire dal* 40% *dei campioni*

Esempio 9 (***Anti-trasformazione della sequenza dell'esempio 6***)**.**
Azzerando 6 *foglie su* 8 *(*$k = 6$*), ottengo la sequenza ricostruita:*

$$x_r[n] = \{3,\ 3,\ 1,\ 1,\ -1,\ -1,\ -3,\ -3\}$$

a fronte di:

$$x[n] = \{2,\ 4,\ 2,\ 0,\ 0,\ -2,\ -4,\ -2\}$$

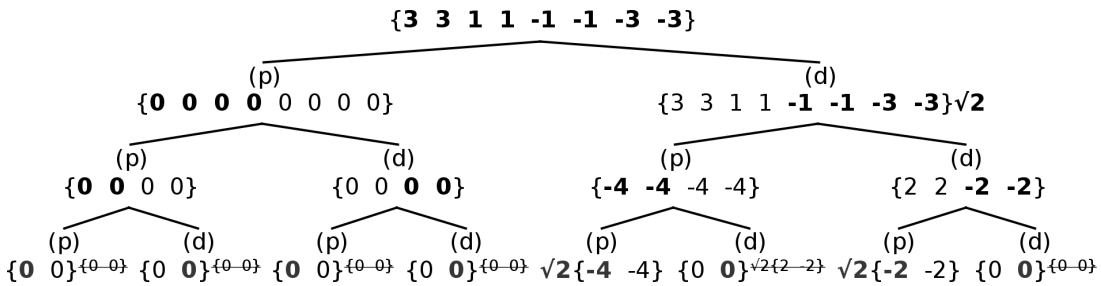


Figura 2.7: *Ricostruzione della sequenza a partire dal* 25% *dei campioni*

Chiaramente, a parità di percentuale di foglie tagliate, maggiore è la simmetria riscontrata nel segnale di partenza (e nelle sequenze in cui esso viene scomposto), minore sarà l'errore commesso in ricostruzione: se nell'esempio soprastante (**esempio 9**) avessimo azzerato il 60% delle foglie (i.e. 5 su un totale di 8), avremmo ricostruito con perfezione la sequenza di partenza (poiché **la spiccata simmetria consente la concentrazione dell'energia in un basso numero di foglie**).

Per questo motivo, la trasformata presentata in questa sezione si rivela performante solamente per sequenze con simmetria rispetto all'asse $\frac{L+1}{2}$, in particolar modo se anche le sequenze in cui essa viene scomposta hanno simmetria centrale.

Si rende dunque necessaria la ricerca di un modo per valorizzare le simmetrie locali delle sequenze (ovvero simmetrie rispetto ad assi differenti da $\frac{L+1}{2}$).
Di questo processo discuteremo nella prossima sezione.

2.2 Trasformata a scomposizione ottima

L'approccio presentato nella **sezione 2.1** risulta essere di estrema facilitá implementativa quanto di limitata efficienza per segnali non conformi alle caratteristiche precedentemente presentate. È facile dedurre come, se fosse possibile trovare un metodo per convogliare ad ogni iterazione la maggior parte dell'energia (i.e. contenuto informativo) del segnale in una delle due sequenze (x_p o x_d), si arriverebbe al poter ricostruire il segnale di partenza (di lunghezza L) a partire da $\frac{L}{2}$ (o $\frac{L+1}{2}$, nel caso di sequenza dispari) campioni, commettendo un errore piú o meno accettabile in funzione della bontá del processo in discussione.

Per questo motivo, uno dei risultati piú importanti a cui si é giunti é l'identificazione di un asse di simmetria, che d'ora in avanti chiameremo n_0, grazie alla quale è possibile di dividere la sequenza in due parti:

- Una prima sottosequenza, che chiameremo x', che se scomposta in pari/dispari risulta convogliare la maggior parte dell'energia in una delle due (massimizzando E_p o E_d);
- Una seconda sottosequenza, che chiameremo x_s, sulla quale verrá reiterato l'intero procedimento.

Questo nuovo approccio fa si che la trasformazione del dato avvenga basando ogni iterazione su un asse che garantisce una **scomposizione** *"ottima"*, ovvero in grado di **massimizzare l'energia di una delle due sequenze** ($x_p[n]$ o $x_s[n]$): il prezzo da pagare é, chiaramente, il dover trascurare alcuni campioni per fare si che questo avvenga (i.e. la creazione di un'ulteriore sequenza x_s, che dovrá essere trattata separatamente, e porterá alla costruzione di un albero non simmetrico nella forma) [2].

L'asse in questione viene individuata attraverso la ricerca del massimo (in valore assoluto) dell'autoconvoluzione del segnale di partenza:

$$2 \cdot n_0 = \underset{m}{\operatorname{argmax}} |(x * x)[m]| + 1^*$$

In questo nuovo contesto l'algoritmo si compone di tre passaggi:

1. Si computa l'autoconvoluzione della sequenza e se ne individua il/i massimo/i: nel caso nel quale ce ne sia piú di uno, la strada ideale da percorrere sarebbe scomporre rispetto a tutti i candidati e verificare quale sia alla fine la migliore trasformata;
2. Si scompone la sequenza in oggetto in $x_p[n; n_0]$ e $x_d[n; n_0]$ (rispetto all'asse di simmetria trovato al punto precedente). Se $n_0 \neq \frac{L+1}{2}$ questo passaggio si applicherá solo su x'.
3. Si itera il procedimento sulla parte informativa delle nuove sequenze ottenute, ovvero sui primi $\lfloor \frac{L+1}{2} \rfloor$ campioni di x_p e gli ultimi $\lfloor \frac{L}{2} \rfloor$ campioni di x_d, e sulla sequenza x_s.

*Il termine " + 1" è legato alla **Definizione 3**, **Capitolo 1**.

Esempio 10 (***Scomposizione ottima di una sequenza***)**.**
Data la sequenza a supporto finito, con $L = 9$, $x[n] = \{1,\ 3,\ -2,\ 6,\ 7,\ -9,\ 9,\ 10,\ 2\}$, *seguendo i passaggi sopra elencati:*

1. *Viene computato il massimo dell'autoconvoluzione della sequenza e ne viene estratto l'indice (**figura 2.8**):*

$$n_0 = \frac{\underset{m}{\operatorname{argmax}} |(x * x)[m]| + 1}{2} = \frac{11 + 1}{2} = 6$$

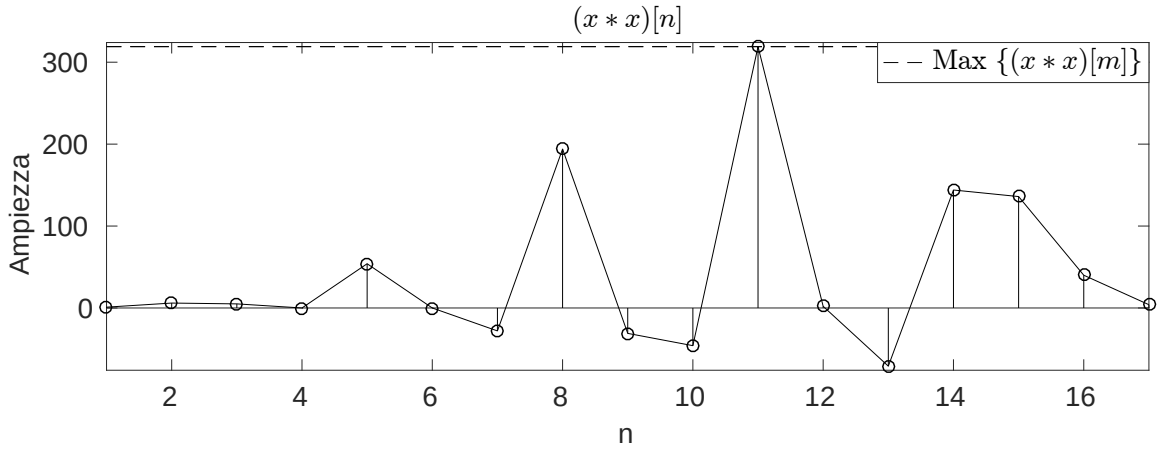


Figura 2.8: *Calcolo dell'indice associato al massimo valore assoluto di* $|(x * x)|$

2. *Viene calcolata la quantitá:*

$$d = min\{n_0 - L, L - n_0\} = 3$$

*La sequenza di partenza, come illustrato in **figura 2.9**, viene ora divisa in:*

$$x'[n] = \{x[n] : n \in [n_0 - d; n_0 + d]\} = \{-2,\ 6,\ 7,\ -9,\ 9,\ 10,\ 2\}$$

in nero nell'immagine, di lunghezza $L' = 2 \cdot d = 6$, *e* $x_s[n]$, *in verde, composta dai rimanenti* $L - L' = 2$ *campioni. Dopodiché si passa al calcolo della parte pari e della parte dispari della sequenza* $x'[n]$, *di cui verrà preservata solo la parte informativa (**figura 2.10**).*

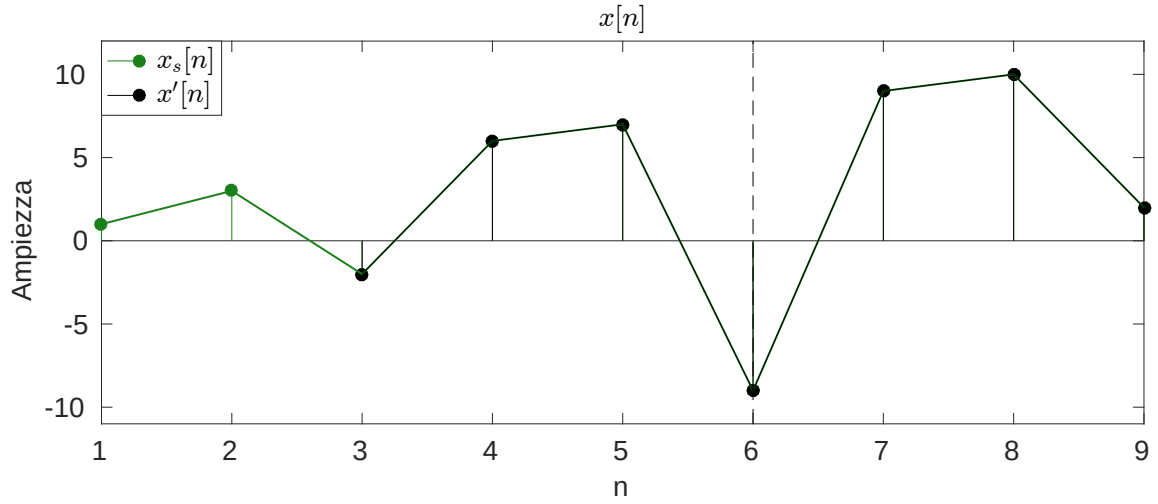


Figura 2.9: *Divisione della sequenza nelle due parti* $x'[n]$ *e* $x_s[n]$

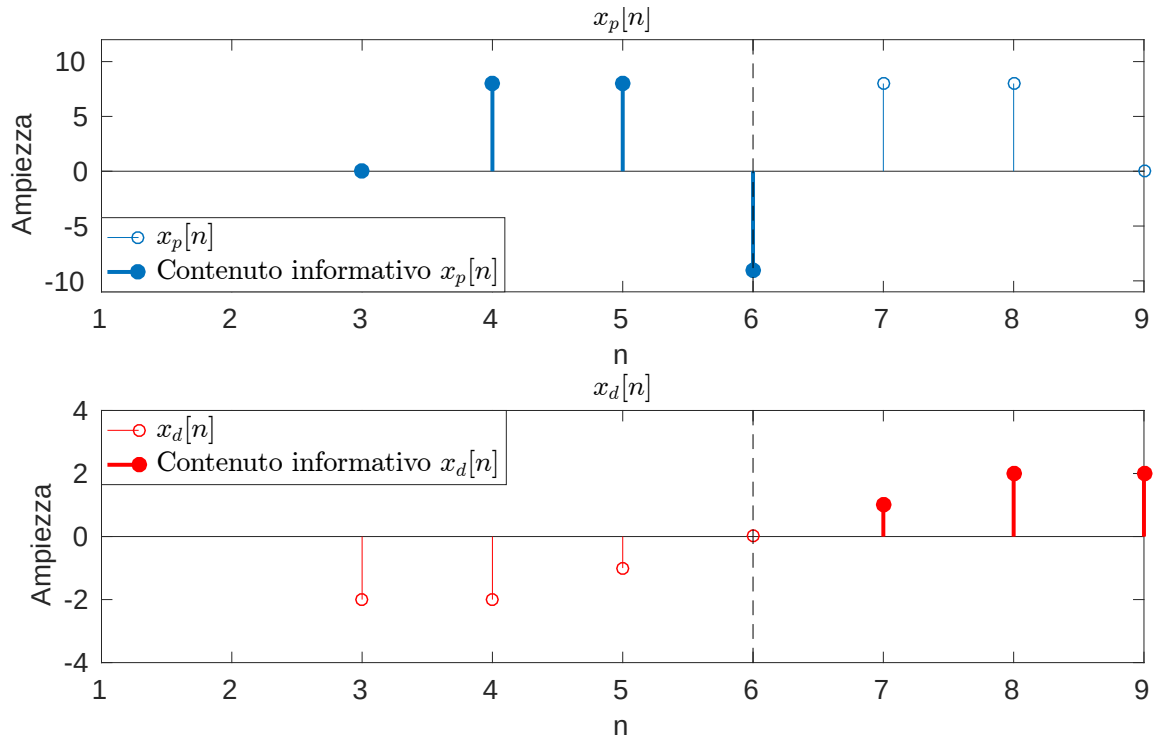


Figura 2.10: *Scomposizione in parte pari e dispari della sequenza* $x'[n]$

L'iterazione del processo su ogni nuova sequenza prodotta porta alla costruzione di una struttura dati ad albero ternario, come illustrato in **figura 2.11**.

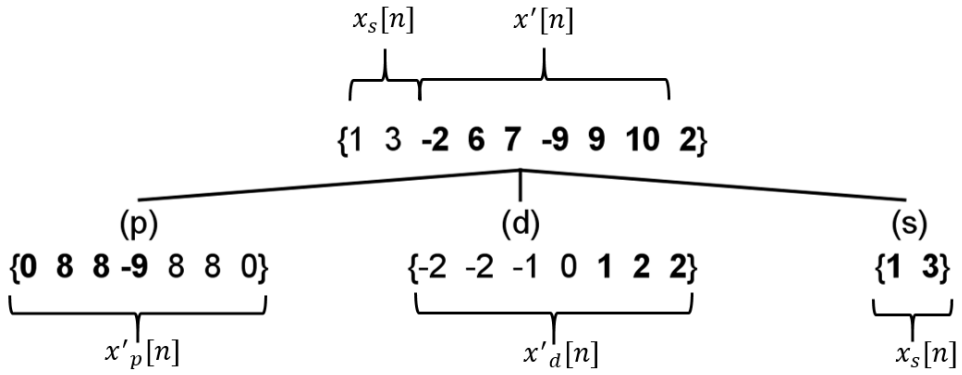


Figura 2.11: *Albero ternario creato dalla prima iterazione sulla sequenza in oggetto.*

Si noti come ad ogni iterazione, il numero di nodi figli generato dipende dalla simmetria della sequenza $x[n]$ presa in considerazione:

- Se $x[n]$ è una **sequenza a simmetria centrale pari o dispari**(i.e. $n_0 = \frac{L+1}{2}$), non sussiste la necessità di dividere in due sequenze x' e x_s. Essa viene direttamente scomposta nelle due sequenze x_p e x_d, delle quali terremo solamente la parte informativa, secondo il procedimento presentato nelle sezioni precedenti. In particolare:

  i. Se L é dispari (i.e. l'asse di simmetria cade su un campione), la parte pari risulterà avere un campione in più di quella dispari: **il sample in n_0 viene aggregato alla prima**, poiché in tutti i casi vale $\boldsymbol{x'_d[n_0] = 0}$ (**figure 2.12, 2.13 e 2.14**);

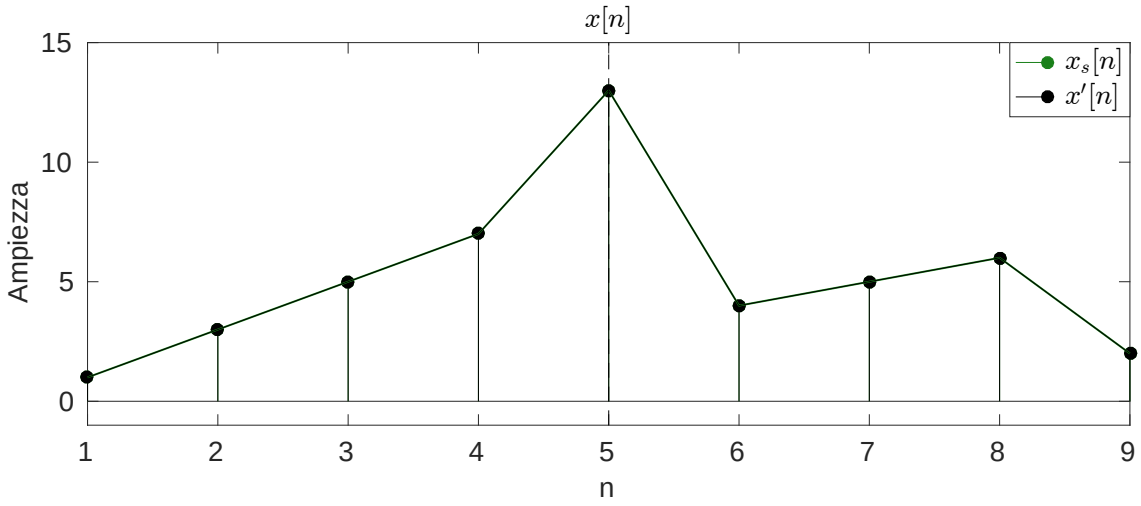


Figura 2.12: *Calcolo dell'indice associato al massimo valore assoluto di* $|(x * x)|$

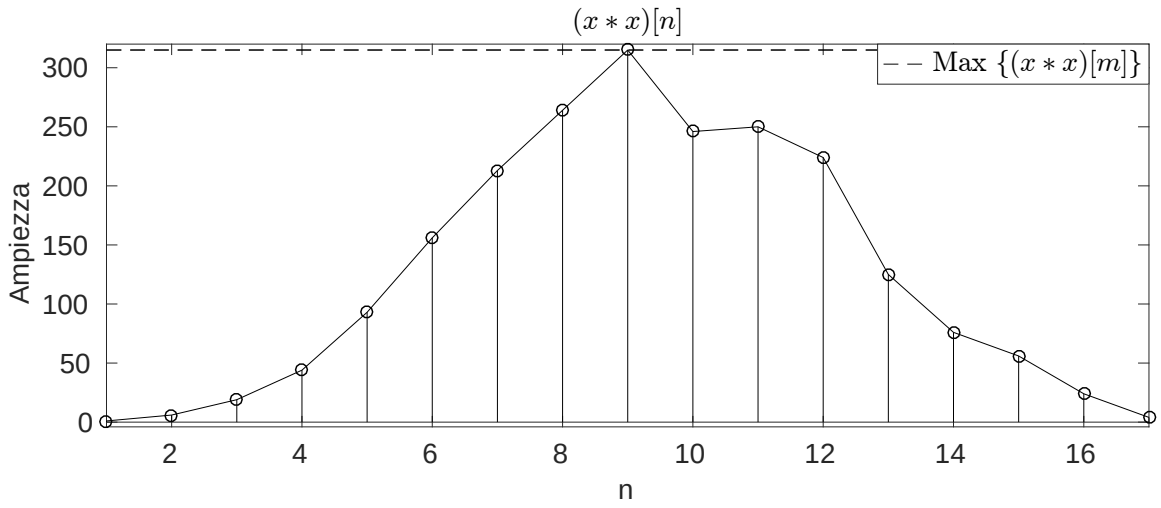


Figura 2.13: *Autoconvoluzione della sequenza*

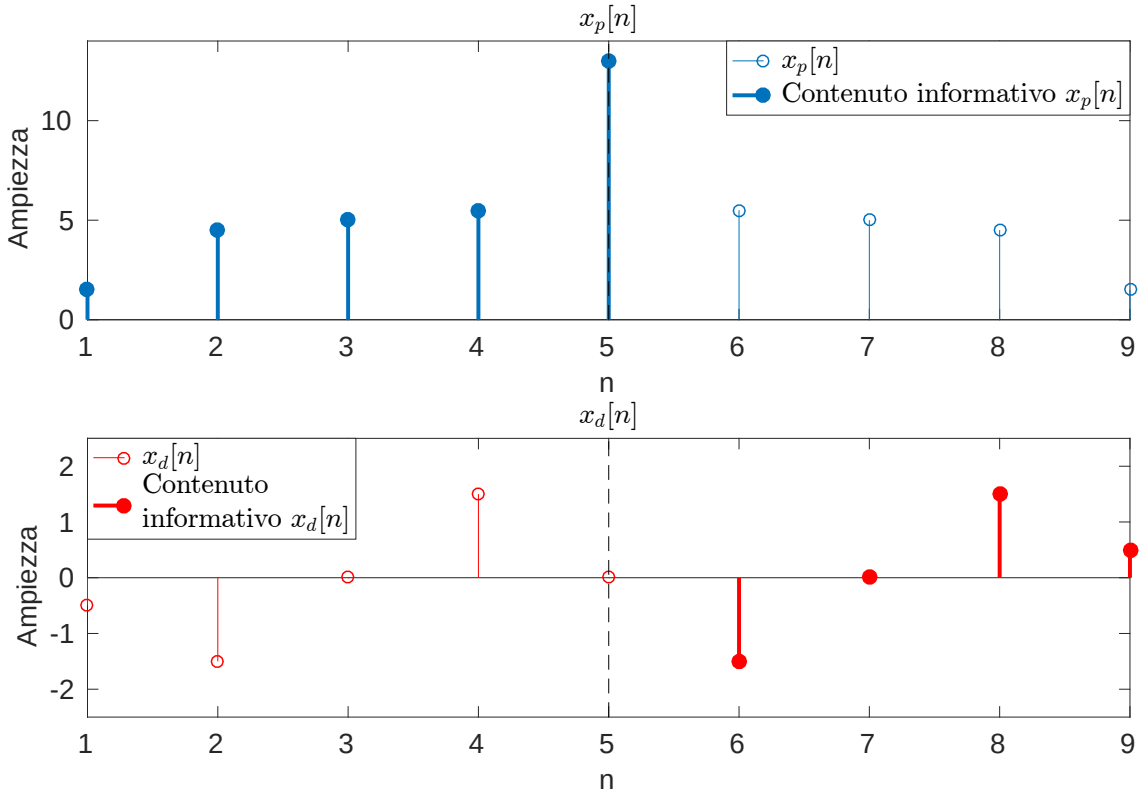


Figura 2.14: *Scomposizione della sequenza a simmetria centrale*

ii. Se L è pari (i.e. l'asse di simmetria cade tra due campioni), le due parti avranno lunghezza identica ($\frac{L}{2}$) **(figure 2.15, 2.16 e 2.17)**

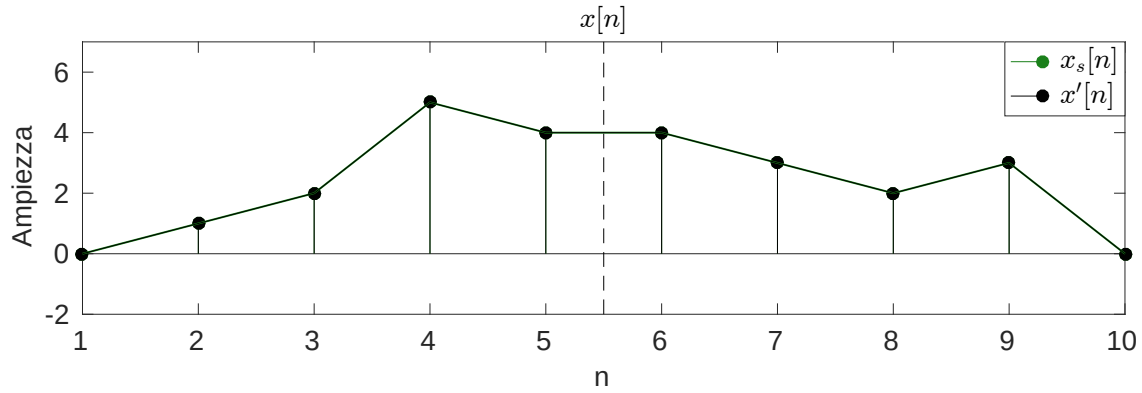


Figura 2.15: *Sequenza a simmetria centrale (L pari)*

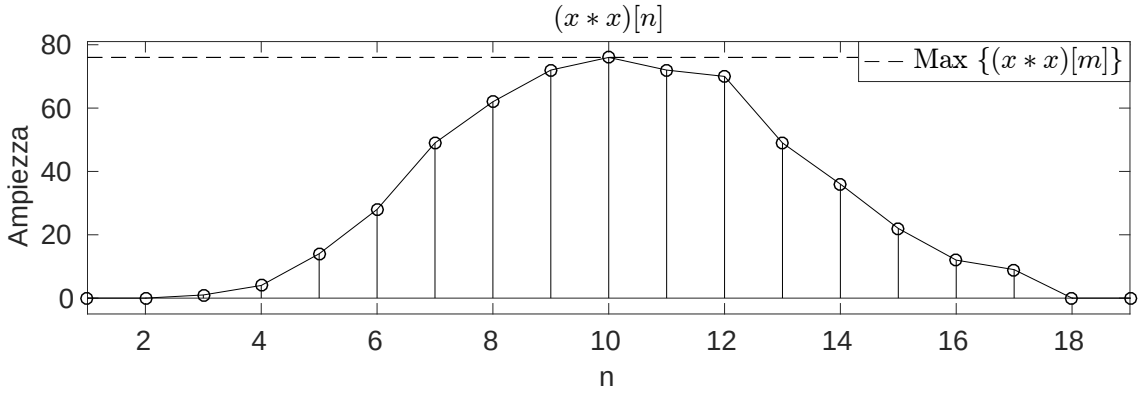


Figura 2.16: *Autoconvoluzione della sequenza*

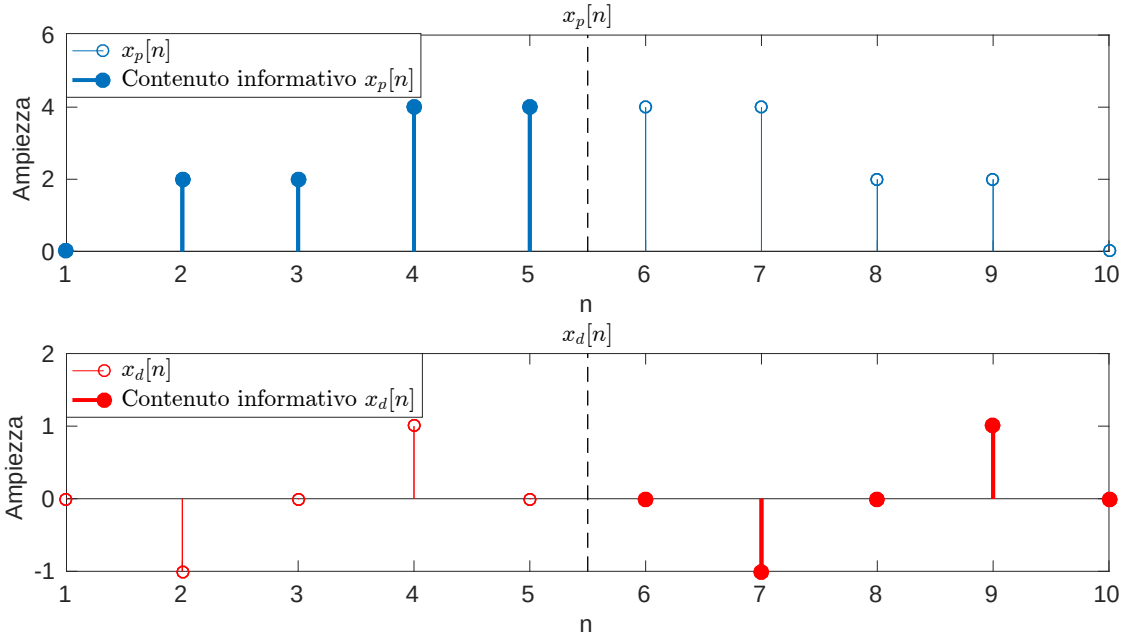


Figura 2.17: *Scomposizione della sequenza a simmetria centrale*

- Se $x[n]$ **non é una sequenza a simmetria centrale pari o dispari** (i.e. $n_0 \neq \frac{L+1}{2}$), invece, é d'obbligo l'individuazione delle due sequenze x' e x_s. Esistono alcuni casi particolari:

  i. Se n_0 cade sul primo (o sull'ultimo) campione della sequenza, essa viene scomposta in due parti: la prima (x') consta esclusivamente del campione stesso (che diverrá dunque una foglia), mentre la seconda (x_s) di tutti gli altri campioni;

