
UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Dipartimento di

Ingegneria dell'Informazione

Corso di Laurea in

Ingegneria dell'Informazione

(Laurea triennale DM 270/04 – indirizzo Ingegneria)


# Compressione audio senza perdite – Teoria e pratica


Relatore:
*Prof. Giancarlo Calvagno*

Laureando:
*Samuele Contessa*



Anno accademico
*2020/2021*

Data di laurea
*22/11/2021*

# Indice

# Introduzione

Con l'avvento dell'elettronica digitale è diventato sempre più agevole ed affidabile l'uso dei segnali digitalizzati al posto di quelli analogici, poiché questo nuovo formato permette una manipolazione e distribuzione dei dati estremamente semplificata. È stato quindi reso possibile il trasferimento e la velocizzazione di una mole enorme di servizi all'interno delle apparecchiature intelligenti.

L'utilizzo più vasto di questo "salto quantico", da parte dell'utente comune, riguarda la conversione dei segnali audio e video in digitale. Vi è stata infatti una diffusione esponenziale dei servizi di streaming audio e video mediante la rete Internet, la riduzione dei costi per la produzione di unità di memoria portatili (CD, chiavette USB) e la loro conseguente diffusione capillare, la manipolazione interamente software di tutti i tipi di media (al punto che chiunque sia in possesso di un personal computer è in grado di creare album musicali, effetti speciali filmografici, grafiche pubblicitarie, ecc…).

Di contro a questo incredibile aumento di traffico dati, nasce il problema della capacità di trasmettere questa mole di informazioni. Le soluzioni sono sostanzialmente due e per ottenere il massimo delle prestazioni è necessario implementarle entrambe: si tratta dell'aumento della capacità di trasmissione e della riduzione della quantità di dati da trasmettere.

In questo progetto ci siamo occupati del secondo ambito, focalizzandoci su un tipo di codifica dei segnali audio digitalizzati con lo scopo di comprimerne le dimensioni senza inficiare sull'autenticità delle informazioni contenute, ovvero la tecnica di compressione audio senza perdite.[1]

Dal quadro generale dell'ambito appena presentato ci si addentrerà poi nello specifico, avvalendosi ulteriormente di un programma realizzato con il software MATLAB al fine di verificare l'efficienza di una compressione senza perdite secondo metodologie trattate.

[1] In inglese "lossless".

# 1. Tipologie di algoritmi

## 1.1 Differenza tra “senza perdite” e “con perdite”

Abbiamo parlato di compressione “senza perdite”, chiarendo che si tratta di un tipo di metodologia che mantiene intatta la veridicità delle informazioni, ma cosa intendiamo nel concreto? E che differenza c’è nella compressione “con perdite”?

Nella tecnica lossy[2] è accettata una distorsione del segnale di partenza successiva alla codifica, giustificata dalla drastica riduzione della quantità di dati, si tratta di rapporti 12:1, contro il massimo usuale di 3:1 delle compressioni lossless. Un miglioramento 4 volte superiore a scapito della perdita di qualità (la quale può essere adeguatamente bilanciata in base al servizio richiesto).
A questo punto la domanda sorge spontanea: per quale ambito risulta ancora utile la compressione lossless?
Questo tipo di compressione che non subisce alcuna degradazione sul segnale di partenza, rappresenta il metodo per ottenere il massimo della qualità, necessaria per la rielaborazione informatica dei segnali, ad esempio all’interno di case discografiche, o per donare la miglior esperienza all’utente finale. La codifica lossless sarà quindi sempre di sfondo per il grande pubblico, ma resterà una componente fondamentale per le nicchie che lo richiedono o necessitano.

## 1.2 Formati ed algoritmi

Dobbiamo innanzitutto fare una distinzione tra i formati dei file in cui vengono salvati i segnali digitalizzati compressi e gli algoritmi utilizzati a tal fine chiamati *codec,*[3] nonostante quest’ultimi solitamente diano il nome alle estensioni;[4] è infatti possibile codificare con un algoritmo diverso dal tipo di formato (anche se nella pratica non troviamo quasi mai quest’eventualità, se non per applicazioni specifiche). Forniamo quindi un elenco dei principali codec e formati utilizzati.

[2] Dall’inglese “con perdite”.
[3] Dall’inglese “codificatore/decodificatore”.
[4] Le estensioni dichiarano in breve il formato del file.

### 1.2.1 Con perdite - lossy

- **MP3:** *Moving Picture Expert Group-1/2 Audio Layer 3*, noto anche come *MPEG-1 Audio Layer III* o *MPEG-2 Audio Layer III*. Codec standard più utilizzato per il suo ottimo rapporto tra qualità e compressione, pubblicato nel 1998 come apripista, offre una qualità tradotta in quantità di informazione tra 32 e 320 Kbit/s con una compressione fino a 13:1.
- **AAC:** *Advanced Audio Coding* creato da MPEG (Moving Picture Experts Group),[5] ma di proprietà Apple, molto simile alla tecnica dell'MP3 in termini di compressione, ma con qualità superiore.
- **WMA:** *Windows Media Audio*, creato da Microsoft, molto meno supportato dell'MP3, meno efficiente in termini di compressione, migliore nella qualità; per un'ottima definizione audio (CD quality) si realizza una compressione 4:1.
- **Ogg Vorbis:** formato open source non molto diffuso, nonostante offra un'ottima qualità, paragonabile all'MP3, ma con il bisogno di un bitrate minore, motivo per cui è ampiamente sfruttato in ambito tecnico, nonché da Spotify.[6]
- **DTS:** *Digital Theater System*, come suggerisce il nome è uno dei formati utilizzati per il cinema o anche per il broadcasting[7] televisivo; la sua compressione è di 6,5:1 pur mantenendo una buonissima qualità.

### 1.2.2 Senza perdite - lossless

- **WAV:** *WAVEform audio file format*, è un derivato del gruppo *RIFF (Resource Interchange File Format),*[8] quindi un puro contenitore, un modo di salvare i dati, non un algoritmo di compressione di per sé; tuttavia è resa la possibilità di incapsulare diversi tipi di codec al suo interno (ad esempio MP3 o diversi tipi di PCM[9]). Può dunque arrivare a ridurre molto la grandezza del file, a differenza del codec implementato. Nasce nel 1991, ancor oggi lo