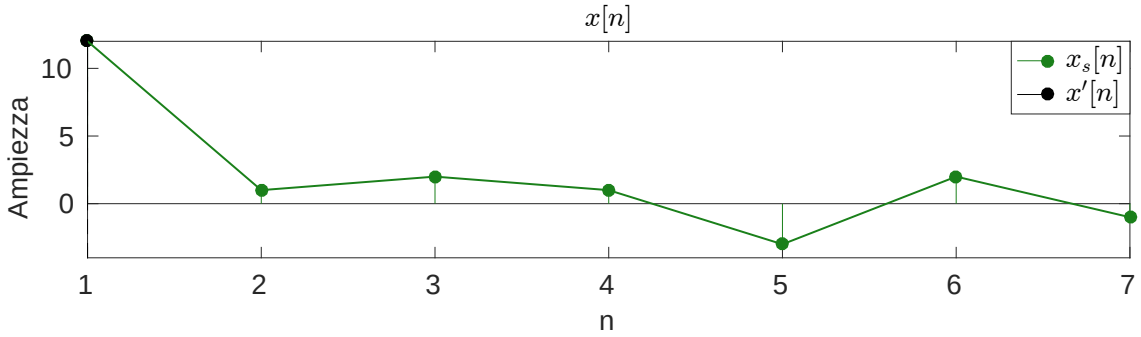


Figura 2.18: *Sequenza con asse n_0 sul primo campione*

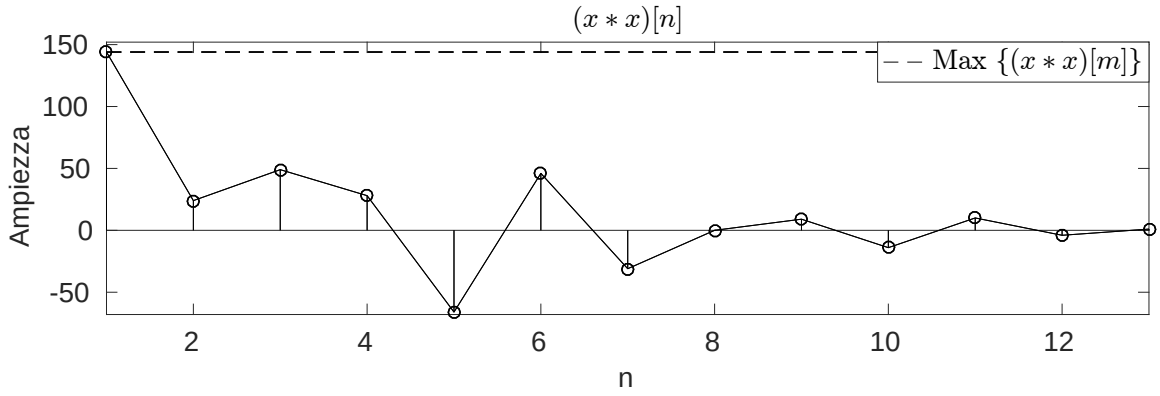


Figura 2.19: *Autoconvoluzione della sequenza*

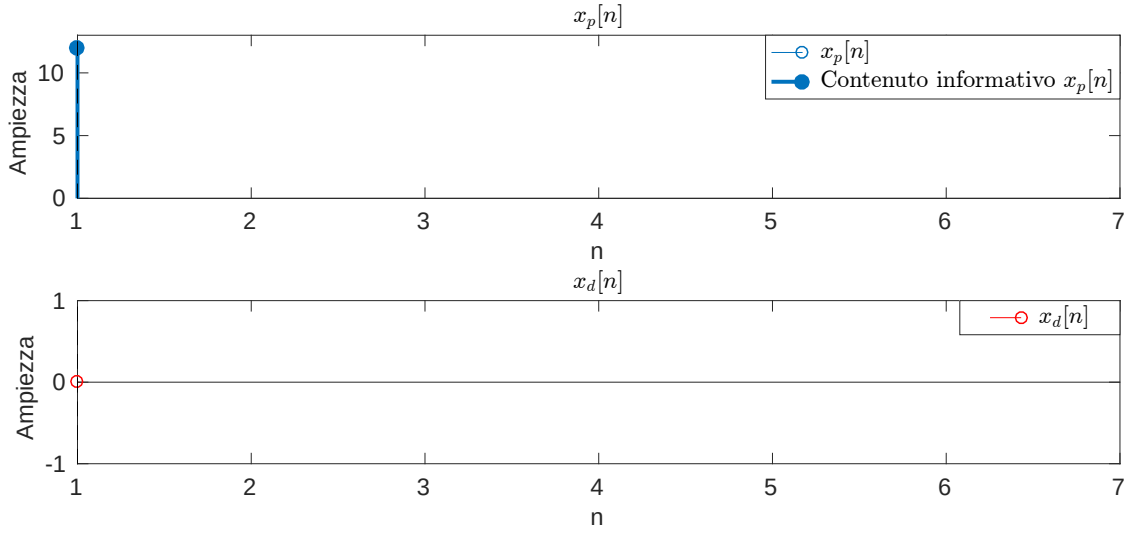


Figura 2.20: *Scomposizione della sequenza x'*

ii. Se n_0 cade in prossimitá del centro della sequenza (alla distanza massima di un campione), x_s sará composta da un solo sample (che diverrá dunque una foglia);

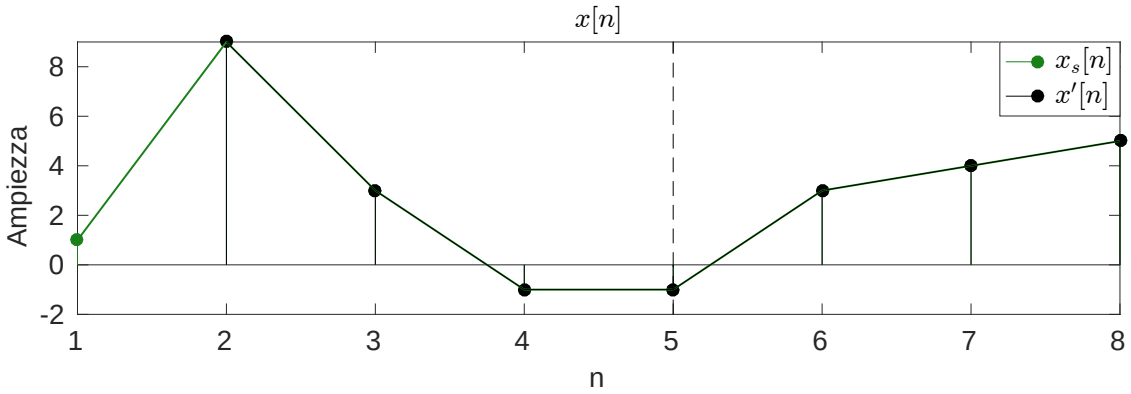


Figura 2.21: *Sequenza con n_0 ad un campione di distanza da $\frac{L+1}{2}$*

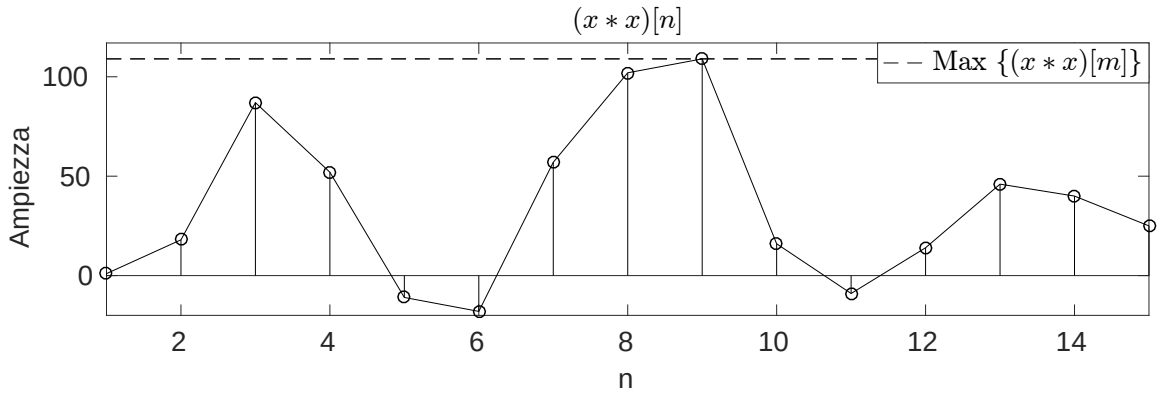


Figura 2.22: *Autoconvoluzione della sequenza*

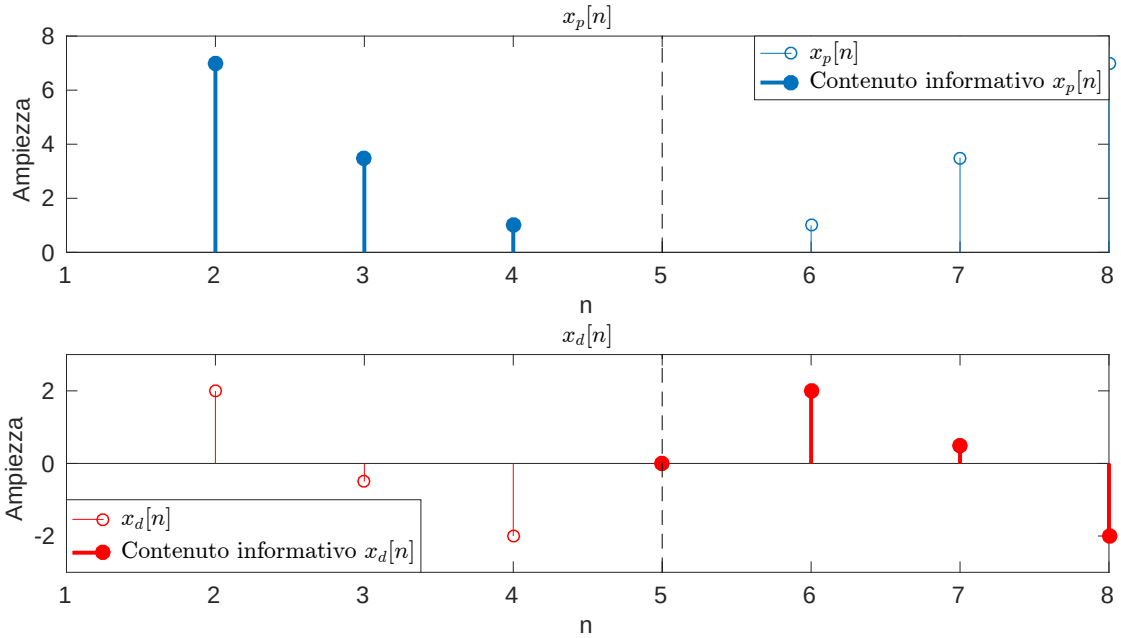


Figura 2.23: *Scomposizione della sequenza $x'[n]$*

Esempio 11 (***Trasformazione di una sequenza mediante iterazione di scomposizioni ottime***)**.**
Data la sequenza a supporto finito, con $L = 9$, $x[n] = \{1,\ 3,\ -2,\ 6,\ 7,\ -9,\ 9,\ 10,\ 2\}$, *applicando iterativamente l'algoritmo di scomposizione si ottiene la trasformata della sequenza:*

$$X[n] = \{16,\ -9 \cdot \sqrt{2},\ -9 \cdot \sqrt{2},\ 0,\ \frac{3}{\sqrt{2}} + 2,\ 2 - \frac{3}{\sqrt{2}},\ 1,\ 3,\ 1\}$$

seguendo lo schema:

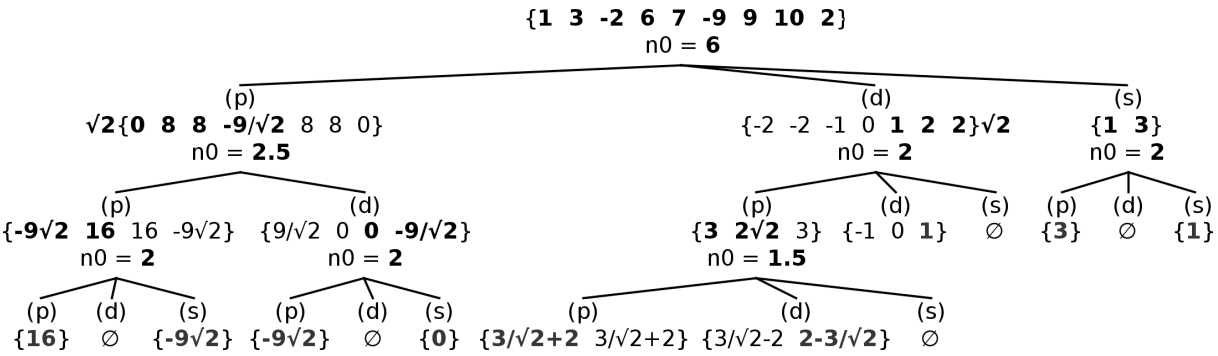


Figura 2.24: *Scomposizione della sequenza* $x[n]$

Come approfondito dalla sezione che segue, e facilmente osservabile dall'esempio sopra presentato, la capacitá di concentrare l'energia in pochi campioni di $X[n]$ della trasformata a scomposizione ottima é di gran lunga superiore rispetto alla trasformata a scomposizione centrale.

Il fine ultimo, si ricorda, rimane la ricostruzione del segnale originale, commettendo il minimo errore possibile, a partire da un numero limitato di campioni del segnale trasformato. Siccome nel caso di trasformazione mediante iterazione di scomposizioni ottime $n_0 \neq \frac{L+1}{2}$, l'algoritmo che permette di realizzare l'anti-trasformata differisce da quello presentato alla fine della **sezione 2.1**, in particolare poiché per ricondursi al segnale di partenza é questa volta necessario conoscere la posizione degli assi di simmetria n_0 di ogni sequenza. Noto questo dato, come presentato nell'esempio che segue, é possibile ricondursi alla sequenza di partenza.

Esempio 12 (***Anti-trasformazione della sequenza dell'esempio 11***)**.**
Azzerando 5 *foglie su* 9*, si ottiene la sequenza ricostruita:*

$$x_r[n] = \{0,\ 3,\ 0,\ 8,\ 8,\ -9,\ 8,\ 8,\ 0\}$$

a fronte di:

$$x[n] = \{1,\ 3,\ -2,\ 6,\ 7,\ -9,\ 9,\ 10,\ 2\}$$

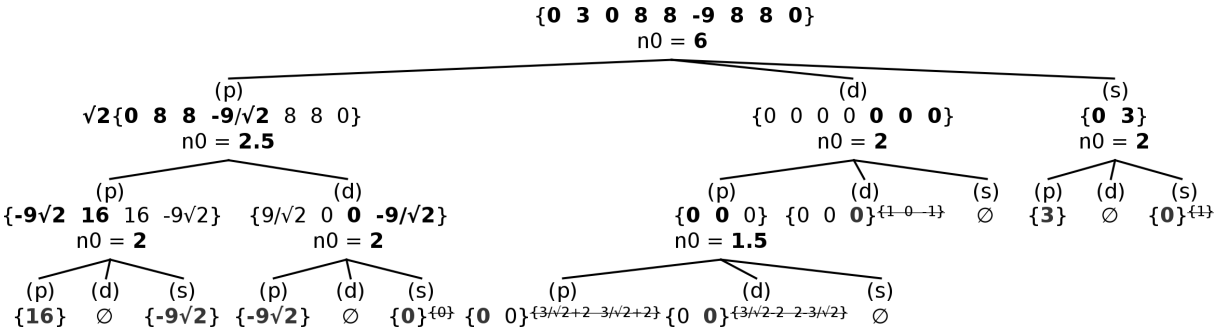


Figura 2.25: *Ricostruzione della sequenza* $x[n]$

*Si noti come, a differenza dell'****esempio 6****, la sequenza presa in considerazione viene ricostruita in maniera discreta a partire dal* 55% *dei campioni pur non essendo "ad-hoc" (i.e. presentando simmetrie particolari), come quelle precedentemente trattate.*

2.3 Metodi a confronto

Di seguito viene analizzato il comportamento di ambedue le trasformate sulla sequenza di lunghezza $L = 8$ casualmente generata:

$$x[n] = \{-3,\ -5,\ -1,\ 5,\ 5,\ -7,\ 9,\ -8\}$$

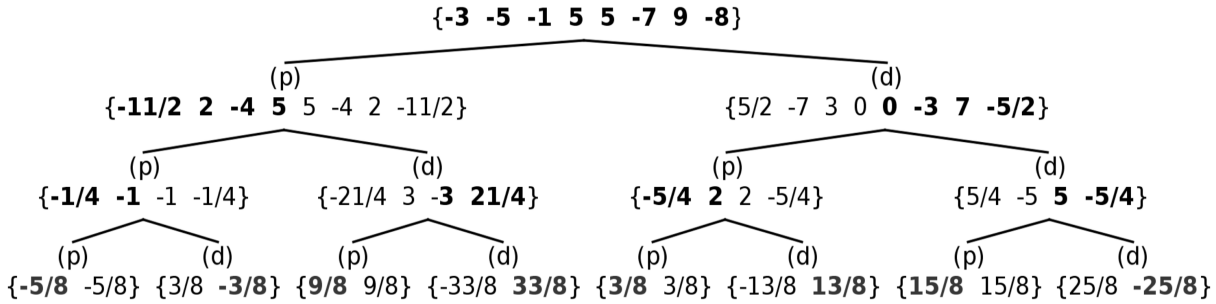


Figura 2.26: *Trasformata della sequenza $x[n]$: scomposizione centrale iterata*

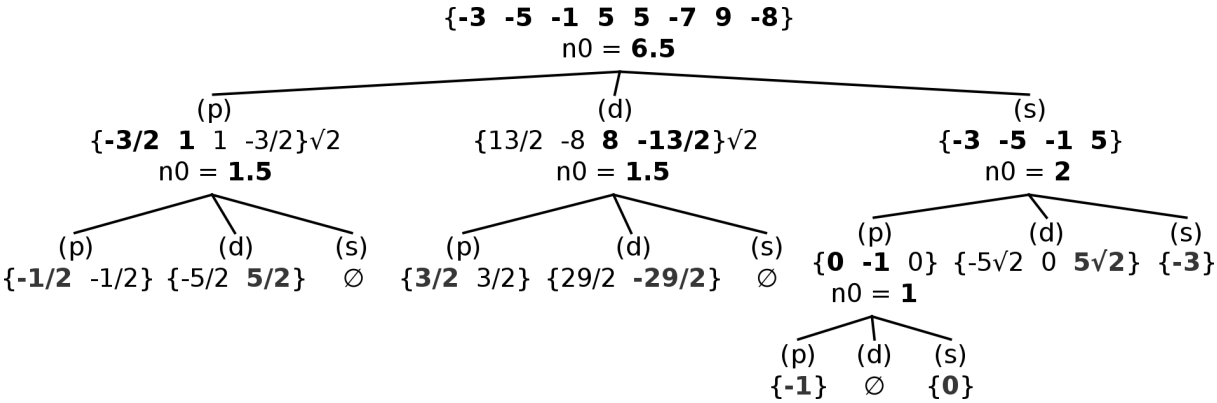


Figura 2.27: *Trasformata della sequenza $x[n]$: scomposizione ottima iterata*

É possibile analizzare l'errore quadratico medio commesso in ricostruzione al variare del numero di foglie poste a zero (i.e. $k < L$):

$$MSE = \frac{1}{8} \cdot \sum_{n=1}^{L=8} (x[n] - x_r[n])^2$$

Come precedentemente esposto, le foglie candidate ad essere azzerate sono quelle alle quali è associata l'energia minore.

Di seguito indicheremo con MSE_c l'errore quadratico medio associato alla trasformata a scomposizione centrale, e con MSE_o quello associato alla trasformata a scomposizione ottima.

Ponendo a zero 2 delle 8 foglie e ricostruendo secondo gli schemi riportati in **figura 3.28** e **figura 3.29**:

$$x_r^c = \{-3,\ -4.25,\ -0.25,\ 3.75,\ 3,\ -7,\ 9,\ -8.75\}$$

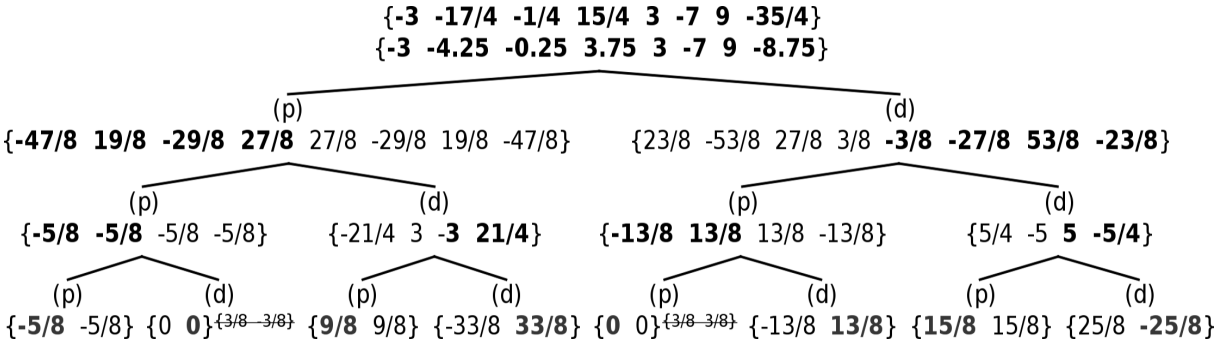


Figura 2.28: *Ricostruzione della sequenza $x[n]$ a partire dal 75% dei campioni: scomposizione centrale iterata*

$$x_r^o = \{-3,\ -5,\ -1,\ 5,\ 4,\ -11/2,\ 21/2,\ -9\}$$

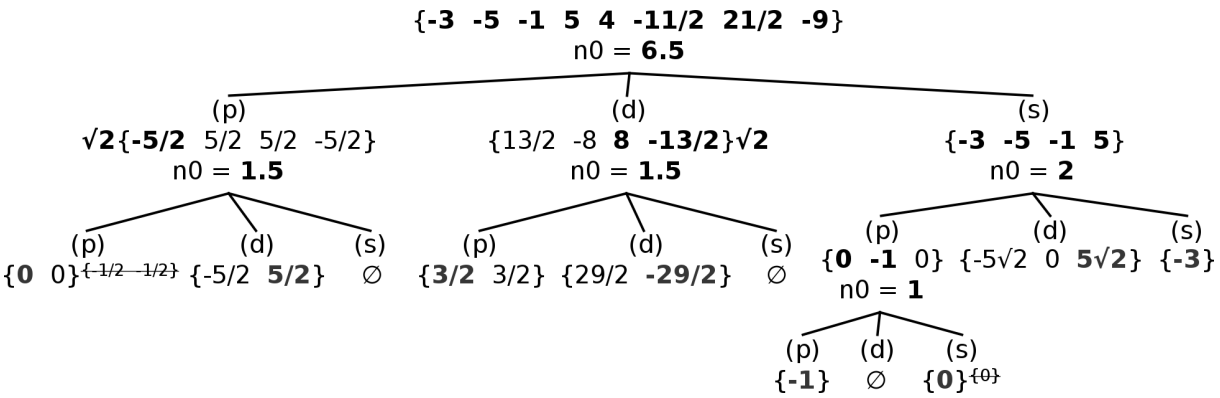


Figura 2.29: *Ricostruzione della sequenza $x[n]$ a partire dal 75% dei campioni: scomposizione ottima iterata*

Si ottengono i seguenti risultati:

$$MSE_c = \frac{1}{8} \cdot \sum_{n=1}^{L=8} (x[n] - x_r^c[n])^2 = \frac{7.25}{8} = 0,90625$$

$$MSE_o = \frac{1}{8} \cdot \sum_{n=1}^{L=8} (x[n] - x_r^o[n])^2 = \frac{4.5}{8} = 0,5625$$

Ponendo a zero 4 delle 8 foglie e ricostruendo secondo gli schemi riportati in **figura 3.30** e **figura 3.31**:

$$x_r^c = \{-1.25,\ -2.5,\ -0.75,\ 4.5,\ 3.75,\ -7.5,\ 10.75,\ -7\}$$

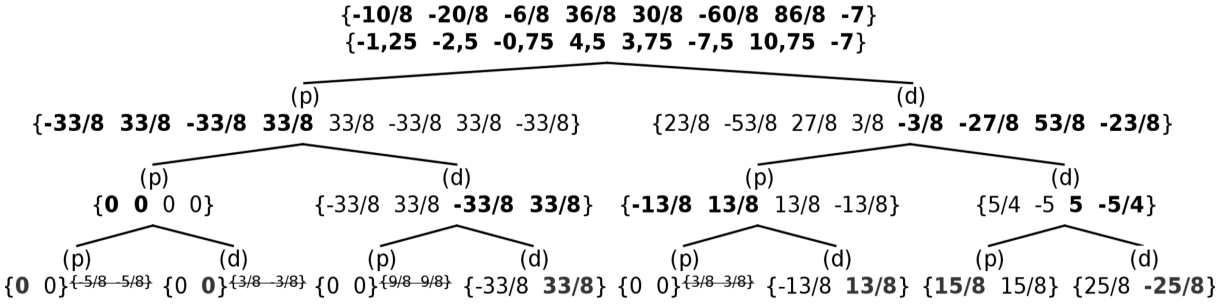


Figura 2.30: *Ricostruzione della sequenza $x[n]$ a partire dal 50% dei campioni: scomposizione centrale iterata*

$$x_r^o = \{-2,\ -5,\ 0,\ 5,\ \frac{19}{4},\ -\frac{19}{4},\ \frac{39}{4},\ -\frac{39}{4}\}$$

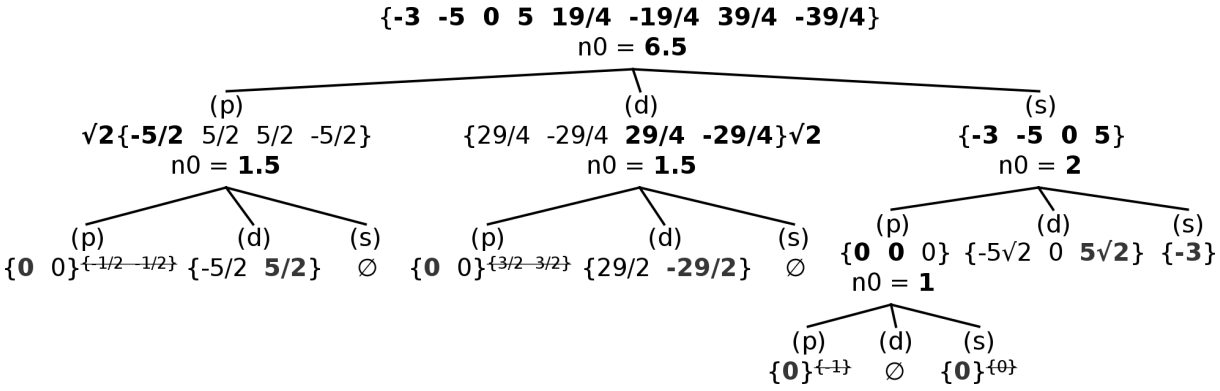


Figura 2.31: *Ricostruzione della sequenza $x[n]$ a partire dal 50% dei campioni: scomposizione ottima iterata*

Si ottengono i seguenti risultati:

$$MSE_c = \frac{1}{8} \cdot \sum_{n=1}^{L=8} (x[n] - x_r^c[n])^2 = \frac{15.5}{8} = 1,9375$$

$$MSE_o = \frac{1}{8} \cdot \sum_{n=1}^{L=8} (x[n] - x_r^o[n])^2 = \frac{9.75}{8} = 1,21875$$

In entrambe le occasioni, come atteso (e per le motivazioni sopra esposte), la trasformata ottenuta dall'iterazione di scomposizioni rispetto all'asse di simmetria individuato dal massimo dell'autoconvoluzione si rivela più performante in termini di errore quadratico medio commesso nella ricostruzione. Inoltre è facile vedere come essa porti ad una più favorevole distribuzione dell'energia sui campioni del segnale trasformato $X[n]$ (i.e. ne concentri gran parte in un numero molto limitato).

Chiamando $X|_k[n]$ la trasformata della sequenza $x[n]$ privata dei k coefficienti ai quali è associata minor energia, ricordando che per far si che quest'ultima venga mantenuta nella trasformazione da $x[n]$ a $X[n]$ è necessario moltiplicare le sequenze per $\sqrt{2}$ ad ogni livello, possiamo valutare la trasformata attraverso un secondo parametro di valutazione:

$$\eta = \frac{E_{X|_k}}{E_X} \cdot 100 = \frac{E_{X|_k}}{E_x} \cdot 100$$

Si noti anche come:

$$\eta = \frac{E_{x_r}}{E_x} \cdot 100$$

Svolgendo i calcoli è possibile affermare che:

- Nel caso in cui $k = 2$,

$$\eta_c = \frac{E^c_{X|_{k=2}}}{E_x} \cdot 100 = \frac{E_{x^c_{r,k=2}}}{E_x} \cdot 100 = \frac{276,75}{279} \cdot 100 = 99,19\%$$

$$\eta_o = \frac{E^o_{X|_{k=2}}}{E_x} \cdot 100 = \frac{E_{x^c_{r,k=2}}}{E_x} \cdot 100 = \frac{278,75}{279} \cdot 100 = 99,91\%$$

- Nel caso in cui $k = 4$,

$$\eta_c = \frac{E^c_{X|_{k=4}}}{E_x} \cdot 100 = \frac{E_{x^c_{r,k=4}}}{E_x} \cdot 100 = \frac{263,5}{279} \cdot 100 = 94,4\%$$

$$\eta_o = \frac{E^o_{X|_{k=4}}}{E_x} \cdot 100 = \frac{E_{x^o_{r,k=4}}}{E_x} \cdot 100 = \frac{275,5}{279} \cdot 100 = 98,745\%$$

- Nel caso in cui $k = 6$,

$$\eta_c = \frac{E^c_{X|_{k=6}}}{E_x} \cdot 100 = \frac{E_{x^c_{r,k=6}}}{E_x} \cdot 100 = \frac{182.5}{279} \cdot 100 = 65,41\%$$

$$\eta_o = \frac{E^o_{X|_{k=6}}}{E_x} \cdot 100 = \frac{E_{x^c_{r,k=6}}}{E_x} \cdot 100 = \frac{260,25}{279} \cdot 100 = 93,80\%$$

L'indice appena presentato η verrà utilizzato nel **capitolo 6** come ulteriore metro di confronto tra le trasformate la nostra implementazione della trasformata basata su simmetrie locali e altre che le affiancheremo.

Capitolo 3

Implementazione in linguaggio C

In questo capitolo presenteremo le parti salienti della nostra implementazione dell'algoritmo esposto nella **sezione 3.2**.
Benché il linguaggio scelto sia, per versatilitá ed efficienza, il C11, eccezion fatta per alcune sezioni (i.g. strutture dati, funzioni particolari ...) in seguito prediligeremo l'utilizzo di pseudo-codice per garantire una maggiore leggibilitá ed aprire alla possibile riscrittura in linguaggi differenti.

Si noti come, per esigenze implementative, vi sono differenze non trascurabili tra l'implementazione C e l'algoritmo di trasformazione basato su scomposizione ottima precedentemente presentato. A beneficio del lettore, elenchiamo le principali:

1. ***Differenza tra le parti informative mantenute di ogni sequenza.***
   Nella sezione precedente abbiamo illustrato, sia con immagini di sequenze che con modelli ad albero, come sia possibile ricostruire una sequenza $x[n]$ mantenendo solo parte dei campioni delle sequenze $x_p[n]$ e $x_d[n]$: in particolare, precedentemente sono state selezionati i primi $\lfloor \frac{L+1}{2} \rfloor$ campioni di x_p e gli ultimi $\lfloor \frac{L}{2} \rfloor$ di x_d.
   Nel codice C, essendo le sequenze rappresentate da vettori che, canonicamente hanno indice del primo elemento uguale a 0 e indice dell'ultimo elemento uguale a $L-1$, per semplicitá vengono mantenuti **i primi $\lfloor \frac{L+1}{2} \rfloor$ della parte pari** e **i primi $\lfloor \frac{L}{2} \rfloor$ della parte dispari**;

2. ***Scelta del candidato ottimo per la scomposizione n_0.***
   Nel caso in cui vi siano due valori di massimo uguali in modulo nell'autoconvoluzione della sequenza, l'asse di simmetria non viene scelto in base ad una effettiva convenienza sulla scomposizione finale (i.e. mediante trasformazione e successivo *backtracking*) ma semplicemente in base all'ordine (il primo n_0 é il candidato prescelto).

3.1 Definizione macro e formati file di input e output

Data l'esigenza di riscalare per $\sqrt{2}$ e il suo inverso (rispettivamente durante il calcolo della trasformata e dell'antitrasformata), in testa al codice sono presenti le direttive al pre-processore (*macro*):

```
#define SQRT2     1.41421356237309;
#define SQRT2inv  0.70710678118655;
```

Questo permette di eliminare i tempi necessari al calcolo della radice di due (presenti nel caso essa venisse calcolata grazie all'ausilio di una libreria, per esempio `math.c` o `cmath`).

Per quanto riguarda il processo di trasformazione lo script **accetta in ingresso file di testo** formattati nel seguente modo (spazi e *newline* `"\n"` inclusi):

```
n_righe n_colonne
sig[1, 1] sig[1, 2] ... sig[1, n_colonne]
sig[2, 1] sig[2, 2] ... sig[2, n_colonne]
...
sig[n_righe, 1] sig[n_righe, 2] ... sig[n_righe, n_colonne]
```

dove i valori dei vari campioni sono variabili di tipo `float` (4 byte), e **produce file binari** contenenti:

```
n_righe n_colonne
leaves[1, 1] leaves[1, 2] ... leaves[1, n_colonne]
length(vettore_n0_riga_1) vettore_n0_riga_1
leaves[2, 1] leaves[2, 2] ... leaves[2, n_colonne]
length(vettore_n0_riga_2) vettore_n0_riga_2
...
leaves[n_righe, 1] leaves[n_righe, 2] ... leaves[n_righe, n_colonne]
length(vettore_n0_riga_n_righe) vettore_n0_riga_n_righe
```

dove `n_righe`, `n_colonne` e `length(vettore_n0_)` sono interi da 4 byte, mentre ogni valore `leaves[ , ]` è un float da 4 byte. Si noti come, questa volta, essendo il file binario, ogni valore (intero o a virgola mobile che sia) non è spaziato nè dal precedente nè dal successivo: dovremo quindi premurarci di leggere blocchi di dimensione corretta.

Per quanto riguarda il processo di ricostruzione lo script **accetta in ingresso file binari** formattati come sopra, e **produce a sua volta file binari** contenenti questa volta, ovviamente, il segnale ricostruito a partire dalle foglie e dai vettori degli assi per le scomposizioni ottime.

3.2 Trasformata

Come precedentemente introdotto, la trasformazione avviene grazie alla costruzione di un albero ternario. Viene dunque introdotto un nuovo tipo, ovvero la struttura dati utilizzata per la sua rappresentazione:

```
struct T{
    struct T *p     // nodo sequenza pari
    struct T *d     // nodo sequenza dispari
    struct T *s     // nodo sequenza spaiata
    float *corr     // sequenza di livello
    float n0;       // asse ottimo n0
    int len;        // lunghezza della sequenza di livello
}TREE;
```

Ogni nodo dell'albero contiene:

- Un puntatore al nodo nel quale sarà contenuta la parte informativa della sequenza dispari ricavata dalla sequenza in questione;
- Un puntatore al nodo nel quale sarà contenuta la parte informativa della sequenza dispari ricavata dalla sequenza in questione;
- Un puntatore al nodo nel quale sarà contenuta la parte informativa della sequenza spaiata ricavata dalla sequenza in questione;
- Un puntatore alla prima cella del vettore di `float` contenente la sequenza di livello (necessario all'allocazione dinamica);
- La posizione dell'asse n_0 che definisce come la sequenza corrente debba essere divisa.
- La lunghezza della sequenza corrente;

Per quanto riguarda la creazione dell'albero ternario (i.e. calcolo del vettore delle foglie, e quindi l'effettiva trasformazione) abbiamo prediletto l'approccio ricorsivo e l'allocazione dinamica di strutture all'approccio iterativo e all'allocazione statica.
Questo ci garantisce una maggiore libertá sulla gestione della memoria, che porta:

- La possibilitá di poter passare per riferimento tutte le strutture dati utilizzate alle funzioni di cui si compone il codice, garantendo buone prestazioni in tal senso e un uso tanto efficiente quanto limitato della memoria;
- La possibilitá di de-allocare istantaneamente grosse quantitá di memoria nel momento in cui una struttura dati assolvesse il proprio compito e/o diventasse obsoleta.

Ma chiaramente impone, di contro, il dover adoperare una maggiore attenzione, poichè in questo modo:

- Le strutture dati sono univoche in tutto il flusso d'esecuzione (i.e. tutte le funzioni possono modificarle);
- La memoria allocata per una struttura dati rimane tale fino a che non viene specificato altrimenti.

3.2.1 Autoconvoluzione: metodo diretto e indiretto (GSL)

Come presentato nella **sezione 3.2**, la prima operazione da compiere per scomporre una sequenza é la ricerca dell'asse di simmetria "ottimo", che operativamente avviene mediante l'individuazione dell'indice del massimo in modulo dell'autoconvoluzione della sequenza in questione. Esistono due metodi per computare l'autoconvoluzione, ossia il **metodo diretto** e il **metodo indiretto**.

Il **metodo diretto** applica la classica definizione di convoluzione tra sequenze:

$$z[n] = x[n] * y[n] = \sum_{n=-\infty}^{\infty} x[m] \cdot y[n-m]$$

di semplice implementazione.
In pseudo-codice, la ricerca dell'indice del massimo puó essere scritta come:

```
int indice_max_autoconv(float x[]){
    temp = 0;
    max = 0;
    L = lenght(x);
    L_conv = 2 * L - 1;
    for(i = 0, i < L_conv, i++){
        for (j = 0, k = i; j < L ; j++, k--)
            if(k >= 0 && k < L)
                temp += (x[k] * x[j]);

        if(abs(temp) > max){
            max = abs(temp);
            indice_max = i;
        }
    }
    n0 = indice_max/2;
    return n0;
}
```

Dove:

- La variabile `k` rappresenta il supporto comune (in campioni) dei due segnali (che scorrono l'uno, ribaltato, entro l'altro;
- La condizione `if(k >= 0 && k < L)` fa si che l'operazione di moltiplicazione venga eseguita solamente dove entrambe le sequenze sono definite;

La funzione sopra esposta é tanto semplice da implementare e da comprendere, quanto inefficiente dal punto di vista della complessitá computazionale, che risulta essere $O((2 \cdot L - 1) \cdot L)$. Questo la rende poco adatta all'utilizzo con sequenze di numerosi campioni, quali quelle monodimensionali che andremo a trattare successivamente.

Il **metodo indiretto**, invece, sfrutta la **trasformata discreta di Fourier** (DFT), definita, per una sequenza $x[n]$ a supporto finito L, come:

$$DFT\{x[n]\} = X[k] = \sum_{n=0}^{L-1} x[n] \cdot e^{-i2\pi n \frac{k}{L}} \qquad k = 0, 1, \ldots, L-1$$

Si noti come le sequenze a supporto finito L, in questa istanza, siano definite per $n = 0, 1, \ldots, L-1$. Controparte computabile della canonica trasformata di Fourier, la DFT gode di invertibilitá:

$$DFT^{-1}\{X[k]\} = x[n] = \frac{1}{L} \sum_{n=0}^{L-1} x[n] \cdot e^{i2\pi n \frac{k}{L}} \qquad n = 0, 1, \ldots, L-1$$

ed é caratterizzata da una periodicitá, derivata dai legami che questi ha con la trasformata di Fourier a tempo discreto ($DTFT$).
Attraverso l'operatore **modulo L**:

$$(n-m)_L = mod_L\{n-m\}$$

é possibile definire l'operazione di **convoluzione circolare** (su una finestra di L samples):

$$z_c[n] = x[n] \circledast y[n] = \sum_{n=0}^{L-1} x[m] \cdot (y[n-m])_L$$

dalla quale é possibile affermare che:

$$x[n] \circledast y[n] = DFT^{-1}\{X[k] \cdot Y[k]\}$$

da cui l'aggettivo "indiretta".
Il procedimento descritto vanta una maggiore efficienza in determinati casi, tipicamente per sequenze molto lunghe aventi numero di campioni pari, o leggermente inferiore, ad una potenza di 2: implementando entrambe le convoluzioni e lasciando la scelta ad un semplice controllo sulla lunghezza L della sequenza, normalmente si è in grado di operare secondo il metodo meno dispendioso in termini di tempo.

Come risulterá chiaro a questo punto, mentre la convoluzione diretta si rivela di facile implementazione, al contrario il metodo indiretto necessita di una finezza implementativa non banale. Per queste ragioni abbiamo deciso di utilizzare, nella nostra implementazione, le ***GNU Scientific Library***, una collezione di funzioni per linguaggio C e C++ contenente delle FFT routines allo stato dell'arte realizzate secondo il classico **algoritmo di Cooley-Tukey**. L'idea alla base dell'algoritmo (che degenera poi in diverse varianti) é quella di applicare il paradigma *divide and conquer*: la trasformata non viene calcolata direttamente sulla sequenza di lunghezza L, ma bensí su sequenze di lunghezza L_1 ed L_2 tali per cui $L = L_1 \cdot L_2$, ottenute dalla decimazione della prima.

Alcuni esempi sono:

- ***Radix-2 FFT routine***, che divide iterativamente la sequenza in due nuove sottosequenze ottenute isolando gli indici pari da quelli dispari, per poi procedere al calcolo della FFT. Risulta, per ovvi motivi, ottima per sequenze di lunghezza $L = 2^k$, nei quali la complessitá computazionale media si rivela essere $O(\frac{L}{2} \cdot log_2(L)$;
- ***Radix-3 FFT routine***, implementanti una logica molto simile ai Radix-2 (decimazione e poi calcolo), ottimizzati per sequenze di lunghezza pari a 3^k, riducono ulteriormente il numero di operazioni,

. . . e via dicendo.

Non essendo la lunghezza della sequenza da trasformare fissa, la nostra implementazione utilizza la routine *GSL* ***Mixed-Radix***, che sfrutta diverse routine (tra le quali quelle sopra esposte, ma anche *Radix-4, Radix-5 . . .*) insieme al fatto che una volta divisa la sequenza di partenza (di lunghezza L) in due sottosequenze (di lunghezza L_1 ed L_2 tali per cui $L = L_1 \cdot L_2$), la trasformazione possa essere continuata mediante l'applicazione di una qualsiasi delle altre routine (dipendentemente da L_1 e L_1).
Questo procedimento si rivela di buona efficienza e alta versatilitá, poiché non obbliga all'inserimento di eventuale padding per ricondurre il problema ad uno dei casi sopra esposti. Di contro, peró, siccome **procede fattorizzando** la lunghezza L della sequenza in ingresso:

- Necessita, in termini di memoria, di spazio aggiuntivo nel quale allocare delle strutture contenenti informazioni sulla fattorizzazione (riutilizzabili per sequenze di egual lunghezza, necessarie alla ricostruzione);
- Nel caso in cui L risulti fattorizzabile nel prodotto di primi non forniti di implementazione *Radix-n*, viene utilizzato un approccio diretto, di complessitá computazionale $O(n^2)$. Questo implica che, per casi particolarmente sfortunati (i.e. L fattorizzabile in prodotto di primi molto grandi), la convoluzione indiretta cosí implementata degeneri in quella diretta.

Di seguito presentiamo, in pseudo-codice, un'implementazione della convoluzione indiretta utilizzando le librerie GSL, lasciando i commenti ai capoversi successivi:

```
indice_max_autoconv(float x[])
{
    L_autoconv = 2 * length(x) - 1;

    // Inizializzazione del vettore a 0
    float data = new array(2 * L_autoconv);

    // Inizializzazione variabili ausiliarie di cui sopra
    gsl_fft_complex_wavetable wavetable;
    gsl_fft_complex_workspace workspace;

    for(i = 0; i < L_autoconv; ++i)
        data[2 * i] = x[i];

    wavetable = gsl_fft_complex_wavetable_alloc(2 * L_autoconv);
    workspace = gsl_fft_complex_workspace_alloc(2 * L_autoconv);

    gsl_fft_complex_forward(data, wavetable, workspace);

    autoconv_fft = gsl_vector_multiply(data, data);

    autoconv = gsl_fft_complex_backward(autoconv_fft, 1, ...
        2 * L_autoconv, wavetable, workspace);

    indice_max_autoconv =
        gsl_get_max_index(gsl_abs_vector(autoconv)));

    indice_max = indice_max_autoconv/2;
    n0 = indice_max/2;

    return n0;
}
```

In primis, alcune delle operazioni presenti nel codice sono associate al metodo mediante il quale le librerie gestiscono vettori complessi: per il vettore autoconvoluzione di $2L-1$ elementi viene allocata il doppio della memoria:

```
length(data) = 2 * L_autoconv;
```

poichè GSL considera come parte reale del vettore gli indici pari di `data`, e come immaginaria gli indici dispari.
Per questo motivo l'inizializzazione viene compiuta con:

```
for(i = 0; i < L_autoconv; ++i)
        data[2 * i] = x[i];
```

Detto ciò, come prima accennato, è di primaria importanza l'allocazione della struttura dati corretta per rendere possibile trasformazione e antitrasformazione di sequenze di arbitraria lunghezza. La *routine FFT* sopra utilizzata è implementata a partire da un algoritmo di tipo ***Mixed Radix***, per il quale *GSL* fornisce due strutture dati specifiche:

1. `wavetable`: tabella contenente la fattorizzazione in numeri primi della lunghezza originale, i twiddle factors e una tabella trigonometrica preallocata;

2. `workspace`: working space addizionale atto a contenere i passaggi intermedi della FFT.

Il calcolo della FFT della sequenza in ingresso viene a questo punto eseguito grazie a:

```
gsl_fft_complex_forward(data, wavetable, workspace);
```

Successivamente, si effettua la moltiplicazione nel dominio delle frequenze:

```
autoconv_fft = gsl_vector_multiply(data, data);
```

ed infine si antitrasforma il vettore `data`:

```
autoconv = gsl_fft_complex_backward(autoconv_fft, 1, ...
    2 * L_autoconv, wavetable, workspace);
```

A partire dalle basi teoriche precedentemente esposte, è chiaro come l'antitrasformata così ottenuta corrisponda all'autoconvoluzione della sequenza originale.

Infine, consci di come le librerie gestiscano i vettori complessi, una volta calcolato l'indice corrispondente al massimo dell'autoconvoluzione è necessario compiere una divisione per due, per mitigare la lunghezza doppia del vettore `autoconv`.
Questa operazione è corretta solamente perchè nel nostro caso, essendo il segnale di partenza reale, il massimo risiederà sempre e solo nella parte reale del vettore autoconvoluzione (i.e. solo gli indici pari del vettore autoconv conterranno valori diversi da zero).
In questo modo si ottiene `indice_max`, ovvero $2 \cdot n_0$, che ulteriormente diviso per 2 produce l'asse di simmetria che porta alla scomposizione ottima della sequenza finita.

3.2.2 Costruzione dell'albero ternario

In pseudo-codice, l'implementazione della costruzione dell'albero ternario:

```
createTree(TREE t) : (float leaves[], float n0s[])
{
    if(length(t.corr) == 1)
    {
        leaves.add(t.corr[0]);
        return;
    }

    t.n0 = indice_max_autoconv(t.corr);
    n0s.add(t.n0);
    nsamples = min(t.len - 1 - floor(t.n0), ceil(t.n0));
    start    = ceil(t.n0) - nsamples;
    end      = floor(t.n0) + nsamples;

    [t.e, t.o, t.u] = getSplitTree(t.corr, start, end);

    createTree(t.p);
    delete(t.p);
    createTree(t.d);
    delete(t.d);
    createTree(t.s);
    delete(t.s);
}
```

Il parametro di ingresso alla funzione "`createTree`" è il primo nodo dell'albero ternario contenente la sequenza iniziale $x[n]$ che dovrà essere trasformata.

I due parametri di uscita sono:

1. `leaves`, ovvero le foglie dell'albero costruito (i.e. segnale trasformato, $X[n]$);
2. `n0s`, vettore relativo agli n0 di ogni singola sequenza scomposta, partendo dal livello più alto e scendendo progressivamente.

Il primo passo da effettuare è il controllo sulla lunghezza della sequenza corrente (piede della ricorsione); se tale valore risulta uguale a 1 allora la sequenza corrente conterrà una foglia della sequenza trasformata e andrà memorizzata di conseguenza nel vettore "`leaves`" ritornato. Se la sequenza corrente presenta invece una lunghezza > 1 si calcola la sua autoconvoluzione individuando l'asse n_0 per la suddivisione ottima della sequenza in parte pari, dispari e spaiata a seconda della posizione del suddetto asse ed infine lo si aggiunge al vettore "`n0s`" ritornato.

La funzione di supporto getSplitTree è definita come:

```
getSplitTree(float x[], int start, int end) : (TREE tr[3])
{
    win       = end - start + 1;
    on_sample = is_odd(win);
    for(i = 0; i < win/2 ; ++i)
    {
        tr[0].corr.add((x[start + i] + x[end - i])/2 * SQRT2);
        tr[1].corr.add((x[start + i] - x[end - i])/2 * SQRT2);
    }
    if(on_sample) tr[0].corr.add(x[start + i]);

    for(i = 0; i < start; ++i)
        tr[2].corr.add(x.v[i]);
    for(i = 0; i < t.len - end - 1 ; ++i)
        tr[2].corr.add(x[end + i + 1]);
}
```

Con un semplice calcolo viene identificata la "finestra" di campioni da considerare nel calcolo della parte pari e dispari della sequenza (minima distanza tra l'inizio/fine della sequenza e l'asse n_0).
Se la finestra considerata contiene un numero dispari di campioni significa che l'asse n_0 cade su un campione e che tale dovrà essere copiato nella parte pari della sequenza (non effettuando la riscalatura per $\sqrt{2}$).
Infine si richiama ricorsivamente la funzione "`createTree`" sui nodi sottostanti (pari, dispari, spaiato).
Si noti come, alla fine di ogni ricorsione, le parti pari, dispari e spaiate vengano eliminate per non appesantire la memoria a run-time, mantenendo per tutta la durata dell'esecuzione del codice solo le informazioni (strutture dati $\Rightarrow$ nodi dell'albero) necessarie all'identificazione della sequenza trasformata e al vettore degli n_0.
Viene di seguito riportato un esempio esplicativo che fa utilizzo di una delle sequenze precedentemente utilizzate (**Sezione 3.3**).

Esempio 13 (***Comportamento della funzione createTree***)**.**

Sequenza iniziale: `t.corr` $= \{-3,\ -5,\ -1,\ 5,\ 5,\ -7,\ 9,\ -8\}$

Paramentro di ingresso: `t` *(TREE)*

La chiamata alla funzione `createTree` *produce in uscita:*

Sequenza "`leaves`*":* $\{-\frac{1}{2},\ \frac{5}{2},\ -\frac{3}{2},\ \frac{29}{2},\ -1,\ -0,\ -5\sqrt{2},\ -3\}$

Sequenza "`n0s`*":* $\{\frac{11}{2},\ \frac{1}{2},\ \frac{1}{2},\ 2,\ 1\}$

3.3 Antitrasformata

In pseudo-codice, l'implementazione della ricostruzione della sequenza originale a partire dal vettore "leaves" e "n0s":

```
reconstruct(TREE t, float leaves[], float n0s[]) : void
{
    if(length(t.corr) == 1)
    {
        t.corr[0] = leaves.getNext();
        return;
    }
    t.n0 = n0s.getNext();
    nsamples  = min(t.len - 1 - floor(t.n0), ceil(t.n0));
    start     = ceil(t.n0) - nsamples;
    end       = floor(t.n0) + nsamples;
    win       = end - start + 1;
    on_sample = is_odd(win);

    dim_p = nsamples + on_sample;
    dim_d = nsamples;
    dim_s = t.len - win;

    t.p = new TREE;
    t.p.len = dim_p;

    t.d = new TREE;
    t.d.len = dim_d;

    t.s = new TREE;
    t.s.len = dim_s;

    reconstruct(t.p, leaves, n0s);
    reconstruct(t.d, leaves, n0s);
    reconstruct(t.s, leaves, n0s);

    for(i = 0; i < win/2 ; ++i)
    {
        t.corr[start + i] =
            (t.p.corr[i] + (dim_d ? t.d.corr[i] : 0)) * SQRT2inv;
        t.corr[end - i] =
            (t.p.corr[i] - (dim_d ? t.d.corr[i] : 0)) * SQRT2inv;
    }

    if(on_sample) t.corr[start + i] = t.p.corr[i];
```

```
        for(i = 0; i < start; ++i)
            t.corr[i] = t.s.corr[i];
        for(i = 0; i < t.len - end - 1 ; ++i)
            t.corr[end + i + 1] = t.s.corr[i];

        delete(t.d);
        delete(t.p);
        delete(t.s);
    }
```

La funzione `reconstruct` accetta come parametri di ingresso:

1. `t`, variabile di tipo `TREE`, nodo di partenza nel quale risulta inizializzato solamente l'attributo `len` (lunghezza sequenza delle foglie ⇒ base della ricostruzione);

2. `leaves`, sequenza delle foglie (segnale trasformato);

3. `n0s`, vettore contenente le posizioni dei vari assi di simmetria.

e non ritorna nessun valore: la sequenza ricostruita sarà contenuta nel nodo iniziale passato come parametro alla funzione.
È possibile sintetizzare il comportamento della funzione evidenziando i passi principali da essa eseguiti:

1. Come prima cosa, noto il fatto che la lunghezza di $x[n]$ è eguale a quella di $X[n]$, sfruttando il primo elemento del vettore `"n0s"` la funzione è in grado di stabilire quali siano le dimensioni delle sottosequenze (pari, dispari, spaiata) di livello direttamente inferiore al nodo radice.

2. Dopodichè, dando precedenza al nodo contenente la sequenza relativa alla parte pari della scomposizione (sempre presente), seguita da quello relativo alla parte dispari per terminare poi con quello relativo alla parte spaiata, viene iterato il procedimento grazie alla ricorsione.

3. Dopo un certo numero di operazioni, come evidenziato, ad esempio, dal terzo passo della ricorsione di **esempio 14**, si giunge ad avere una sequenza di lunghezza unitaria. Grazie al piede della ricorsione, il vettore del nodo corrente `"corr[0]"` è inizializzato con uno degli elementi del vettore `leaves`.
   L'utilizzo di `getNext()` permette di scorrere il vettore delle foglie (i.e. assegnare in maniera univoca ogni elemento);

4. Dopo aver compiuto l'assegnazione di due elementi del vettore delle foglie ad altrettanti vettori di nodi (siano essi relativi a parti pari, dispari o spaiate), il `return` fa si che si arrivi all'effettiva ricostruzione di un nodo di livello appena superiore all'ultimo;

5. Il procedimento viene iterato grazie alla ricorsione e l'albero viene ricostruito a partire dai rami pari, o "di sinistra" nelle rappresentazioni grafiche dei capitoli precedenti.

Esempio 14 (***Ricostruzione dell'albero a partire dal vettore di n_0 e delle foglie***)**.**

Facendo riferimento alla sequenza $x[n]$ di ***sezione 3.3****, ripresa nell'****esempio 13****:*

```
leaves = [-1/2, -5/2, -3/2, 29/2, -1, 0, -5*SQRT2, -3],;
length(leaves) = 8,;
n0s = [11/2, 1/2, 1/2, 2, 1];
```

Si parte quindi con la creazione di una nuova variabile `TREE t` *della quale verrà inizializzato solamente il parametro lunghezza:* `t.len = length(leaves) = 8`.
Come sopra illustrato, per prima cosa avviene l'estrazione della lunghezza delle sequenze di livello successivo al nodo radice (sfruttando la posizione nota dell'asse di simmetria):

Prima chiamata

```
n0 = 11/2,
len = 8,
nsamples = min(8 - 1 - floor(5.5), ceil(5.5)) = 2,
start = 6 - 2 = 4,
end = 5 + 2 = 7,
win = 7 - 4 + 1 = 4,
on_sample = 0,
dim_p = 2,  dim_d = 2,  dim_s = 8 - 4 = 4.
```

Note dunque le dimensioni delle sequenze nei tre rami, si procede con l'allocazione della memoria e l'inizializzazione delle lunghezze:

```
t.p = new TREE;
t.p.len = dim_p = 2;

t.d = new TREE;
t.d.len = dim_d = 2;

t.s = new TREE;
t.s.len = dim_s = 4;
```

per poi ricorrere allo stesso modo su ognuna di esse.

Seconda Chiamata (t.p)

```
n0 = 1/2,
len = 2,
nsamples = min(2 - 1 - floor(0.5), ceil(0.5)) = 1,
start = 1 - 1 = 0,
end = 0 + 1 = 1,
win = 1 - 0 + 1 = 2,
on_sample = 0,
dim_p = 1,  dim_d = 1,  dim_s = 2 - 2 = 0.
```

Terza Chiamata (t.p.p)

```
len = 1,
corr[0] = leaves[0] = -1/2.
```

Quarta Chiamata (t.p.d)

```
len = 1,
corr[0] = leaves[1] = -5/2.
```

A questo punto, si hanno tutti gli elementi necessari alla ricostruzione di un primo nodo dell'albero. Dopo aver eseguito le due chiamate sopra riportate il flusso d'esecuzione rientra, trovandosi prima del ciclo `for` *responsabile della ricostruzione.*
Alla funzione sono note:

$$x_p = \{-\frac{1}{2}\}, \quad x_d = \{\frac{5}{2}\}, \quad x_s = \emptyset$$

Trovandoci, dopo i `return`*, al nodo "*`t.p`*", le cui proprietà sono riportate sotto alla voce "seconda chiamata":*

```
t.p.corr[0] = (-1/2 - 5/2) * SQRTinv = -3/SQRT2 = -3*SQRT2/2,
t.p.corr[1] = (-1/2 + 5/2) * SQRTinv =  2/SQRT2 =  SQRT2.
```

Dopo la ricostruzione del nodo "`t.p`*", la funzione esegue le operazioni di cui sopra con il fine di determinare tutti i nodi aventi come padre il nodo "*`t.d`*":*

Quinta Chiamata (t.d)

```
n0 = 1/2,
len = 2,
nsamples = min(2 - 1 - floor(0.5), ceil(0.5)) = 1,
start = 1 - 1 = 0,
end = 0 + 1 = 1,
win = 1 - 0 + 1 = 2,
on_sample = 0,
dim_p = 1,  dim_d = 1,  dim_s = 2 - 2 = 0.
```

Sesta Chiamata (t.d.p)

```
len = 1,
corr[0] = leaves[2] = -3/2.
```

Settima Chiamata (t.d.d)

```
len = 1,
corr[0] = leaves[3] = 29/2.
```

Similmente a prima viene ricostruita la sequenza del nodo appena superiore a quello contenente le foglie:

$$x_p = \{-\frac{3}{2}\}, \quad x_d = \{\frac{29}{2}\}, \quad x_s = \emptyset$$

```
t.d.corr[0] = (-3/2 + 29/2) * SQRTinv =  13/SQRT2 = 13*SQRT2/2,
t.d.corr[1] = (-3/2 - 29/2) * SQRTinv = -16/SQRT2 = -8*SQRT2.
```

A questo punto, disponendo sia di `t.p.corr` *che di* `t.d.corr` *è necessario ricostruire la parte di albero relativa a* `t.s` *per poter arrivare alla ricostruzione di quella necessaria:*

Ottava Chiamata (t.s)

```
n0 = 2,
len = 4,
nsamples = min(4 - 1 - floor(2), ceil(2)) = 1,
start = 2 - 1 = 1,
end = 2 + 1 = 3,
win = 3 - 1 + 1 = 3,
on_sample = 1,
dim_p = 2,  dim_d = 1,  dim_s = 4 - 3 = 1.
```

Nona Chiamata (t.s.p)

```
n0 = 1,
len = 2,
nsamples = min(2 - 1 - floor(1), ceil(1)) = 0,
start = 1 - 0 = 1,
end = 1 + 0 = 1,
win = 1 - 1 + 1 = 1,
on_sample = 1,
dim_p = 1,  dim_d = 0,  dim_s = 2 - 1 = 1.
```

Decima Chiamata (t.s.p.p)

```
len = 1,
corr[0] = leaves[4] = -1.
```

Undicesima Chiamata (t.s.p.s)

```
len = 1,
corr[0] = leaves[5] = 0.
```

Poichè, stavolta, `length(t.d.p.d.corr) = 0` *(i.e. non è presente la parte dispari), per come è stato concepito l'algoritmo (e conseguentemente steso il codice) è possibile affermare senza indugi che la posizione della parte pari sarà quella nella quale cade l'asse* `t.s.p.n0`*, e che essa non dovrà essere riscalata per* $\sqrt{2}$*.*

In ordine di esecuzione, le operazioni svolte saranno dunque:

```
t.s.p.corr[1] = -1,
t.s.p.corr[0] =  0.
```

Dodicesima Chiamata (t.s.d)

```
len = 1,
t.s.d.corr[0] = leaves[6] = -5*SQRT2.
```

Tredicesima Chiamata (t.s.s)

```
len = 1,
t.s.s.corr[0] = leaves[7] = -3.
```

Da qua in poi, disponendo della sua parte pari, dispari e spaiata, è possibile ricostruire `t.s`*. In ordine di esecuzione, le operazioni svolte:*

```
t.s.corr[1] = (0 - 5*SQRT2) * SQRT2inv = -5,
t.s.corr[2] = -1,
t.s.corr[3] = (0 + 5*SQRT2) * SQRT2inv = +5,
t.s.corr[0] =  -3.
```

Finalmente, disponendo di:

$$x_p = \{-\frac{3\sqrt{2}}{2},\ \sqrt{2}\}, \quad x_d = \{8\sqrt{2},\ -\frac{13\sqrt{2}}{2}\}, \quad x_s = \{-3,\ -5,\ -1,\ 5\}$$

è possibile ricostruire la sequenza di partenza applicando il procedimento illustrato più volte in precedenza, facendo riferimento alle informazioni sul nodo radice contenute in `t`*, calcolate alla prima chiamata. In ordine di esecuzione:*

```
t.corr[4] = (-3/SQRT2 + 13/SQRT2) * SQRT2inv =  5,
t.corr[5] = ( 2/SQRT  - 16/SQRT2) * SQRT2inv = -7,
t.corr[6] = ( 2/SQRT  + 16/SQRT2) * SQRT2inv =  9,
t.corr[7] = (-3/SQRT2 - 13/SQRT2) * SQRT2inv =  -8,
t.corr[0] = -3,
t.corr[1] = -5,
t.corr[2] = -1,
t.corr[3] =  5.
```

La sequenza ricostruita, alla fine dell'elaborazione, si trova nel nodo iniziale all'interno del campo `t.cur[]`. Inoltre, si noti come la deallocazione della memoria associata ai nodi non più necessari a future operazioni, in fondo ad ogni ricorsione, fa in modo che la memoria non venga appesantita più del dovuto.

Capitolo 4

Codice MatLab® e svolgimento dei test

Come brevemente anticipato alla fine del **capitolo 3**, esistono diversi indici che ci permettono di valutare e la capacità di ricostruzione a partire da pochi campioni della trasformata: ciò si rivela di fondamentale importanza poichè apre le strade al confronto con altre trasformate quali la ***Discrete Cosine Transform*** (*DCT*), utilizzata negli stadi della compressione *JPG*, o la più canonica e discussa *FFT*.

In questa sezione viene presentato e commentato il codice MatLab® di supporto utilizzato per compiere le comparazioni di cui sopra nonchè per esportare i risultati presentati e discussi nell'ultimo capitolo del documento.

4.1 (La necessità di scindere i due) casi in studio

Come prima cosa è necessario osservare che, come chiaro al lettore accorto, la trasformata presentata funziona intrinsecamente su sequenze monodimensionali.

La sua applicazione su dati bidimensionali, quali immagini, è quindi sostanzialmente un'applicazione della trasformata per righe (*rowwise*) o per colonne (*columnwise*).
Si rende perciò imprescindibile l'utilizzo di due differenti script che trattino i casi in maniera differente, anche in relazione al fatto che gli indici utilizzati per valutare la bontà dei dati ricostruiti cambiano nelle due circostanze.

Figura 4.1: *Trasformazione per sequenze monodimensionali*

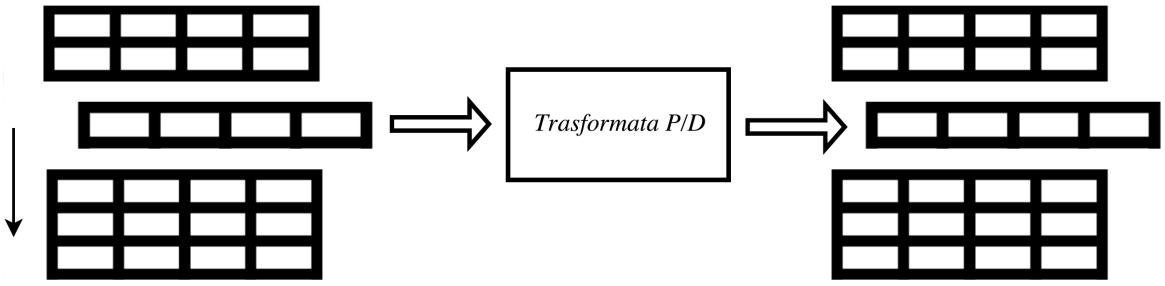


Figura 4.2: *Trasformazione per sequenze bidimensionali*

Implementativamente parlando, abbiamo ritenuto corretto operare una distinzione basata sull'estensione del file in ingresso allo script:

```
[~, orName, orExt] = fileparts(originalSignalPath);

% controlla se il file è un .mat (e.g. dati "raw" ECG)
ismat  = strcmp(orExt, '.mat');

% controlla se il file è un audio
iswav  = strcmp(orExt, '.wav');
ismp3  = strcmp(orExt, '.mp3');
ism4a  = strcmp(orExt, '.m4a') || strcmp(orExt, '.mp4') ...
    || strcmp(orExt, '.aac');
isflac = strcmp(orExt, '.flac');

% altrimenti, assumiamo sia un'immagine (.tiff, .jpg, ...)
```

Come accennato precedentemente, si scinde il caso monodimensionale da quello bidimensionale, invocando script differenti:

```
if( ismat || iswav || ismp3 || ism4a || isflac)
    elab1D(originalSignalPath, ...);

else
    elab2D(originalSigPath, ...);

end
```

Di seguito vengono riportate e commentate le sezioni salienti di entrambi gli script.

4.2 Elaborazione di segnali monodimensionali

Se il segnale da elaborare risulta essere monodimensionale (ECG o segnale digitalizzato, audio a singolo canale, ...), l'elaborazione viene iterata tante volte quanti sono i test di ricostruzione che vogliamo effettuare.

4.2.1 Trasformazione e anti-trasformazione

Come prima cosa viene caricato il file utilizzando la funzione più consona (in ragione della sua estensione), successivamente viene operato un controllo nella directory adibita a contenere i .txt dei segnali da elaborare (facendo riferimento al **capitolo 4**, si ricorda che lo script C accetta in ingresso segnali in formato testuale):

```
% Crea un array di struct, ognuna contenente informazioni riguardo
% ad ogni file nella directory:
dirContent = dir('path_txt_directory');

txtFound = false; % variabile di controllo booleana

% loop per ogni file nella directory, ricerca di una corrispondenza
% tra nomi
for(i = 1:length(dirContent))
    if(strcmp(dirContent(i).name, strcat(orName, '.txt')) == 1)
        txtFound = true;

        txtSigPath = strcat('path_txt_directory', orName, '.txt');
        break;
    end
end
```

Se il file di testo non è presente nella directory, si provvede alla sua creazione, rispettando la formattazione consona al funzionamento dello script in C:

```
if(txtFound == false)
    txtSigPath = strcat('path_txt_directory', orName, '.txt');
    file = fopen(txtSigPath, 'w');
    len = length(originalSig);

    % Il primo numero nel .txt è 1, (segnale monodimensionale 1xN),
    % poi le colonne N, \n e successivamente i valori (float)
    fprintf(file, '1 %d\n', len);
    fprintf(file, '%f ', originalSig);
    fclose(file);

end

clear file;
```

Dopo aver selezionato il path dell'eseguibile e il path del file di output, si procede invocando lo script con il parametro "-t":

```
cmd = sprintf('%s -t %s %s', 'path_eseguibile', txtSigPath, ...
    'path_output');
system(cmd);
```

In accordo con ciò che abbiamo presentato nel capitolo precedente, ciò comanda la creazione di un file binario contenente:

- il numero di righe (1) e il numero di colonne ("len");
- il vettore delle foglie ("leavesVector");
- la lunghezza del vettore degli n_0 ("n0sLen");
- il vettore degli n_0 ("n0s").

I dati nel file vengono letti e salvati nel workspace, dopodichè vengono ordinate le foglie in base all'energia ad esse associate (in modo crescente):

```
[~, index] = sort(abs(leavesVector), 'ascend');
```

Grazie a questo approccio, contrapposto a quello che vedeva l'elaborazione dati svolta direttamente dallo script in C, siamo in grado di gestire il taglio delle foglie meno energetiche in maniera più logica ed ordinata da MatLab®, servendoci di una variabile "leavesToCut":

```
newLeavesVector = leavesVector;
newLeavesVector(index(1:leavesToCut)) = 0;
```

Viene poi invocata la ricostruzione mediante chiamata allo script con parametro "-i": come esposto nel capitolo precedente in questo caso esso accetta in ingresso un file binario, che dobbiamo premurarci di creare ad-hoc:

```
newLeavesVectorFile = fopen('path_binario_ricostruzione', 'w');
fwrite(newLeavesVectorFile, 1, 'int32');
fwrite(newLeavesVectorFile, len, 'int32');
fwrite(newLeavesVectorFile, newLeavesVector, 'single');
fwrite(newLeavesVectorFile, n0sLen, 'int32');
fwrite(newLeavesVectorFile, n0s, 'single');
fclose(newLeavesVectorFile);

cmd = sprintf('%s -i %s %s', 'path_eseguibile', ...
    'path_binario_ricostruzione', 'path_file_output');
system(cmd);
```

4.2.2 Confronto con DCT e FFT: MSE ed η

A questo punto disponiamo sia del segnale originale che del segnale anti-trasformato (a partire da "len-leavesToCut" foglie). Eseguendo in maniera analoga il procedimento per le altre due trasformate in questione:

```
%% DCT e IDCT
dctSig = dct(originalSig);

% Ordinamento dei coefficienti in base alla loro importanza (weight-wise):
[~, ind] = sort(abs(dctSig),'ascend');

% Taglio di parte dei coefficienti:
dctSig(ind(1:leavesToCut)) = 0;

% Ricostruzione
idctSig = idct(dctSig)


%% FFT e IFFT
fftSig = fft(originalSig);

% Ordinamento dei coefficienti in base alla loro importanza (weight-wise)
[~, ind] = sort(abs(fftSig),'ascend');

% Taglio di parte dei coefficienti:
fftSig(ind(1:leavesToCut)) = 0;

% Ricostruzione
ifftSig = ifft(fftSig);
```

Dopodichè avviene il calcolo degli indici attraverso il quale confronteremo le varie trasformate. Nel caso di segnali monodimensionali, le quantità utilizzate per questo fine sono **l'errore quadratico medio** (*MSE*) e il **rapporto tra energie** (η), già presentati alla fine del **capitolo 3**.

```
errorPerSigPD   = originalSig - pdSig;
errorPerSigDCT  = originalSig - idctSig;
errorPerSigFFT  = originalSig - ifftSig;

MSEPD           = sum(errorPerSigPD.^2) / len;
MSEDCT          = sum(errorPerSigDCT.^2) / len;
MSEFFT          = sum(errorPerSigFFT.^2) / len;

enRatioPD       = sum(pdSig.^2)/sum(originalSig.^2);
enRatioDCT      = sum(idctSig.^2)/sum(originalSig.^2);
enRatioFFT      = sum(ifftSig.^2)/sum(originalSig.^2);
```

4.3 Elaborazione di segnali bidimensionali

Se il segnale da elaborare risulta essere bidimensionale (come nel caso di immagini), l'elaborazione viene iterata tante volte quante sono le righe del segnale stesso e tante quanti sono i test di ricostruzione che vogliamo effettuare. Data la similitudine tra molte porzioni di codice dell'elaborazione bidimensionale e quella monodimensionale, alcune parti verranno volutamente omesse (previa segnalazione al lettore) per promuovere, al contrario, le differenze tra i due script.

4.3.1 Trasformazione e anti-trasformazione

Dopo aver caricato il file utilizzando la funzione più consona (in ragione della sua estensione) e operato un controllo nella directory adibita a contenere i .txt dei segnali da elaborare (secondo le modalità illustrate nella sezione precedente), viene richiamato lo script in C mediante il codice precedentemente commentato.
Questa volta, essendo il dato in questione una matrice, il file binario in output conterrà:

- il numero di righe (`"nrows"`) e il numero di colonne (`"ncol"`);
- il vettore delle foglie della **prima** riga;
- la lunghezza del vettore degli n_0 della prima riga;
- il vettore degli n_0 della prima riga;
- il vettore delle foglie della **seconda** riga;

...e via dicendo (per `nrows` volte). Come per il caso monodimensionale, i dati vengono letti e salvati nel workspace. Si noti come, mentre la lunghezza delle sequenze trasformate risulta essere sempre identica (i.e. `ncol` è uguale per ogni riga), la lunghezza dei vettori `n0s` cambia in funzione della riga presa in considerazione. È perciò necessario utilizzare delle strutture dinamiche (array di array di lunghezza differente) che MatLab ® mette a disposizione sotto il nome di *cell*. Proseguendo, in questa istanza ci si rende subito conto come sia possibile ordinare le foglie in due modi differenti:

1. In maniera crescente (rispetto alla loro energia) riga per riga: il codice in questione risulta una semplice iterazione di quello presentato nella sezione precedente;
2. In maniera crescente (rispetto alla loro energia) facendo riferimento alla totalità delle righe presenti nella matrice:

```
leavesVector = reshape(leavesMatrix, 1, nrow*ncol);
newLeavesVect = leavesVector;
newLeavesVect(index(leavesToCut)) = 0;
newLeavesMatrix = reshape(cutLeavesVect, nrow, ncol);
```

Nel caso in cui venga implementato il primo, ogni riga viene trattata alla stessa maniera, ovvero viene ricostruita a partire da un numero fisso di foglie, mentre nel caso in cui venga prediletto il secondo approccio i campioni con i quali avviene la ricostruzione variano riga per riga (si rimanda all'ultima parte del **capitolo 5** la discussione inerente alle differenze nel segnale ricostruito a seconda dell'ordinamento utilizzato).
Successivamente lo script esegue la ricostruzione richiamando l'eseguibile con il parametro `"-i"` e il percorso di un file binario creato ad-hoc, contenente i nuovi vettore delle foglie, i vettori contenenti le posizioni delle assi di simmetria n_0 e le rispettive lunghezze:

```
newLeavesMatrixFile = fopen('path_binario_ricostruzione', 'w');
fwrite(newLeavesMatrixFile, row, 'int32');
fwrite(newLeavesMatrixFile, col, 'int32');

for(i=1:nrow)
    fwrite(newLeavesMatFile, newLeavesMatrix(i, :), 'single');
    fwrite(newLeavesMatFile, n0sLenghtsVector(i), 'int32');
    fwrite(newLeavesMatFile, n0s{i}, 'single');
end

fclose(cutLeavesMatFile);

cmd = sprintf('%s -i %s %s', 'path_eseguibile', ...
    'path_binario_ricostruzione', 'path_file_output');
system(cmd);
```

dove, per chiarezza, `"n0s{i}"` è la modalità d'accesso canonica all'i-esima cell (contenente un vettore) dell'array di array.

4.3.2 Confronto con DCT e FFT: MSE, PSNR ed η

Come per il caso monodimensionale, vengono computate entrambe le trasformate con le quali vogliamo confrontare la nostra implementazione, e vengono ricostruiti i segnali originali preservando solo parte dei coefficienti.
Si noti come, anche questa volta, la versione della DCT presa in considerazione è quella monodimensionale (iterata riga per riga):

```
for(k=1:nrow)
    dctMat(k, :) = dct(originalImage(k, :));
end

dctRow = reshape(dctMat, 1, nrow*ncol);
[~, ind]  = sort(abs(dctRow),'descend');
dctRow(ind(end-leavesToCut:end)) = 0;
dctMat = reshape(dctRow, nrow, ncol);

for(k=1:nrow)
    idctImg(k, :) = idct(dctMat(k, :));
end
```

e similmente per la FFT. Data la diversa natura dei segnali in questione rispetto a quelli precedentemente trattati, abbiamo trovato consono l'inserimento di un terzo parametro con il quale misurare la bontà della ricostruzione, vale a dire il **Peak signal-to-noise ratio** (*PSNR*). Esso è definito come:

$$PSNR = 10 \cdot log_{10} \frac{(Maximum\ pixel\ value)^2}{MSE}$$

È necessario fare qualche osservazione:

- Benchè sia usuale l'utilizzo di 256 livelli per canale, il valore massimo che un pixel può assumere dipende dai bit del quantizzatore responsabile della digitalizzazione dell'immagine;
- Il PSNR è spesso utilizzato per valutare la qualità della ricostruzione di compressioni (chiaramente di tipo *lossy*). Il motivo risiede nel fatto che essendo inversamente proporzionale all'MSE (che in questa istanza misura la differenza media tra i valori dei pixel dell'immagine originale e quelli dell'immagine ricostruita), esso dà modo di valutare il degrado della qualità dell'immagine percepibile dall'occhio umano.

Il codice risulta molto simile a quello presentato nella **sezione 5.2.2**, sebbene riadattato al caso bidimensionale e con l'inserimento del parametro di cui sopra:

```
squaredErrorImagePD  = (double(originalImg) - double(pdImg)) .^2;
squaredErrorImageDCT = (double(originalImg) - double(idctImg)) .^2;
squaredErrorImageFFT = (double(originalImg) - double(ifftImg)) .^2;

MSEPD   = sum(sum(squaredErrorImagePD)) / (nrow * ncol);
MSEDCT  = sum(sum(squaredErrorImageDCT)) / (nrow * ncol);
MSEFFT  = sum(sum(squaredErrorImageFFT)) / (nrow * ncol);

enRatioPD  = sum(sum(pdImg.^2))/sum(sum(single(originalImg).^2));
enRatioDCT = sum(sum(idctImg.^2))/sum(sum(single(originalImg).^2));
enRatioFFT = sum(sum(ifftImg.^2))/sum(sum(single(originalImg).^2));

PSNRPD  = 10 * log10( 255^2 / MSEPD);
PSNRDCT = 10 * log10( 255^2 / MSEDCT);
PSNRFFT = 10 * log10( 255^2 / MSEFFT);
```

Dato che per percentuali di foglie/coefficienti tagliati molto basse l'argomento del logaritmo tende a zero (poichè l'MSE risulta, ovviamente, molto basso), il PSNR potrebbe, per approssimazione, risultare infinito.
Per non incappare in questi problemi (non rappresentabili graficamente) si è scelto di porre un vincolo al Peak signal-to-noise ratio secondo il codice:

```
if(isinf(PSNRPD))
    PSNRPD  = 100;
end
```

identico anche per `PSNRDCT` e `PSNRFFT`.

Capitolo 5

Risultati

Come discusso nei primi capitoli, uno degli indici che ci consente di valutare la bontà di una trasformata è l'errore commesso nella ricostruzione a partire da un numero limitato di coefficienti. Grazie al codice di supporto presentato e analizzato nel **capitolo 5**, sono stati eseguiti svariati test che mettono a confronto quella che è stata rinominata "trasformata pari/dispari" (nei grafici *"PD"*), la trasformata discreta del coseno (DCT) e la più canonica trasformata veloce di fourier (FFT), sia nell'ambito di segnali monodimensionali (per i quali la trasformata pari/dispari è intrinsecamente adatta) che nell'ambito di segnali bidimensionali. Prima di procedere è bene puntualizzare che:

- Per quanto riguarda i risultati associati all'elaborazione di segnali monodimensionali, abbiamo deciso di graficare solo parte del segnale (originale e ricostruito) al fine di rendere visibile ciò che la ricostruzione comporta. I segnali su cui sono stati eseguiti i test, infatti, hanno un numero di samples che si aggira attorno alla centinaia di migliaia: riportare l'intero segnale non sarebbe utili a fini esplicativi. Al contrario, ovviamente, i grafici di MSE e dell'indice η sono relativi all'intera elaborazione;

- Per quanto riguarda l'elaborazione di segnali bidimensionali, è fondamentale ripetere che la DCT utilizzata nei confronti è la **versione monodimensionale** della trasformata, e che ordinamento e taglio dei coefficienti sono stati compiuti sulla totalità degli stessi (esattamente come accade per la PD).

In ordine di trattazione, vengono riportati risultati dell'elaborazione di:

1. Segnali digitali generati con MatLab®, utili per l'interpretazione e il commento di alcuni aspetti della ricostruzione;

2. Elettrocardiogrammi con frequenze di campionamento tra i 250 e i 350 Hz ottenuti dal database open source *PhysioBank* mantenuto da *PhysioNet* [3];

3. Alcuni segnali audio, campionati a 44100 Hz, da noi acquisiti;

4. Immagini in scala di grigi, 512x512, tra le quali alcune tra le classiche per l'elaborazione di immagini [4], e altre cercate ad-hoc per sottolineare alcuni comportamenti della trasformata.

5.1 Segnali monodimensionali: vettori, ECG, audio.

5.1.1 Segnali digitali - seno

Riportiamo di seguito errore quadratico medio *MSE* e percentuale di energia conservata η ottenuti dalla ricostruzione di 1000 campioni del segnale discreto:

$$x[n] = \sin\left(2\pi \cdot \frac{n}{20}\right)$$

al variare della percentuale dei coefficienti azzerati.

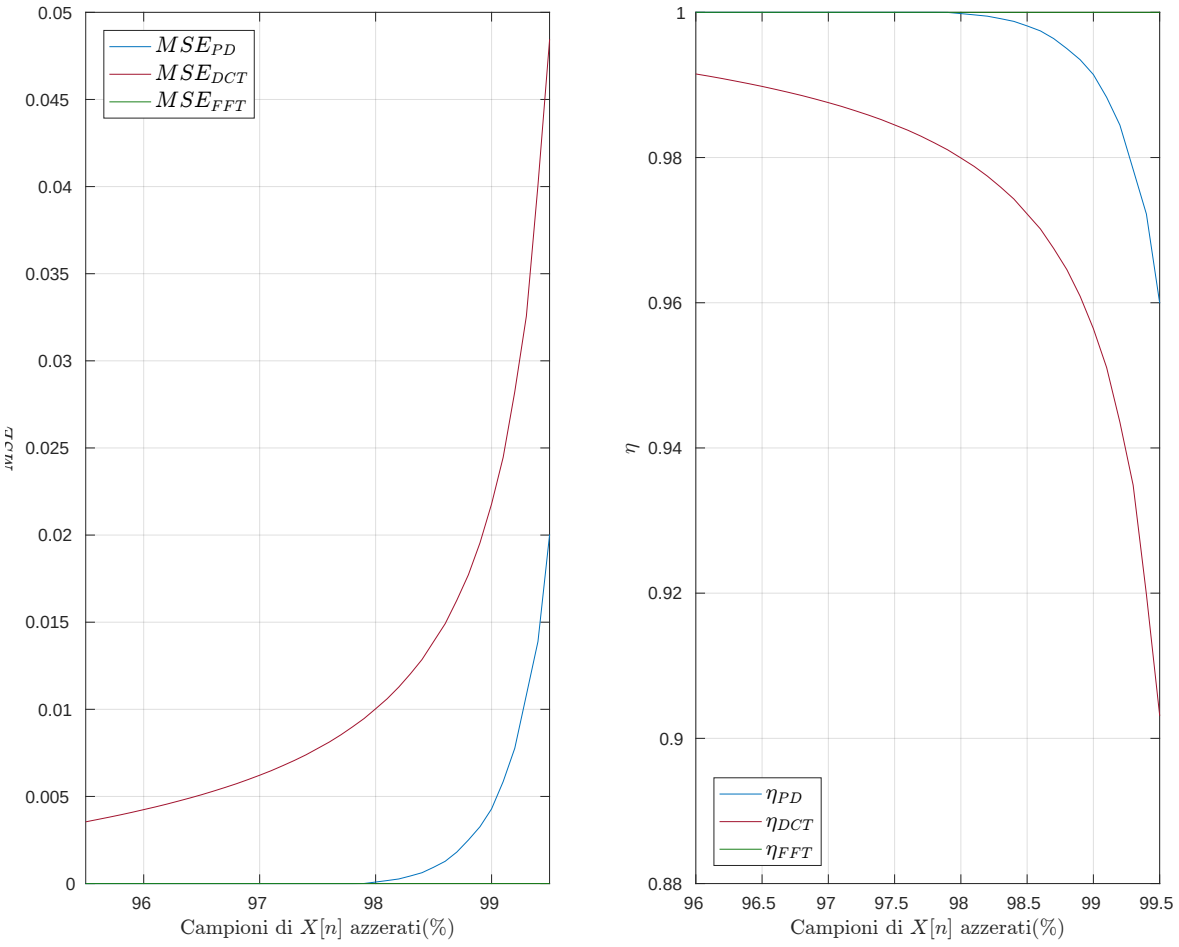


Figura 5.1: *MSE e* η *di* $\sin\left(2\pi \cdot \frac{n}{20}\right)$*: trasformate a confronto*

Come si nota da **figura 5.1**, la *FFT* è in questo caso la trasformata più performante: essendo il segnale in questione una sinusoide, essa necessita di un solo coefficiente per ricostruirla senza il minimo errore. La *DCT*, al contrario, si dimostra meno performante della trasformata *PD* (data la simmetria del segnale, ciò era tanto prevedibile quanto auspicabile). Nelle **figure 5.2 e 5.3** riportiamo i campioni da $n = 1$ ad $n = 100$ del segnale ricostruito (trasformata *PD*) in diversi casi, insieme all'errore di ricostruzione.

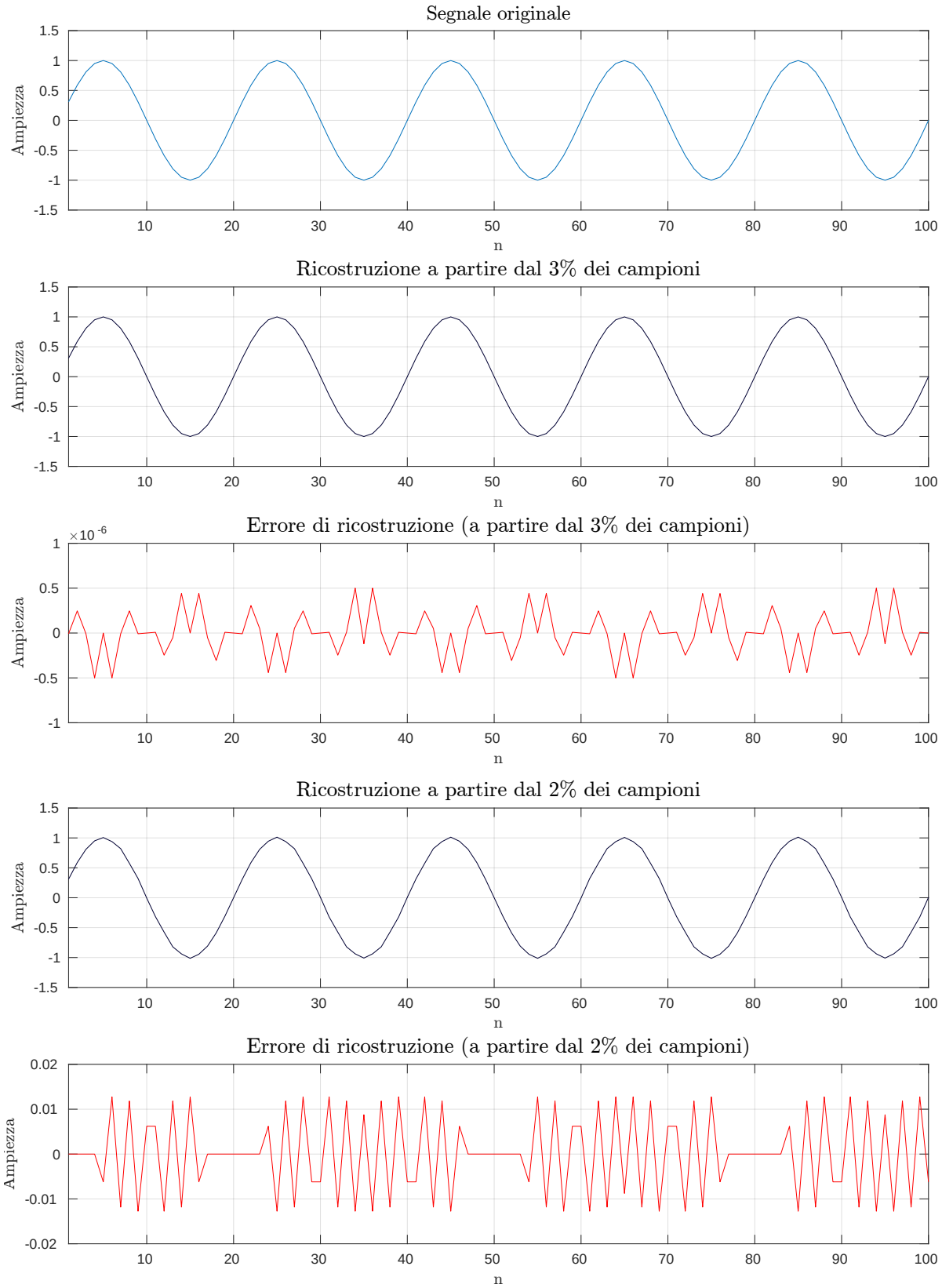


Figura 5.2: *Ricostruzione di* $sin[2\pi \cdot \frac{n}{20}]$ *a partire dal* 3% *e* 2% *dei coefficienti*

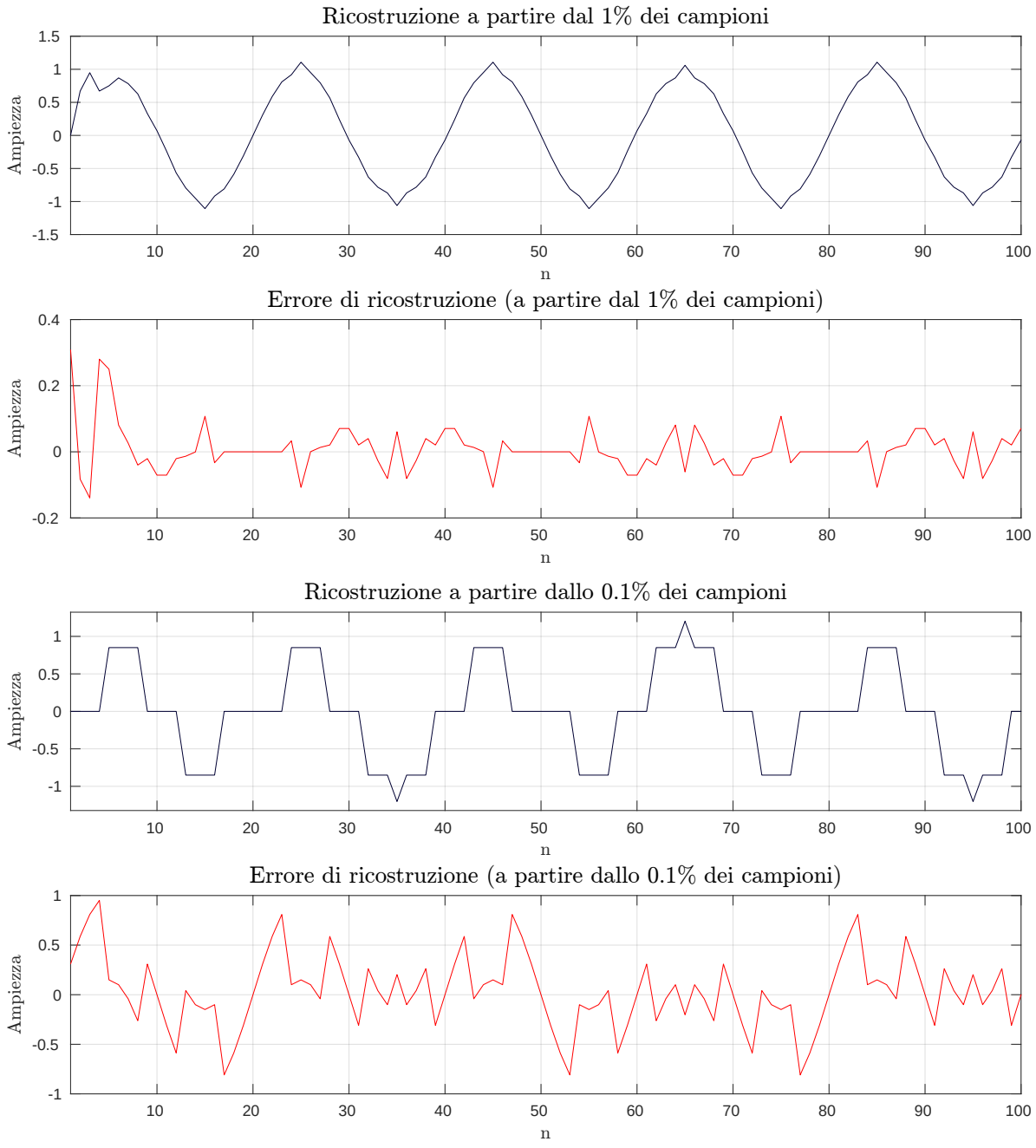


Figura 5.3: *Ricostruzione di* $\sin\left(2\pi \cdot \frac{n}{20}\right)$ *a partire dall'*1% *e dallo* 0,1% *dei coefficienti*

5.1.2 Segnali digitali - seno e rumore gaussiano bianco additivo

Se al segnale precedente (riscalato) aggiungiamo un rumore gaussiano bianco additivo (*AWGN*) generato con il comando `"randn(1, 1000)"` (la cui funzione di densità di probabilità delle ampiezze è una normale standard) otteniamo:

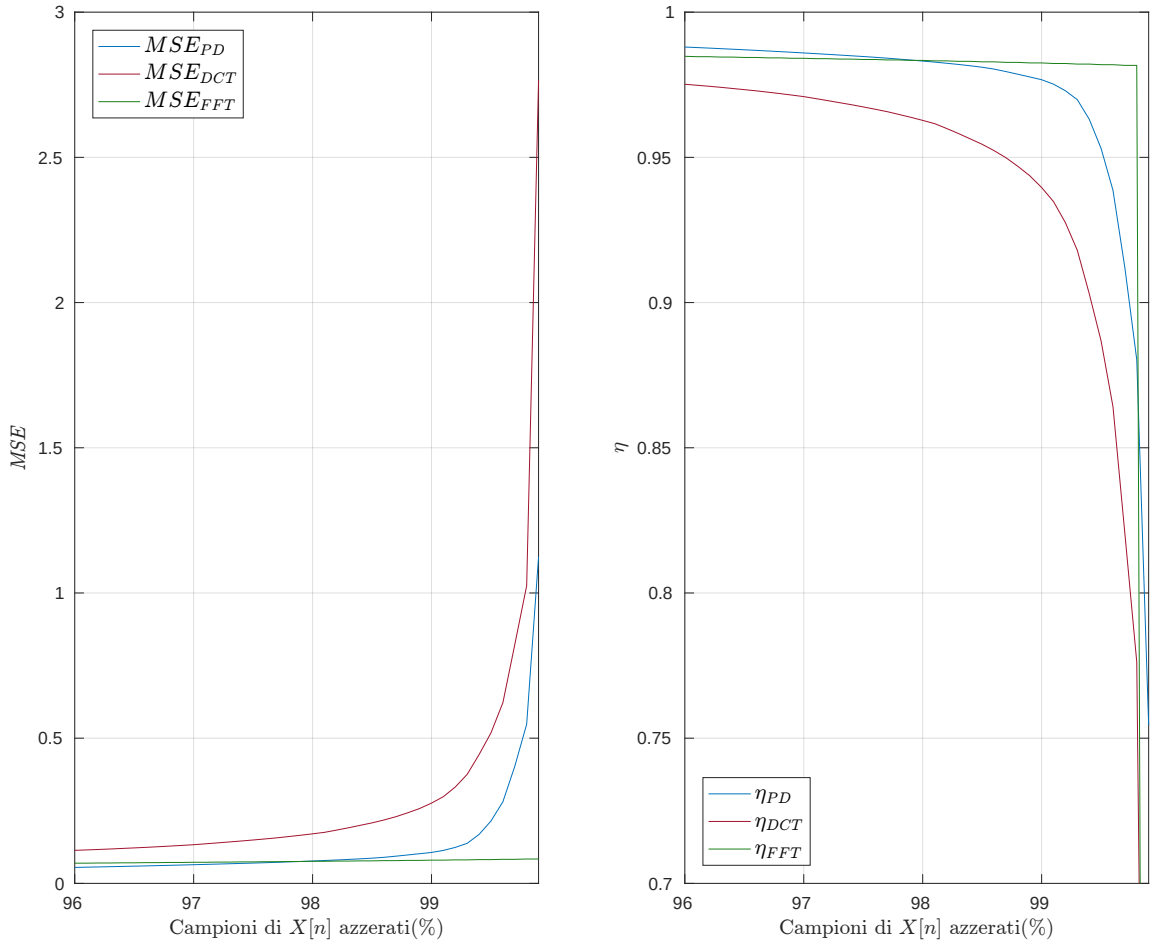


Figura 5.4: *MSE e* η *di* $\sin\left(2\pi \cdot \frac{n}{20}\right) + \omega(n)$*: trasformate a confronto*

Come si nota in **figura 5.4**, questa volta la trasformata *PD* garantisce ottimi risultati anche in confronto alla *FFT* (per percentuali di campioni azzerati minori del 99%): ciò è dovuto alla presenza, appunto, di rumore gaussiano bianco, avente componenti frequenziali significative su di una grossa banda (idealmente da $-\infty$ a $+\infty$).

È interessante notare come, poichè la nostra trasformata tende a preservare al meglio i segnali (o parti di essi) caratterizzati da una certa simmetria, il rumore additivo appaia quasi come filtrato, fintantochè di ampiezza massima limitata, quantomeno in ricostruzioni a partire da un numero di coefficienti della trasformata non troppo limitato (ciò è ben visibile nella **figura 5.5**).

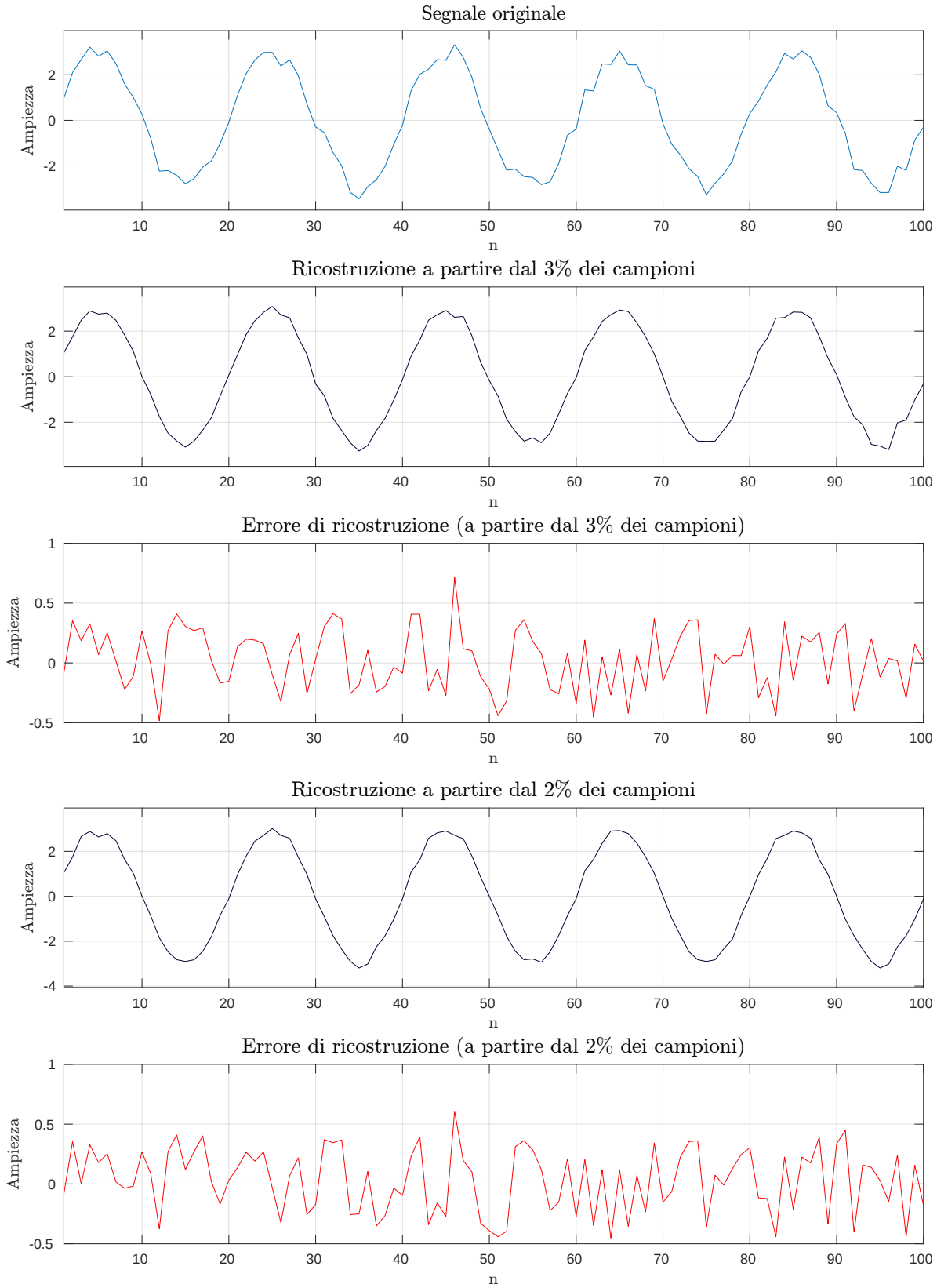


Figura 5.5: *Ricostruzione di* $\sin\left(2\pi \cdot \frac{n}{20}\right) + \omega(n)$ *a partire dal* 3% *e* 2% *dei coefficienti*

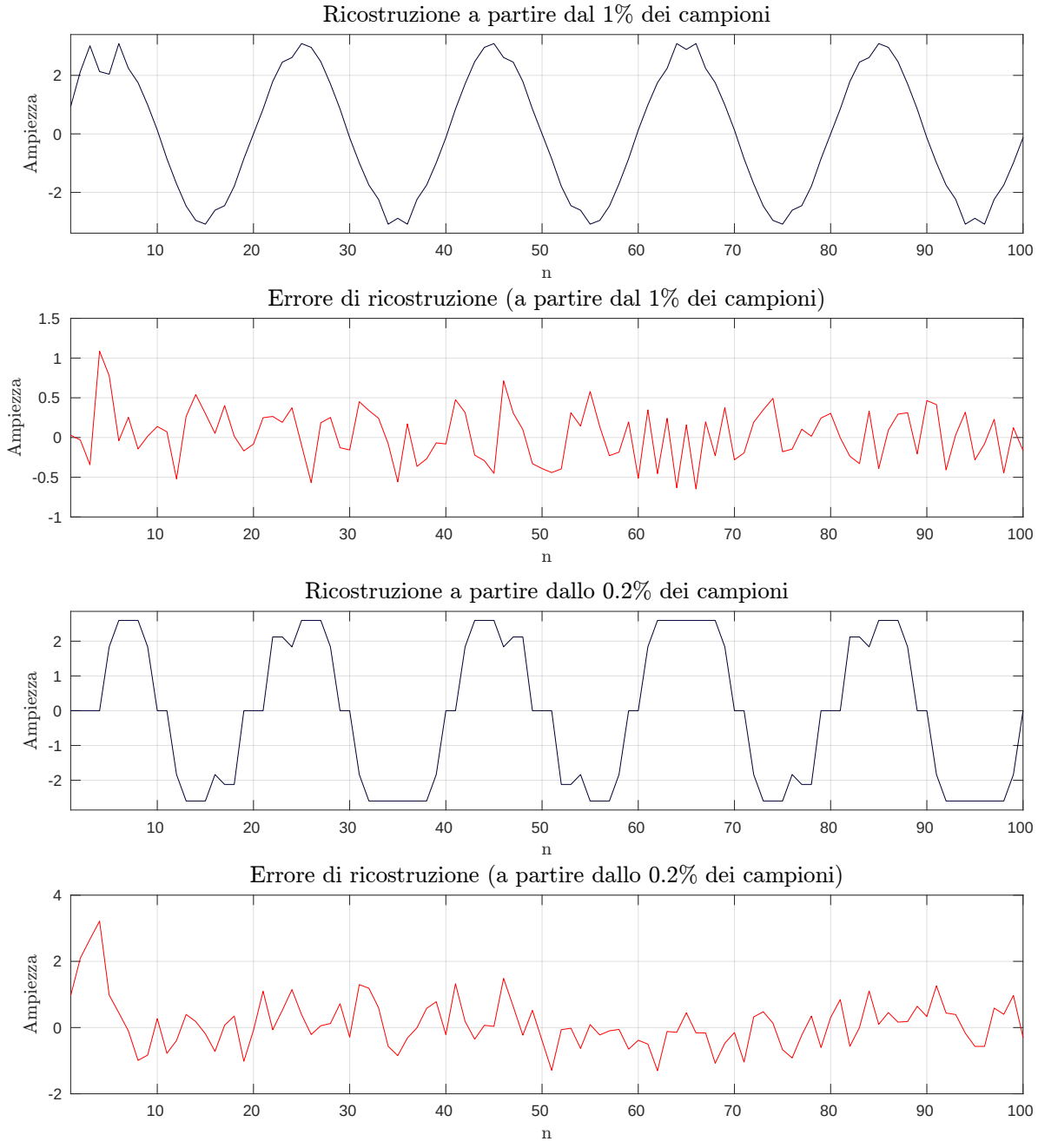


Figura 5.6: *Ricostruzione di* $\sin\left(2\pi \cdot \frac{n}{20}\right) + \omega(n)$ *a partire dall'*1% *e dallo* $0,2\%$ *dei coefficienti*

5.1.3 Segnali digitali - rumore gaussiano bianco

Il segnale preso in considerazione questa volta è un rumore gaussiano bianco (*AWGN*), generato con il comando `"randn(1, 1000)"` (la cui funzione di densità di probabilità delle ampiezze è una normale standard).

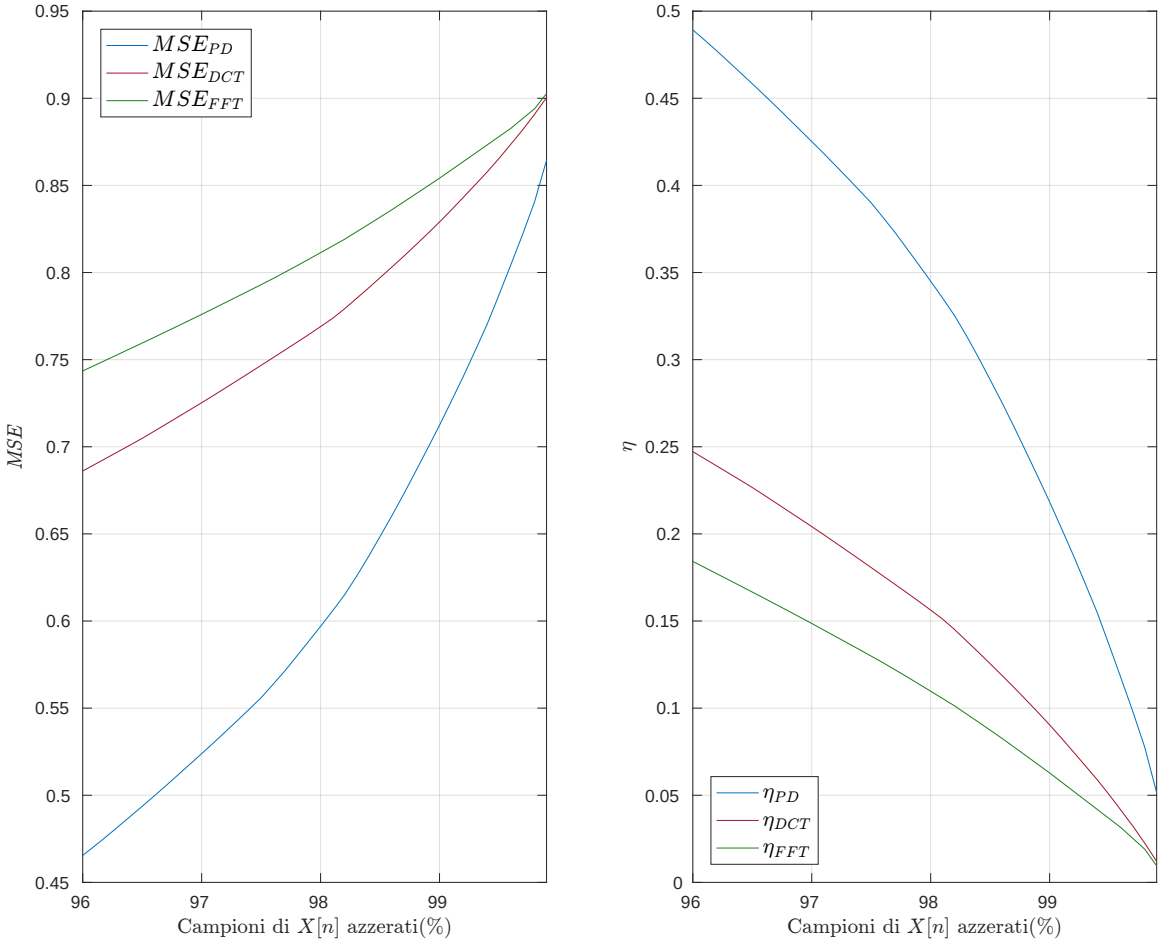


Figura 5.7: *MSE e η di $\omega(n)$: trasformate a confronto*

Come chiaro da **figura 5.7**, questa volta la trasformata *PD* eccelle rispetto alle concorrenti: come accennato nella parte introduttiva del lavoro, la trasformata *PD* adotta un approccio che in questi casi risulta vincente.

Mentre *FFT* e *DCT* tentano di approssimare il rumore bianco mediante somma di armoniche (arduo compito, date le escursioni repentine del segnale in questione), la trasformata pari-dispari sfrutta le caratteristiche simmetriche portate proprio da questa variabilità. Ovviamente all'aumentare della percentuale dei campioni tagliati prima della ricostruzione, ciò che rimane del segnale originale è vagamente la forma, "lo scheletro", costruito però dall'unione di strutture più regolari e caratterizzate da una forte simmetria.

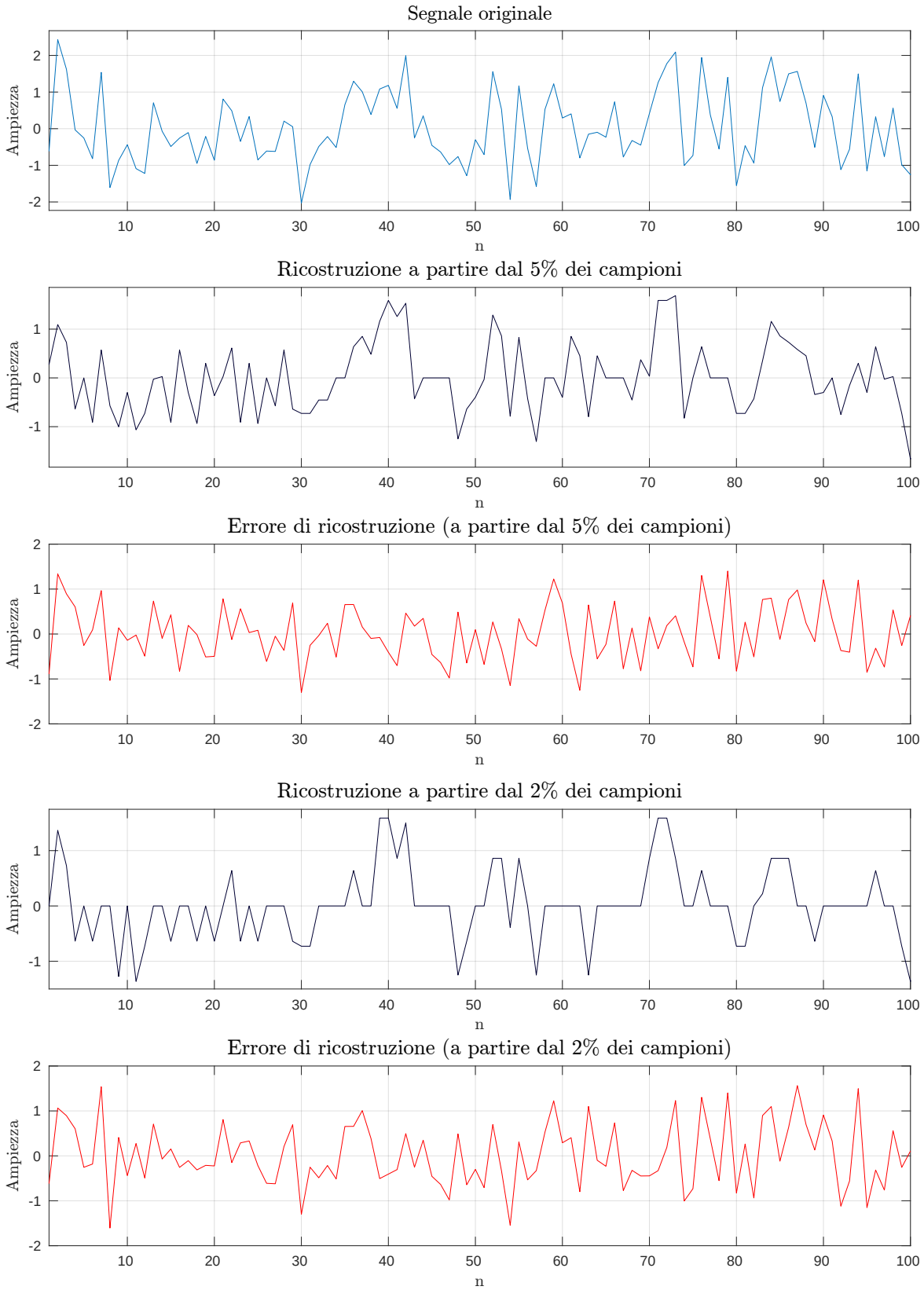


Figura 5.8: *Ricostruzione di $\omega(n)$ a partire dal 5% e 2% dei coefficienti*

Eclatante è l'esempio riportato in **figura 5.9**: possiamo vedere come, a partire da pochissimi campioni (in particolare, in questo caso, solamente 50 su un totale di 1000) la ricostruzione cerchi di ricalcare il segnale di partenza utilizzando solamente strutture perfettamente simmetriche, come visibile nella prima parte del segnale ricostruito, o quasi, come chiaro guardandone l'ultima parte.

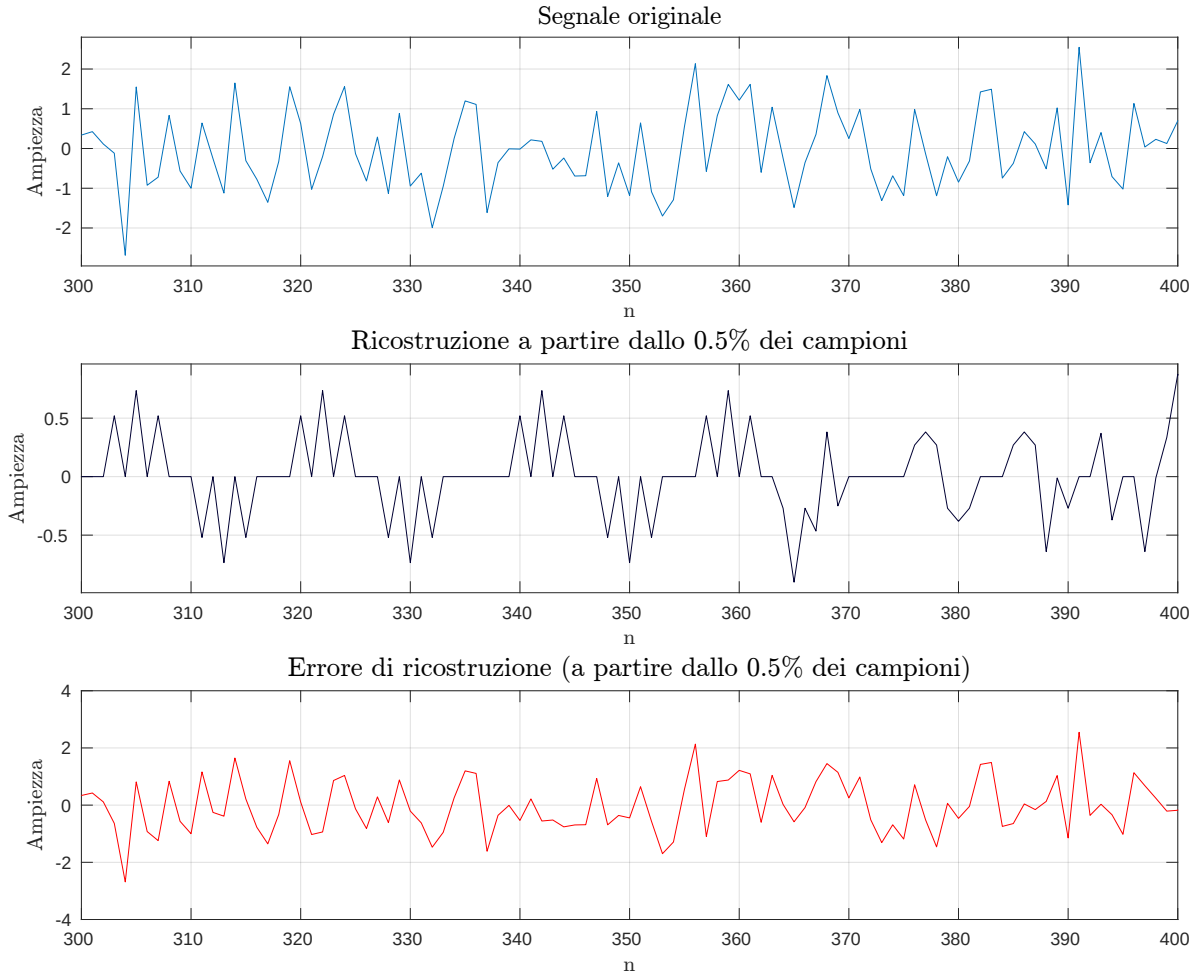


Figura 5.9: *Ricostruzione di* $\omega(n)$ *a partire dallo* 0.5% *dei coefficienti*

5.1.4 ECG 1 - paziente in salute

Cambiando completamente ambito, abbiamo eseguito svariati test anche su segnali relativi a elettrocardiogrammi campionati e quantizzati. Per cominciare, presentiamo i risultati ottenuti dall'elaborazione di un **segnale ECG regolare non pre-elaborato**, relativo ad un paziente in buona salute:

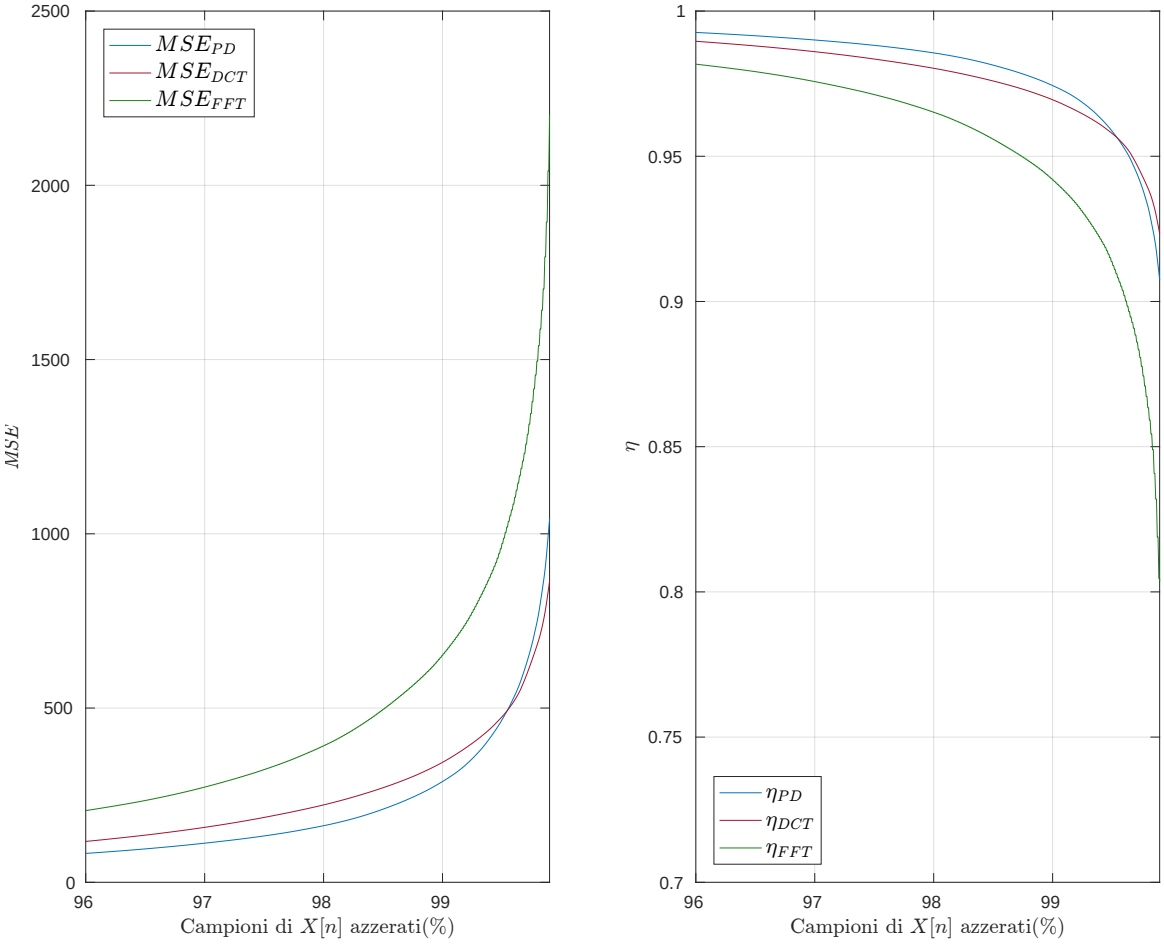


Figura 5.10: *MSE e η relativi all'ECG di un paziente sano: trasformate a confronto*

Godendo in generale di una componente simmetrica non indifferente, il segnale ECG si dimostra essere un buon candidato all'applicazione della trasformata.
Si tenga conto del fatto che i risultati riportati in **figura 5.10** sono relativi a segnali che non hanno subito alcun tipo di elaborazione preliminare: mentre per la *DCT* e la *FFT* la pulizia del segnale potrebbe risultare poco efficace in termini di aumento dell'*MSE*, per quanto riguarda la trasformata *PD* ciò potrebbe portare ad un significativo aumento.

È facile, infatti, dalle immagini che seguono (**figura 5.11**), giudicare come il rumore di fondo presente tra i complessi QRS non aiuti l'incidenza di parti di segnale simmetriche. Si noti infine come nelle ricostruzioni a partire dal 5% dei coefficienti (50 su 1000) e dall'1% dei coefficienti (10 su 1000), il complesso QRS e la forma d'onda tipica dell'ECG risultino ancora ben osservabili.

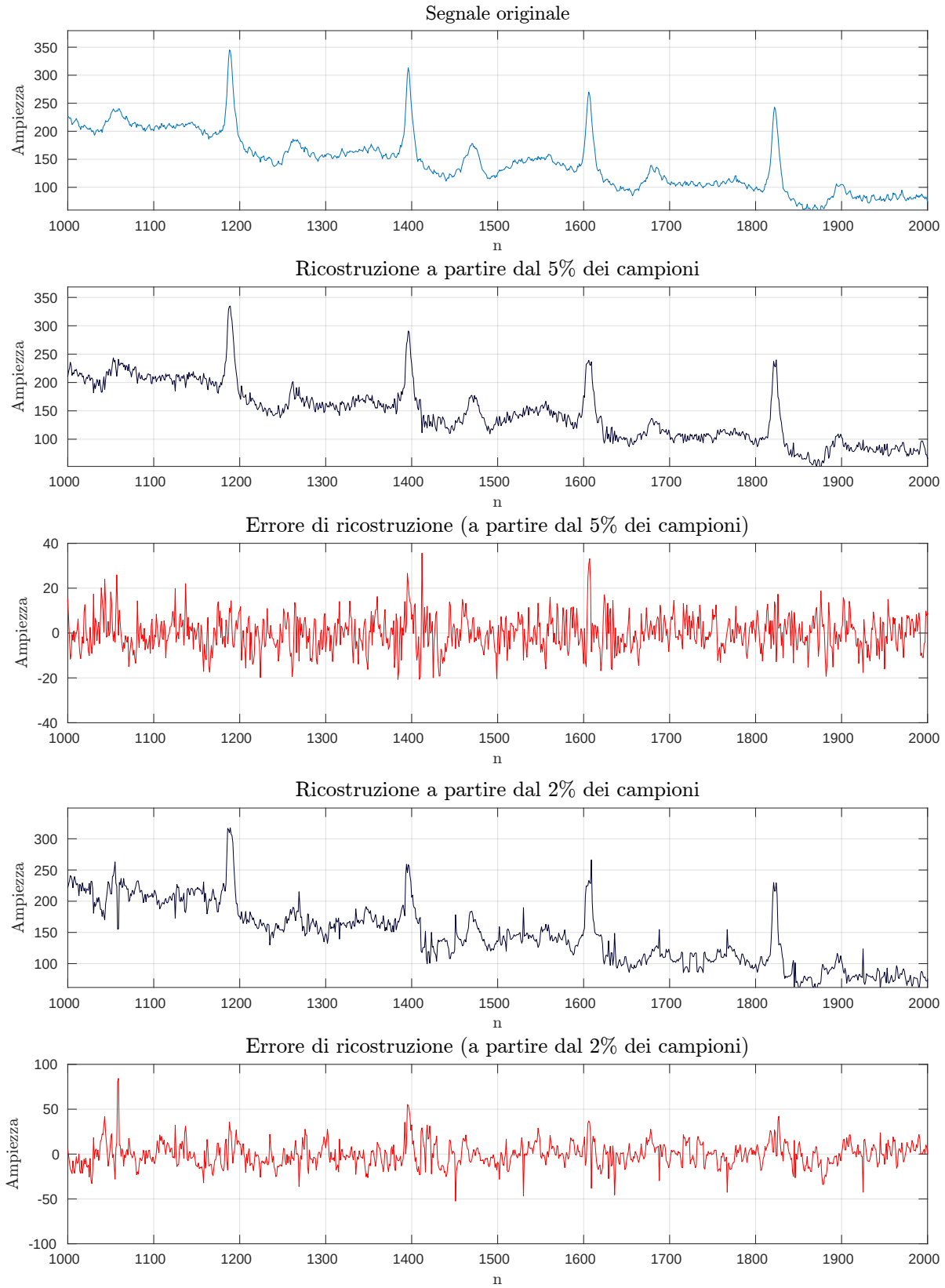


Figura 5.11: *Ricostruzione dell'ECG di un paziente sano a partire dal 5% e 2% dei coefficienti*

5.1.5 ECG 2 - paziente affetto da aritmia ventricolare

Proseguiamo presentando i risultati ottenuti dall'elaborazione di un **segnale ECG relativo ad un paziente affetto da aritmie ventricolari non pre-elaborato**, che rendono il complesso QRS irregolare e introducono nello spettro del segnale componenti a bassa frequenza (attorno ai 4 Hz):

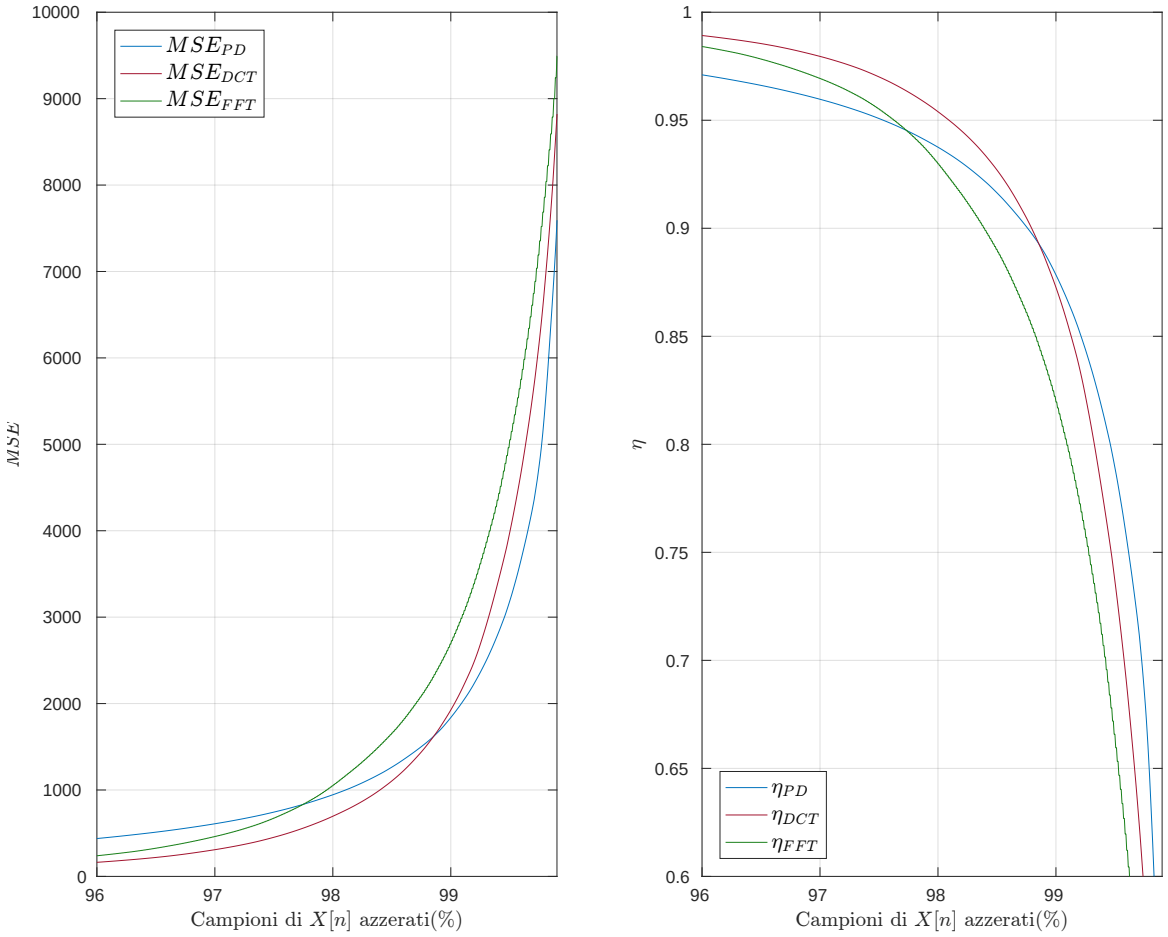


Figura 5.12: *MSE e η relativi all'ECG di un paziente aritmico: trasformate a confronto*

Questa volta, la trasformata *PD* si dimostra di minore efficacia rispetto alle concorrenti *DCT* ed *FFT* (**figura 5.12**): ciò è intuitivamente dovuto alla presenza in diversi istanti casuali di componenti a basse frequenze portate dalla patologia del paziente.
Si tenga conto, nell'interpretazione dei risultati, del fatto che il segnale in questione consta di 75000 campioni e di ben 10 occorrenze (ciascuna tipicamente di una decina di secondi, ovvero circa 3000 samples) di aritmia, che inficiano la ricerca di simmetrie compiuta dall'algoritmo di trasformazione.

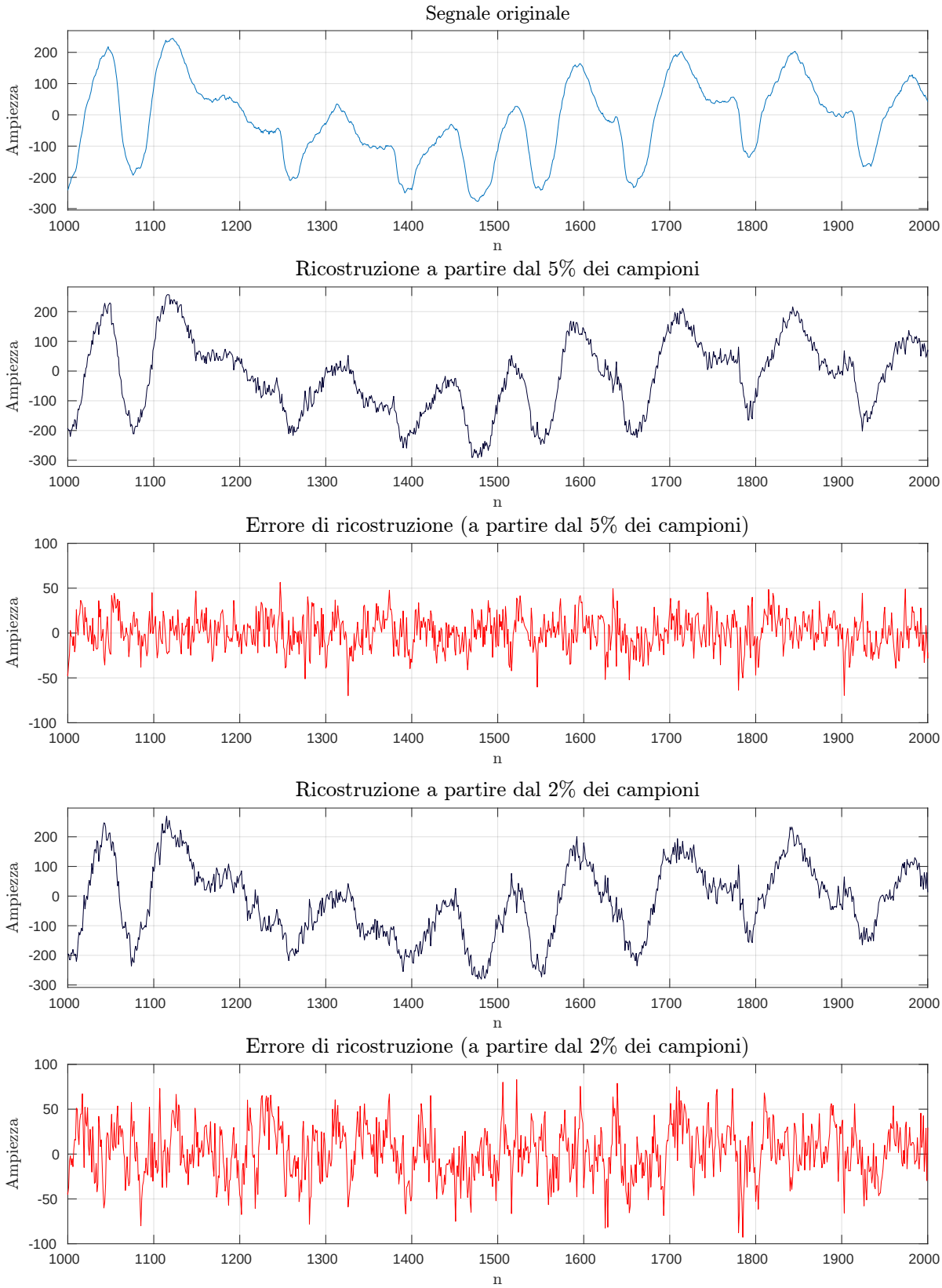


Figura 5.13: *Ricostruzione dell'ECG di un paziente aritmico a partire dal* 5% *e* 2% *dei coefficienti*

5.1.6 Segnale audio - pronuncia della parola *"ciao"*

Proseguiamo presentando i risultati ottenuti dall'elaborazione di un **segnale audio da noi acquisito, campionato alla frequenza di** 44100 Hz:

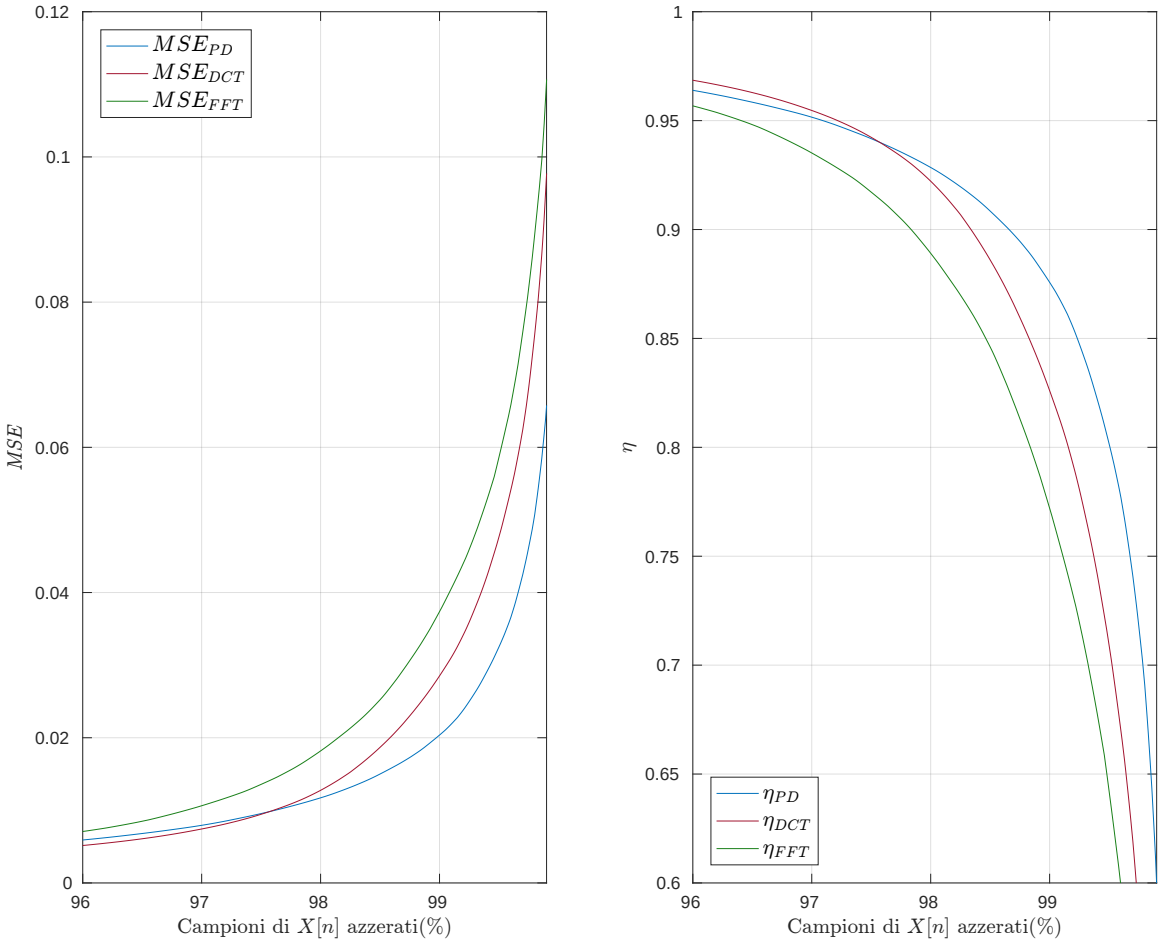


Figura 5.14: *MSE e η relativi al segnale audio (pronuncia della parola "ciao")*

Come si nota in **figura 5.14**, la trasformata *PD* vanta una migliore capacità di rappresentazione, e quindi una più accurata ricostruzione, per percentuali di campioni azzerati molto elevate (sempre dalla **5.14**, da circa il 97%).
Proseguendo, in **figura 5.15** è riportata una porzione di segnale relativa alla pronuncia del fonema *"/a/"*, caratterizzato da forma d'onda (quasi) periodica [5].
La ricorrente presenza di tali componenti rende il segnale affine all'elaborazione con la trasformata *PD*, e l'entità dell'errore molto simile a quella riscontrata nei segnali elettrocardiografici[1].

[1]L'errore è ovviamente da considerare rispetto alla dinamica del segnale

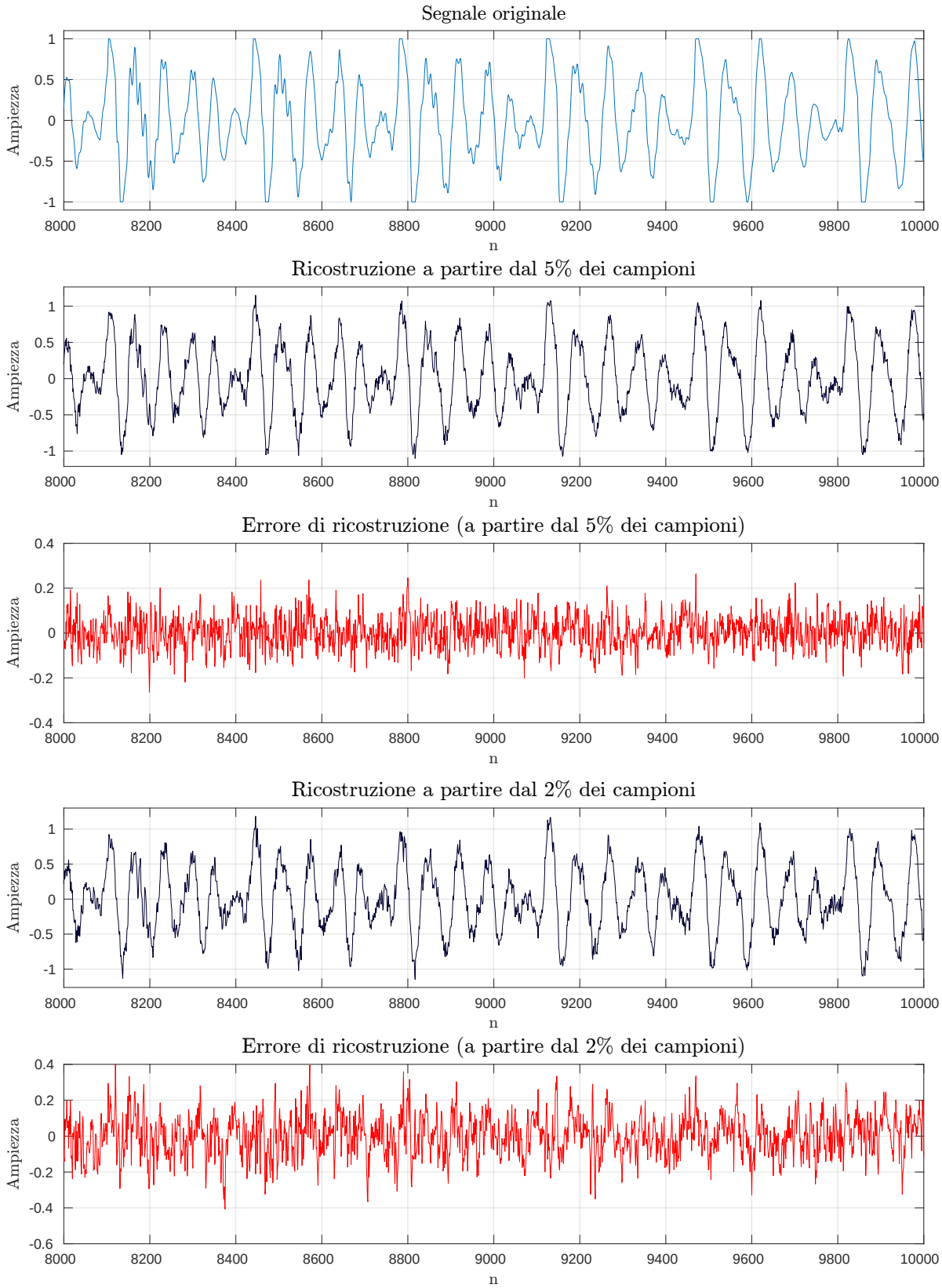


Figura 5.15: *Ricostruzione dell'audio in questione a partire dal* 5% *e* 2% *dei coefficienti*

5.2 Segnali bidimensionali: immagini in scala di grigi

5.2.1 Lena

Le seguenti figure (**5.16, 5.17, 5.18**) affiancano all'originale dell'arcinota modella svedese le ricostruzioni con le diverse anti-trasformate, a partire dal 5% dei campioni.
Nella **figura 5.19**, invece, viene graficato l'andamento dei parametri di confronto.

Immagine originale in scala di grigi

Immagine ricostruita a partire dal 5% dei campioni (PD)

Figura 5.16: *Degrado nella ricostruzione a partire dal 5% dei campioni (PD)*

Immagine originale in scala di grigi

Ricostruzione a partire dal 5% dei campioni (DCT)

Figura 5.17: *Degrado nella ricostruzione a partire dal 5% dei campioni (DCT)*

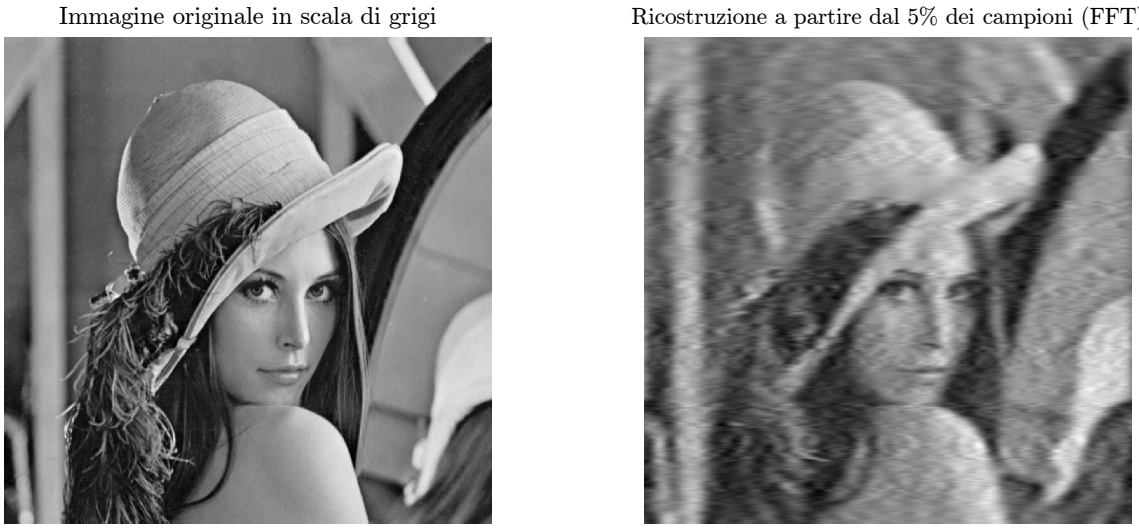


Figura 5.18: *Degrado nella ricostruzione a partire dal 5% dei campioni (FFT)*

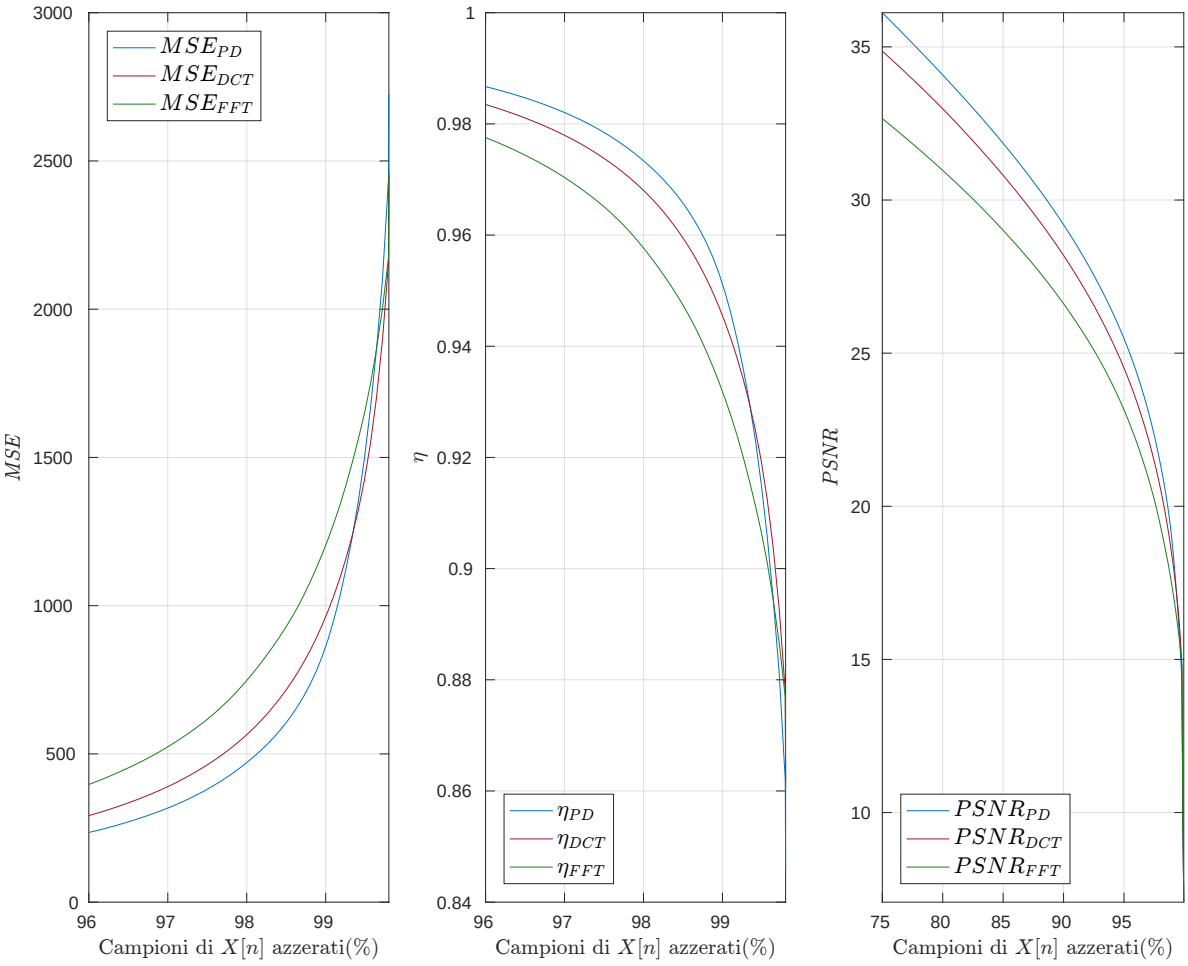


Figura 5.19: *Confrontro tra* MSE, η, PSNR *delle trasformate*

5.2.2 To The Moon®

Le seguenti figure (**5.20, 5.21, 5.22**) affiancano all'originale, tratta da l'omonimo videogioco sviluppato da Freebird Games, le ricostruzioni con le diverse anti-trasformate, a partire dal 5% dei campioni.
Nella **figura 5.23**, invece, viene graficato l'andamento dei parametri di confronto.

Immagine originale in scala di grigi

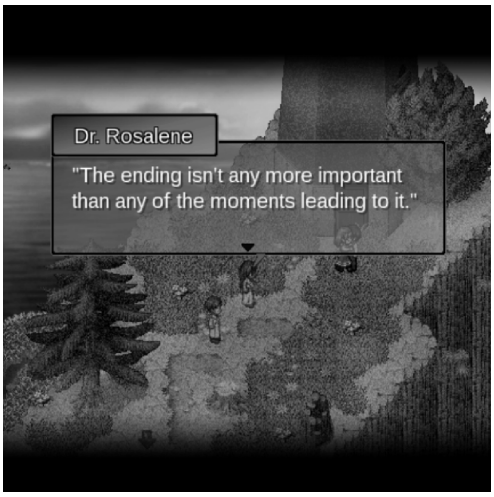


Immagine ricostruita a partire dal 5% dei campioni (PD)

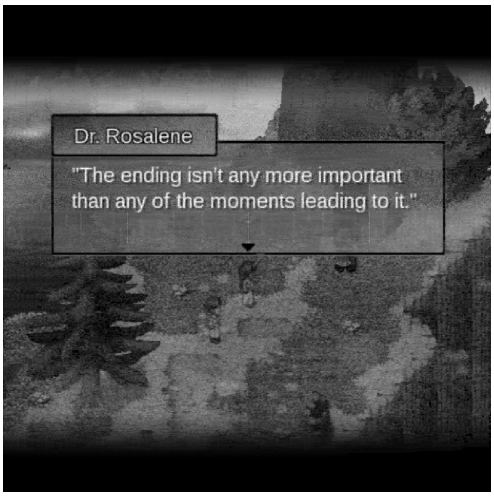
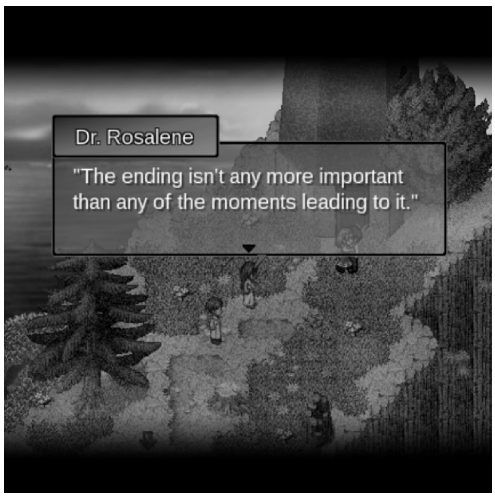


Figura 5.20: *Degrado nella ricostruzione a partire dal 5% dei campioni (EOT)*

Immagine originale in scala di grigi



Ricostruzione a partire dal 5% dei campioni (DCT)

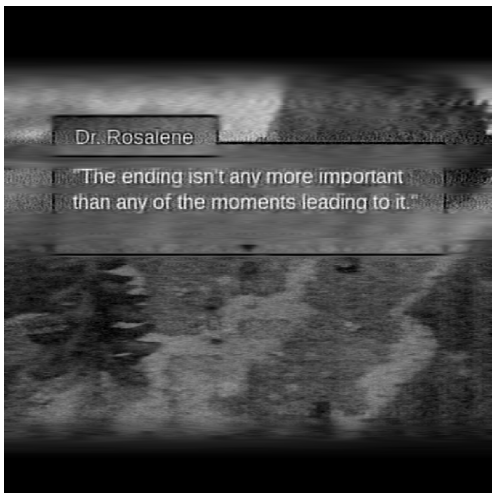
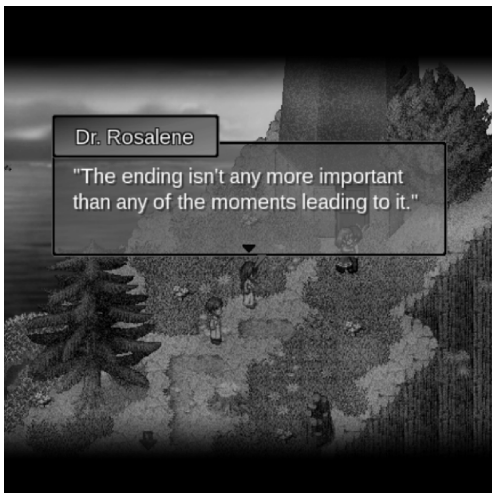


Figura 5.21: *Degrado nella ricostruzione a partire dal 5% dei campioni (DCT)*

Immagine originale in scala di grigi



Ricostruzione a partire dal 5% dei campioni (FFT)

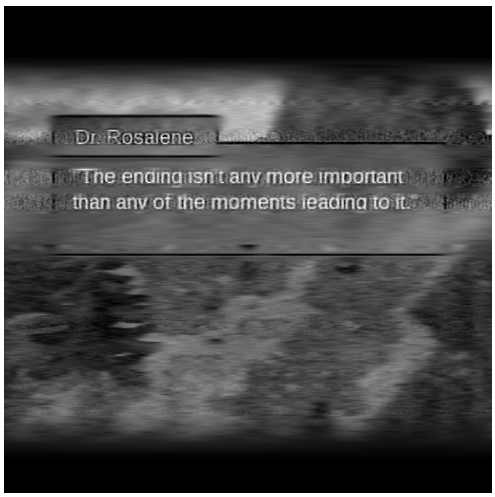


Figura 5.22: *Degrado nella ricostruzione a partire dal 5% dei campioni (FFT)*

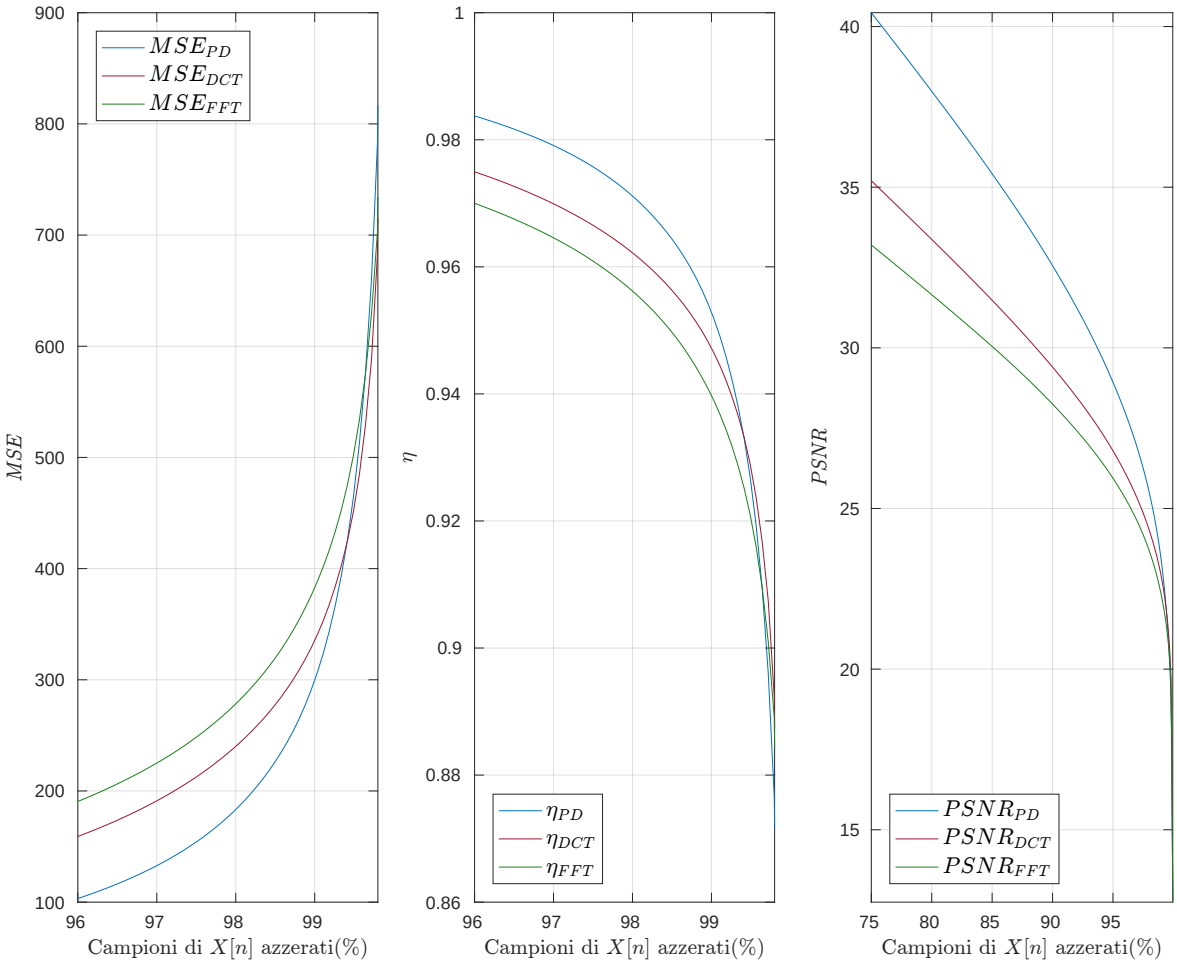


Figura 5.23: *Confrontro tra* MSE, η, PSNR *delle trasformate*

5.2.3 Cameraman

Le seguenti figure (**5.24, 5.25, 5.26**) affiancano all'originale, tratta dal set di immagini standard sopra citato [4], le ricostruzioni con le diverse anti-trasformate a partire dal 5% dei campioni.
Nella **figura 5.27**, invece, viene graficato l'andamento dei parametri di confronto.

Immagine originale in scala di grigi

Immagine ricostruita a partire dal 5% dei campioni (PD)

Figura 5.24: *Degrado nella ricostruzione a partire dal 5% dei campioni (EOT)*

Immagine originale in scala di grigi

Ricostruzione a partire dal 5% dei campioni (DCT)

Figura 5.25: *Degrado nella ricostruzione a partire dal 5% dei campioni (DCT)*

Immagine originale in scala di grigi

Ricostruzione a partire dal 5% dei campioni (FFT)

Figura 5.26: *Degrado nella ricostruzione a partire dal 5% dei campioni (FFT)*

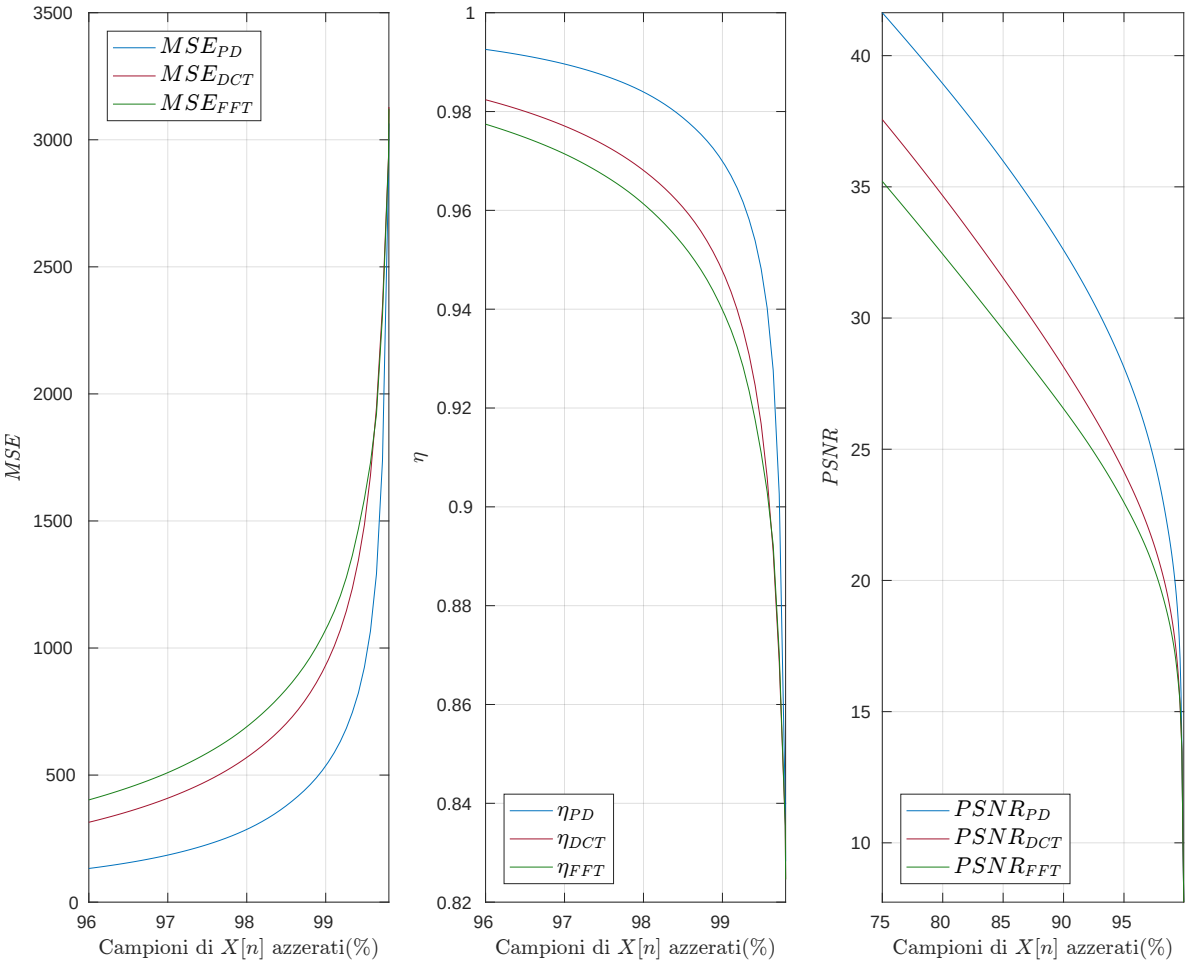


Figura 5.27: *Confrontro tra* MSE, η, PSNR *delle trasformate*

5.2.4 Collage di figure

Le seguenti figure (**5.28, 5.29, 5.30**) affiancano all'originale, presa dal web e ridimensionata ad-hoc, le anti-trasformate a partire dal 5% dei campioni.
Nella **figura 5.31**, invece, viene graficato l'andamento dei parametri di confronto.

Immagine originale in scala di grigi

Immagine ricostruita a partire dal 5% dei campioni (PD)

Figura 5.28: *Degrado nella ricostruzione a partire dal 5% dei campioni (EOT)*

Immagine originale in scala di grigi

Ricostruzione a partire dal 5% dei campioni (DCT)

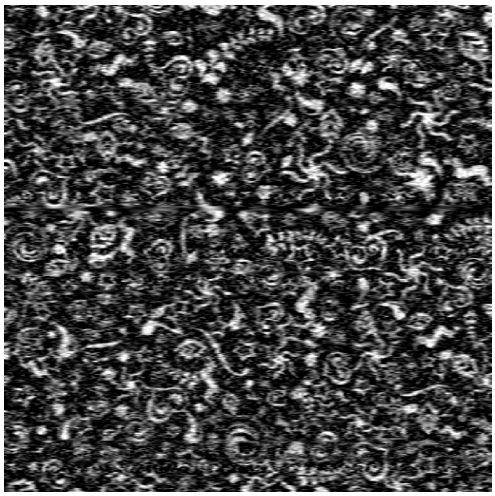


Figura 5.29: *Degrado nella ricostruzione a partire dal 5% dei campioni (DCT)*

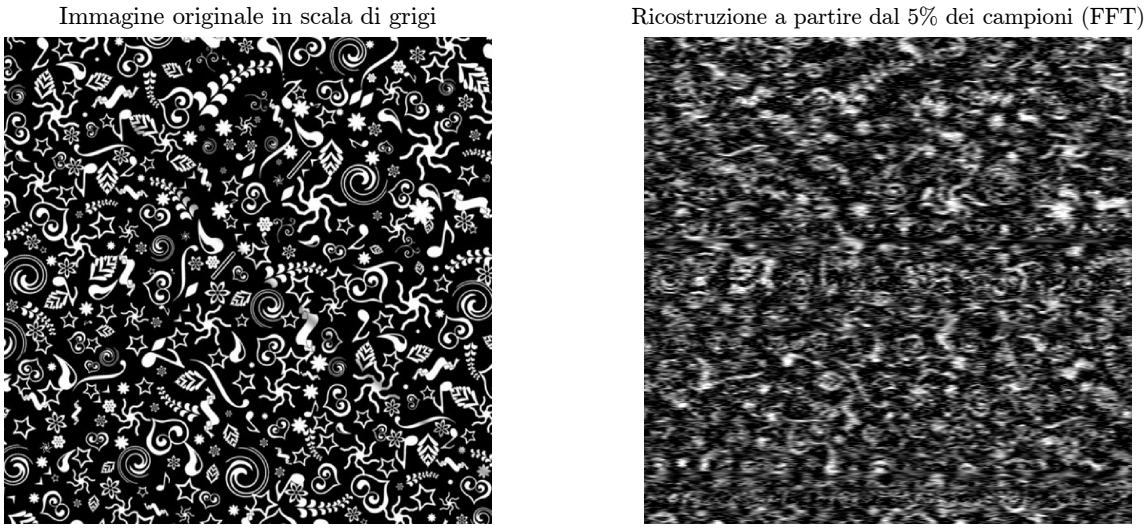


Figura 5.30: *Degrado nella ricostruzione a partire dal 5% dei campioni (FFT)*

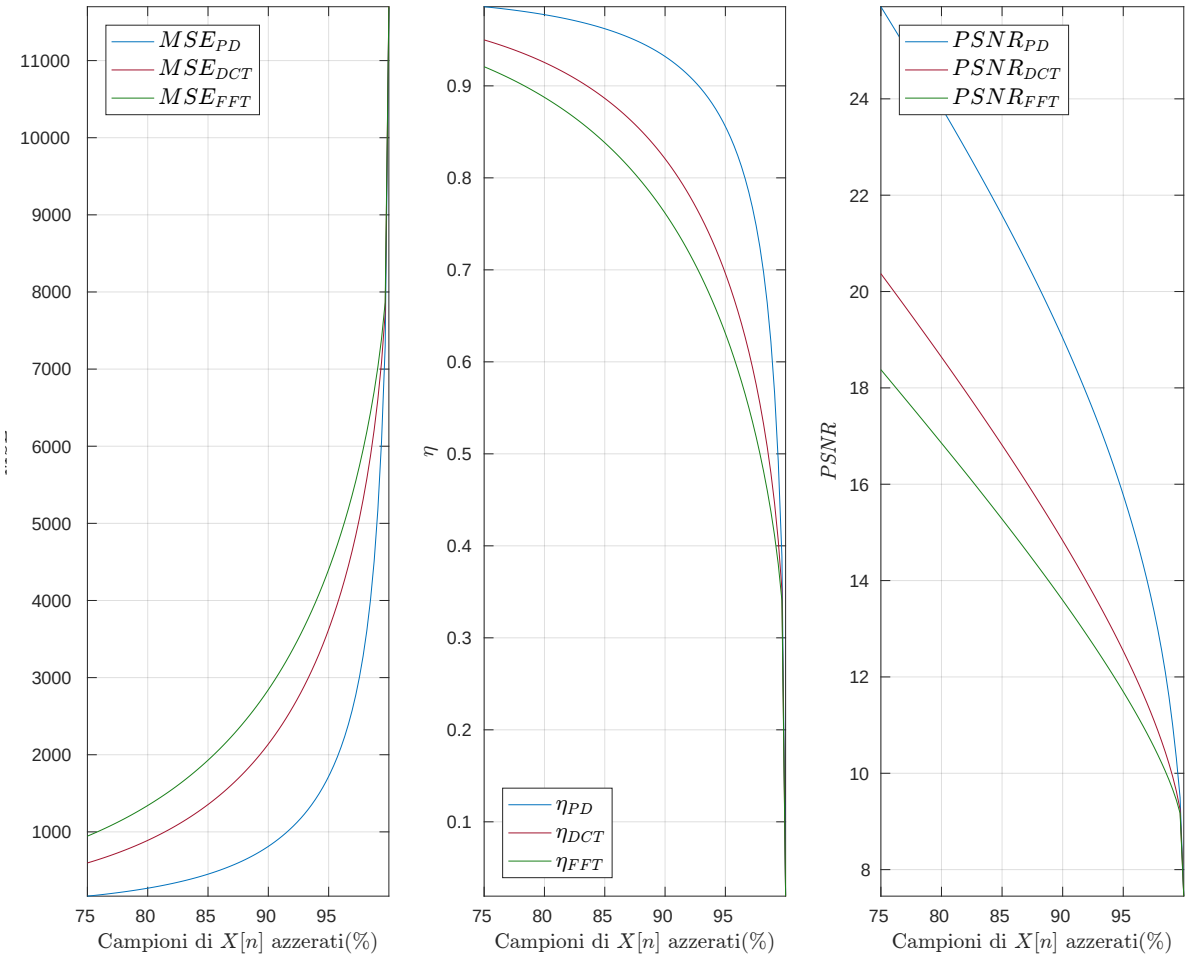


Figura 5.31: *Confrontro tra* MSE, η, PSNR *delle trasformate*

5.2.5 Texture

Le seguenti figure (**5.32, 5.33, 5.34**) affiancano all'originale, presa dal web e ridimensionata ad-hoc, le anti-trasformate a partire dal 5% dei campioni.
Nella **figura 5.35**, invece, viene graficato l'andamento dei parametri di confronto.

Immagine originale in scala di grigi

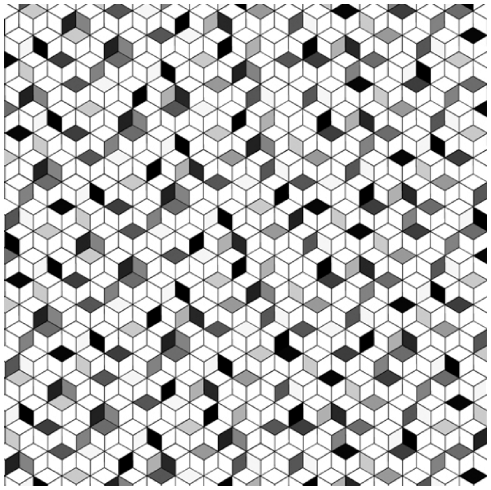


Immagine ricostruita a partire dal 5% dei campioni (PD)

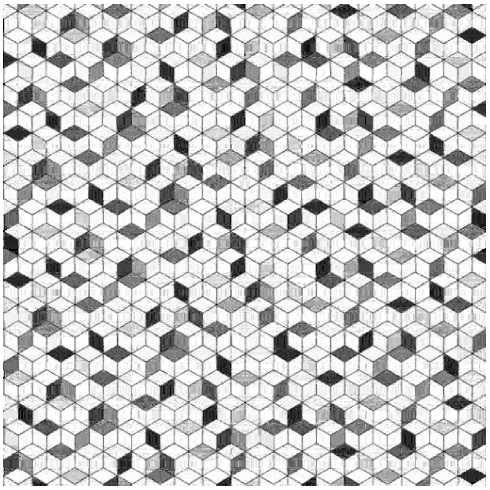
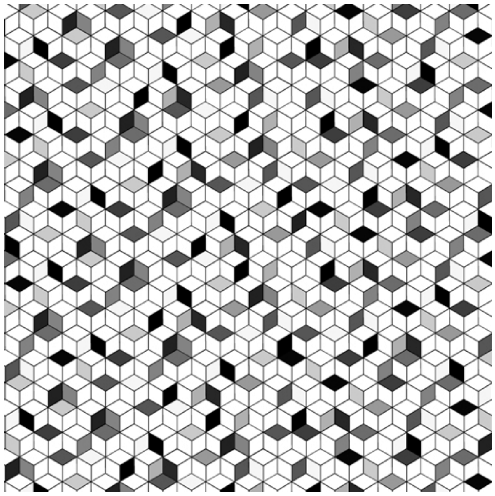


Figura 5.32: *Degrado nella ricostruzione a partire dal 5% dei campioni (EOT)*

Immagine originale in scala di grigi



Ricostruzione a partire dal 5% dei campioni (DCT)

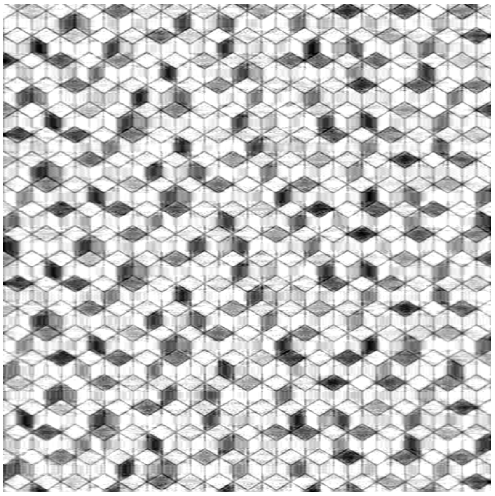


Figura 5.33: *Degrado nella ricostruzione a partire dal 5% dei campioni (DCT)*

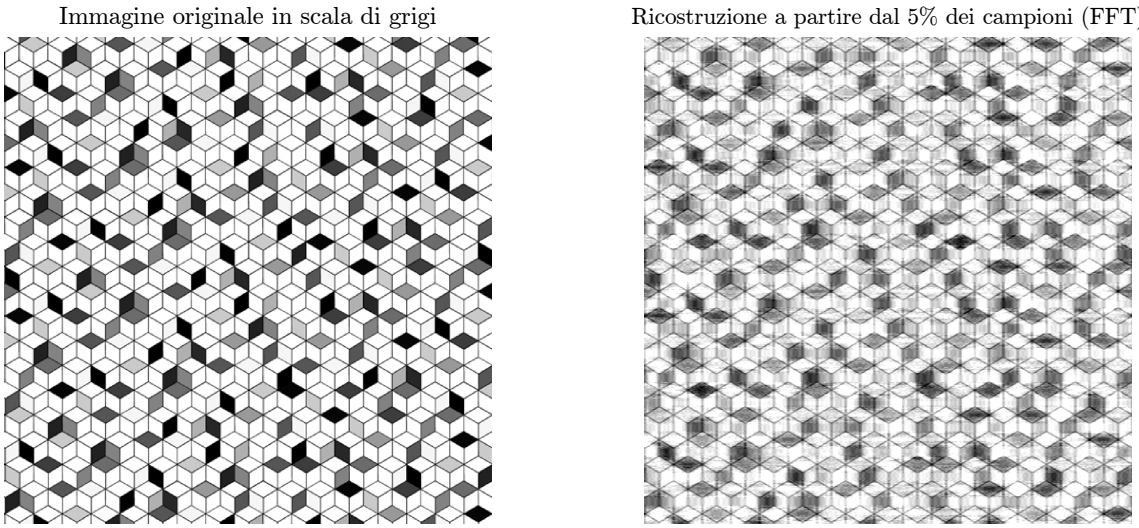


Figura 5.34: *Degrado nella ricostruzione a partire dal 5% dei campioni (FFT)*

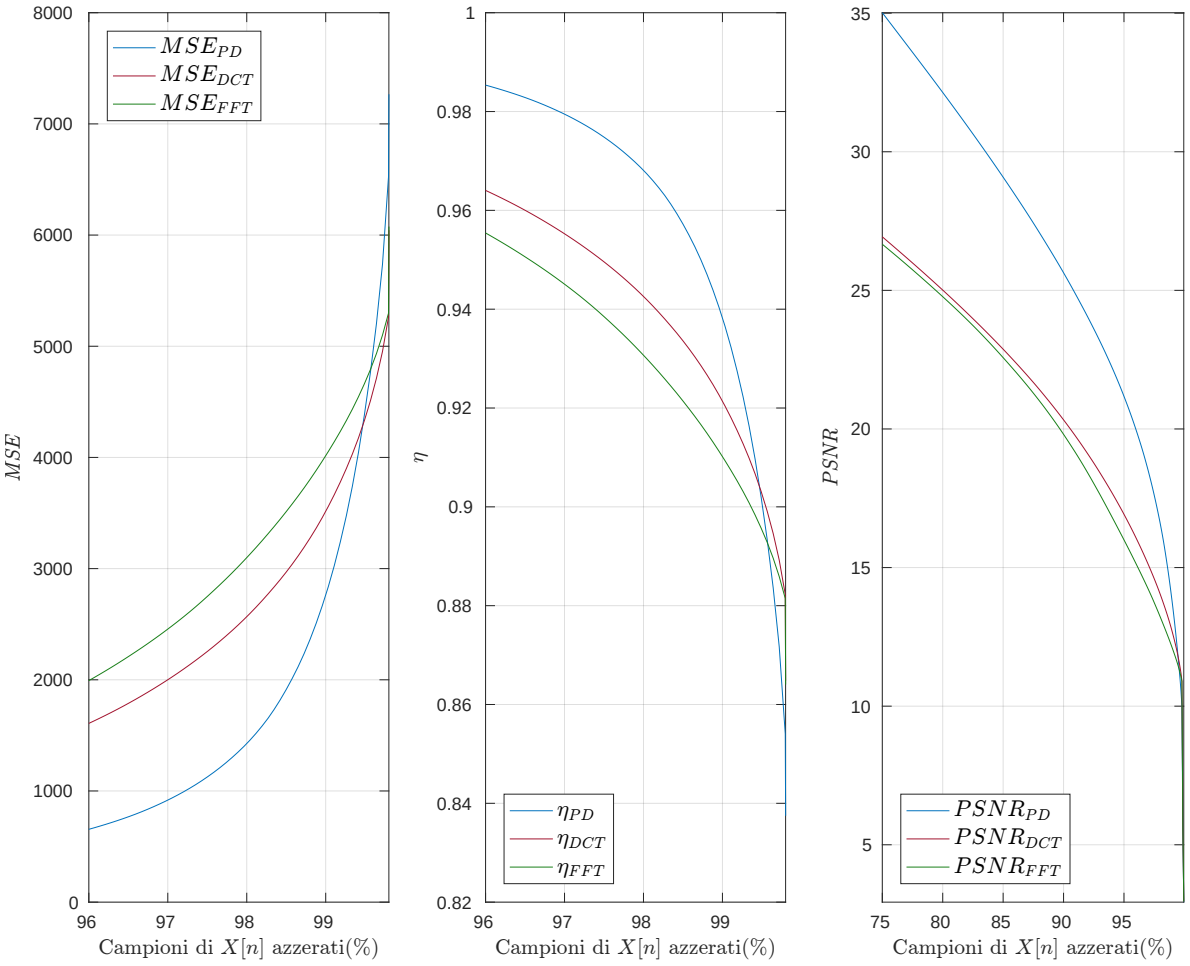


Figura 5.35: *Confrontro tra* MSE, η, PSNR *delle trasformate*

5.3 Impatto del metodo di ordinamento sulla trasformazione di segnali bidimensionali

Come introdotto nel **capitolo 4**, durante il processo di elaborazione di segnali bidimensionali è lecito chiedersi come sia meglio ordinare il vettore delle foglie prima di eseguire l'eliminazione di parte di esse, per poi procedere alla ricostruzione. Mentre compiere l'ordinamento e il successivo taglio riga per riga assicura un trattamento eguale per ognuna di esse, ordinare e tagliare rispetto alla totalità delle foglie apre a nuovi scenari.

È presto dimostrato (si vedano gli **esempi 5 e 6**) come i campioni necessari alla corretta ricostruzione di sequenze con spiccate simmetrie interne siano minori in numero rispetto a quelli necessari per ricostruire una sequenza caratterizzata dall'assenza quasi totale di simmetrie locali. Inoltre, in alcuni casi non troppo rari, si verifica l'eventualità in cui diverse righe risultino essere caratterizzate da una quasi totale costanza nei valori: eclatante è quello riportato nella **figura 5.38** (bande nere in alto e in basso), significativo (e più comune) quello riportato nella **figura 5.40** (prime righe in dall'alto).

È chiaro come mantenere un elevato numero di campioni per la ricostruzione di righe simili possa essere giudicato uno spreco: è intuitivo pensare all'informazione all'interno delle immagini come concentrata attorno a cambi repentini di valore (i.e. contorni di oggetti, volti e parti del corpo ...).
A sostegno di questa posizione, proponiamo degli esempi di elaborazione di parte delle immagini presentate in precedenza adottando i due differenti approcci (per quanto riguarda l'ordinamento). In particolare:

- Nelle immagini che seguono distingueremo i medesimi grazie al ***pedice "abs"*** (da *absolute sorted*), utilizzato per rappresentare la ricostruzione dopo l'ordinamento e il taglio rispetto alla totalità del segnale bidimensionale, e grazie al ***pedice "rw"*** (da *row-wise sorted*), utilizzato al contrario per rappresentare l'approccio per righe;

- Parlando di ricostruzione, per rendere massimamente evidenti le differenze è stato preferito l'utilizzo di immagini ricostruite a partire dal 2% dei campioni del segnale originale (vale a dire una media di 10 campioni su 512 per riga).

È fondamentale riportare il fatto che essendo le immagini originali matrici di dati interi da 8 *bit*, e quindi di valori compresi tra 0 e 255, le ricostruzioni che seguono sono graficate dopo la conversione a *uint8*.
I grafici del PSNR, come vedremo, confermano l'ottimalità dell'approccio ad ordinamento totale: una ricostruzione meno precisa delle righe caratterizzate da forte simmetria per prediligere una ricostruzione più accurata delle righe caratterizzate da forti discontinuità e scarse simmetrie porta ad un MSE minore (e quindi ad un PSNR maggiore, dato che le due grandezze sono legate in maniera inversamente proporzionale).

5.3.1 Lena

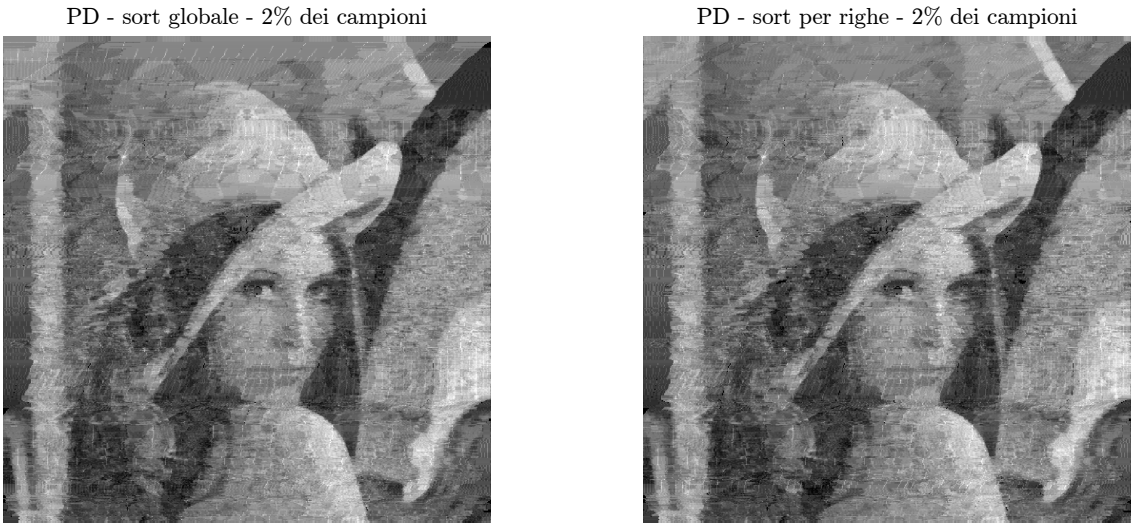


Figura 5.36: *Ricostruzioni a partire dal* 2% *dei campioni, sorting differenti (uint8)*

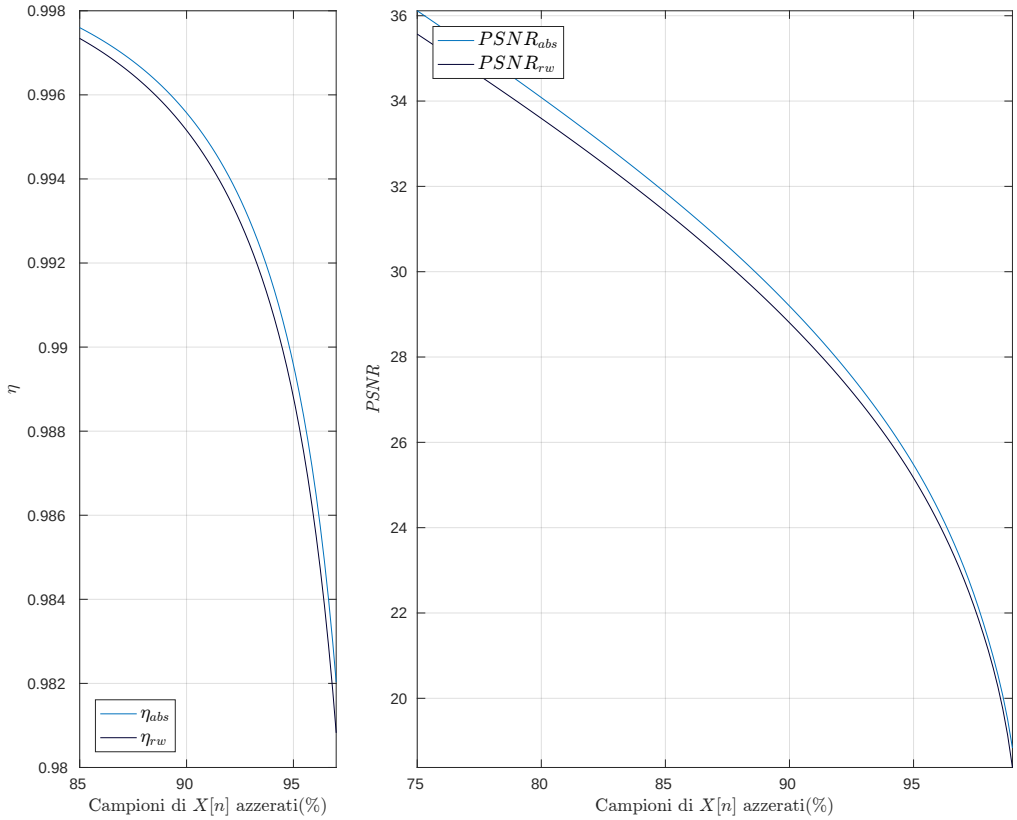


Figura 5.37: η *e* PSNR *al variare del metodo di sorting*

5.3.2 To The Moon®

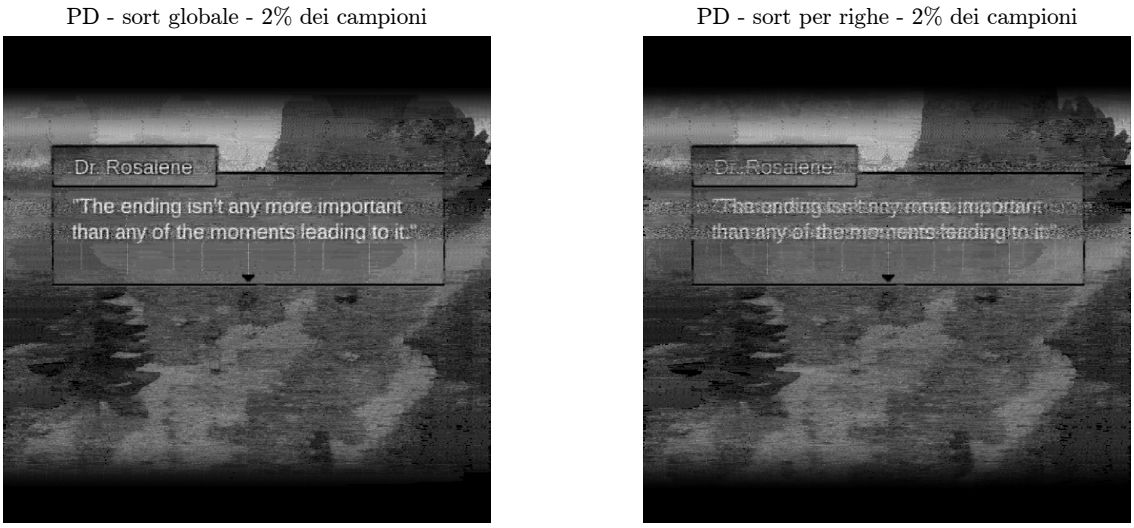


Figura 5.38: *Ricostruzioni a partire dal 2% dei campioni, sorting differenti (uint8)*

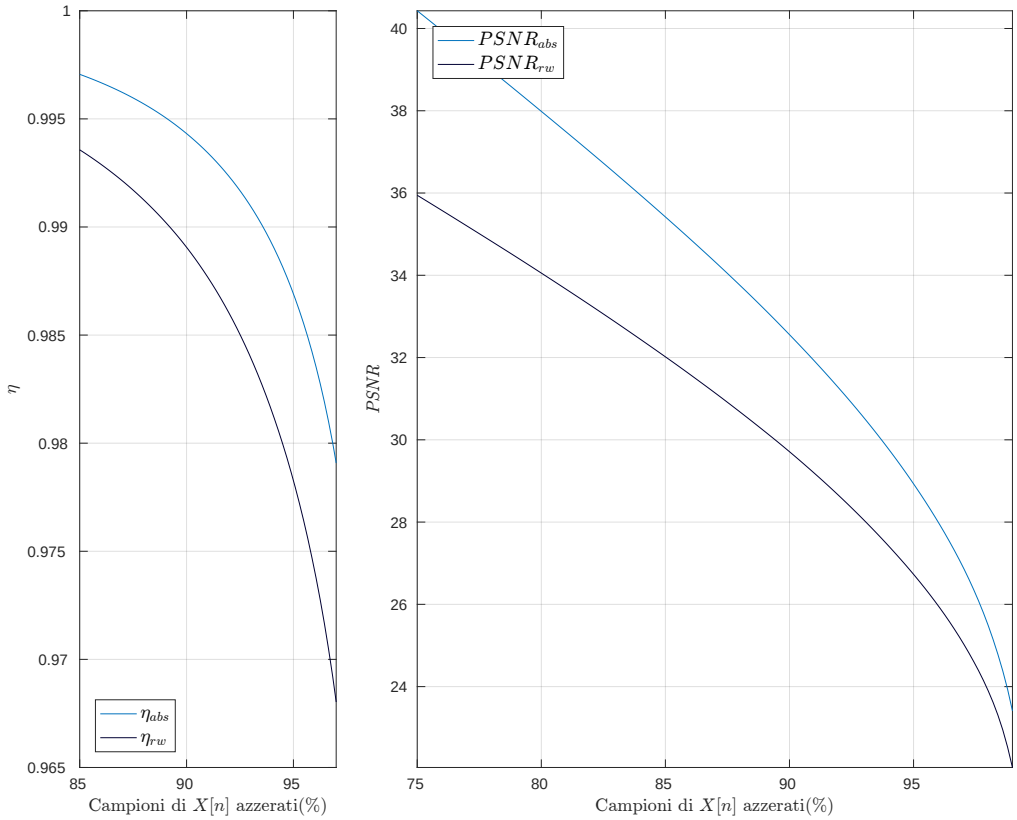


Figura 5.39: η *e* PSNR *al variare del metodo di sorting*

5.3.3 Cameraman

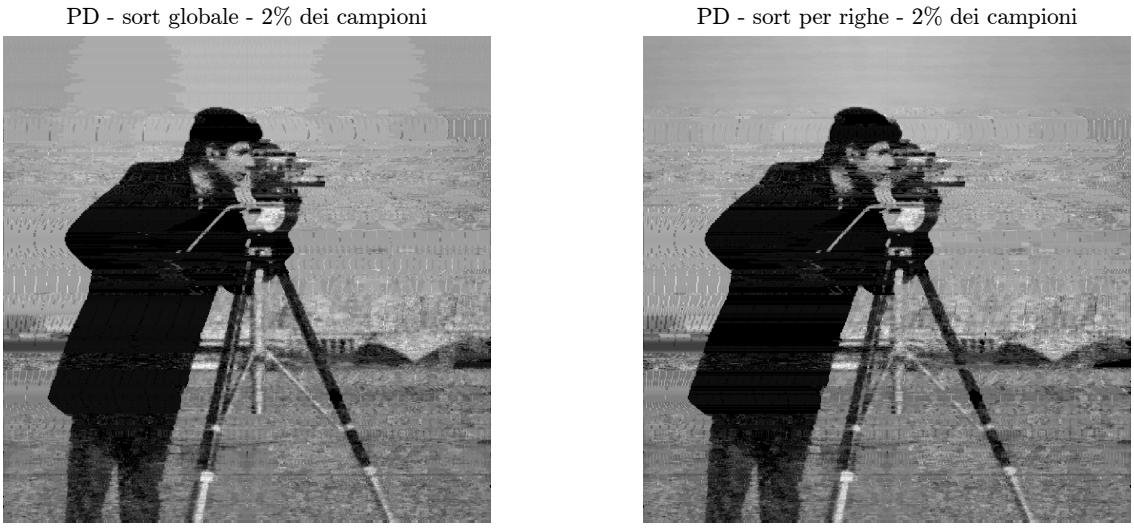


Figura 5.40: *Ricostruzioni a partire dal 2% dei campioni, sorting differenti (uint8)*

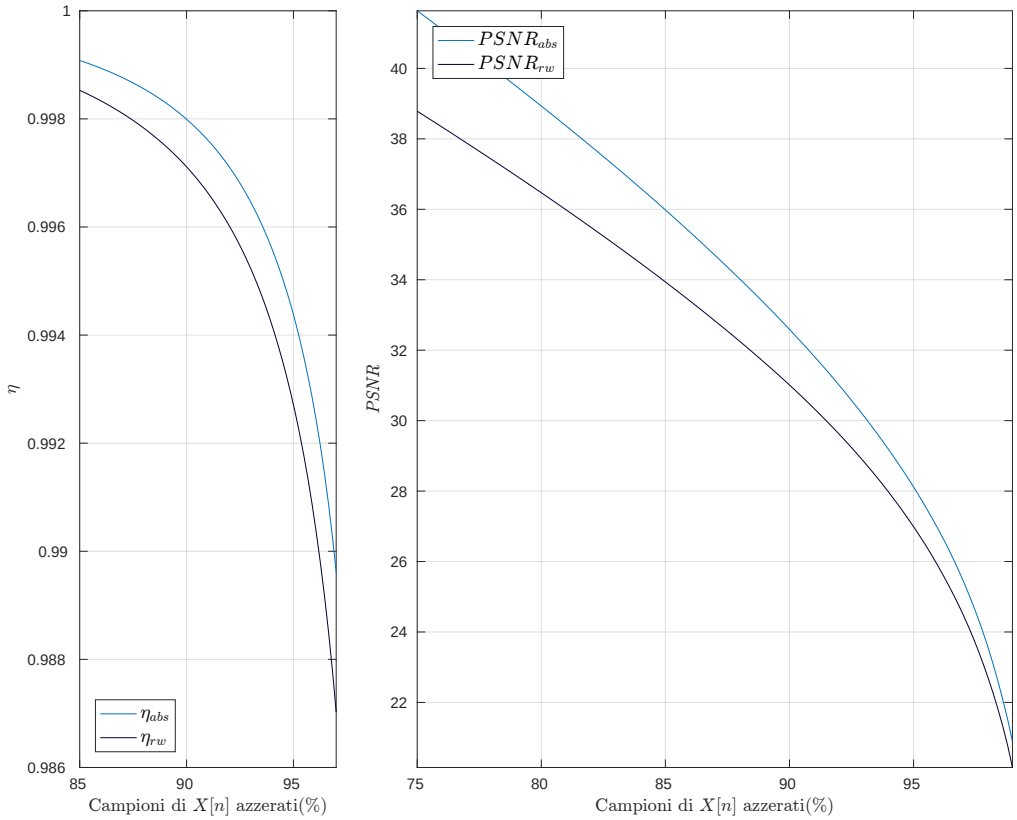


Figura 5.41: η *e* PSNR *al variare del metodo di sorting*

5.3.4 Texture

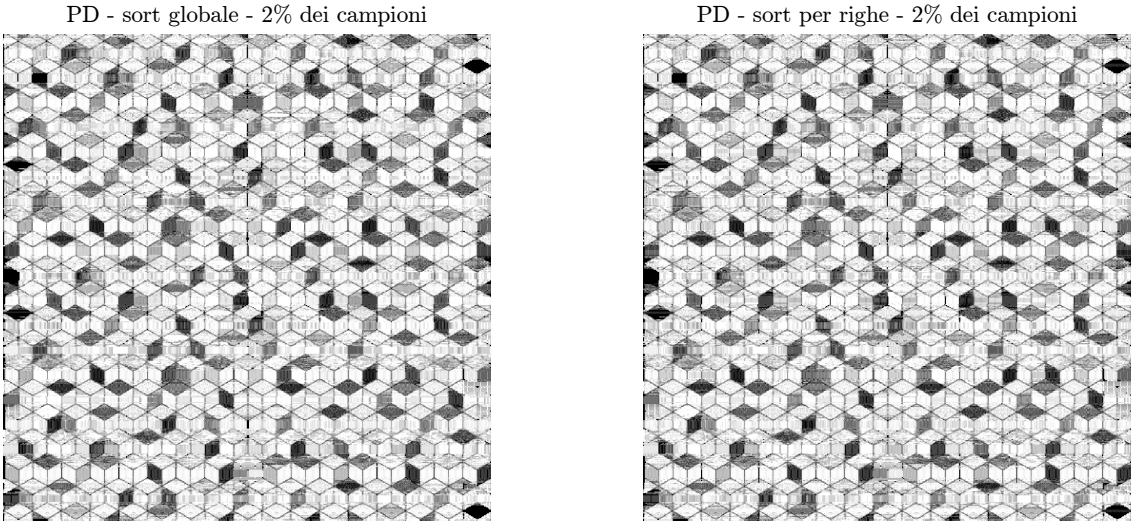


Figura 5.42: *Ricostruzioni a partire dal* 2% *dei campioni, sorting differenti (uint8)*

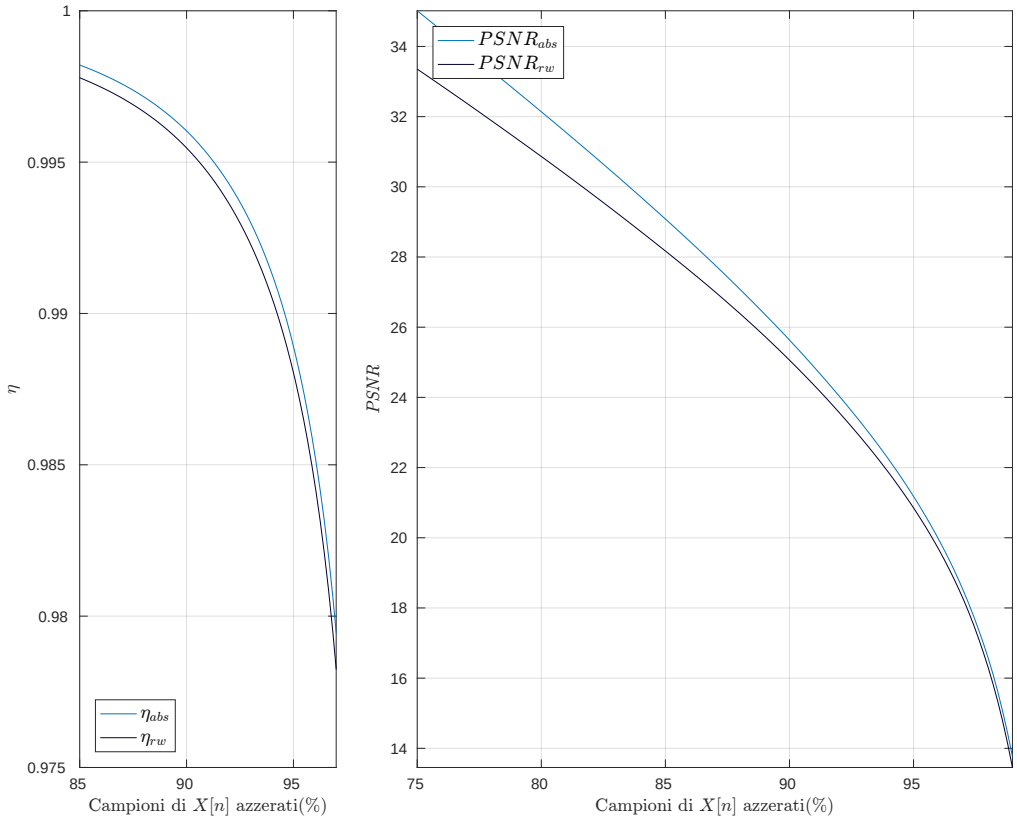


Figura 5.43: η *e* PSNR *al variare del metodo di sorting*

Conclusioni

In questo documento abbiamo presentato, dopo una breve parentesi teorica, la nostra implementazione dell'algoritmo che porta al calcolo della trasformata *PD*.

Come evidenziato dai risultati raccolti ed esposti nel **capitolo 5**, la sua applicazione su segnali monodimensionali di qualsiasi tipo rivela le sue eccellenti prestazioni: nella maggior parte dei casi, supera trasformate dotate di un grande potere compattante, quali *DCT* e *FFT*.

Si consideri anche come i risultati già ottimi in termini di prestazioni presentino dei margini di miglioramento non indifferenti. In particolare, incrementi nell'efficienza in campo monodimensionale potrebbero essere raggiunti a partire da alcune idee presentate nel corso del lavoro ma non sviluppate o implementate, come:

- L'implementazione di una routine di backtracking per la ricerca dell'asse di simmetria ottimo (n_0) in presenza di più candidati dal valore identico;
- L'applicazione di tecniche di pre elaborazione: dalle più semplici, quali la sottrazione della media, fino alle più sofisticate e ricercate, come per esempio specifici filtraggi ad-hoc in base alla tipologia di segnale.

Per quanto riguarda l'applicazione della trasformata a segnali bidimensionali, si ricorda al lettore che essa è per ora eseguita iterando la semplice trasformazione monodimensionale su righe o su colonne. Nonostante il grosso lavoro ancora da intraprendere in tal senso (nella ricerca di una trasformata che lavori nativamente su sequenze bidimensionali), un'idea per migliorare l'algoritmo presentato potrebbe consistere nel dotarlo della capacità di *"decidere"* l'orientamento dell'elaborazione (i.e. per righe o per colonne), in base all'immagine presa in considerazione;

Bibliografia

[1] A. Oppenheim, A. Schafer, and C. Jones,
Discrete-time signal processing, Prentice Hall, 1999.

[2] F. Guerrini, R. Leonardi, and A. Gnutti,
Representation of signals by local symmetry decomposition,
European Signal Processing Conference (EUSIPCO) 2015.

[3] PhysioBank Databases, PhysioNet,
`https://www.physionet.org/physiobank/database/`

[4] "Standard" digital image processing test images package, ImageProcessingPlace.com,
`http://www.imageprocessingplace.com/root_files_V3/image_databases.htm`

[5] Rangaraj M. Rangayyan, *Biomedical signal analysis: a case-study approach 2nd ed., IEEE Press, Wiley-Interscience, 2002*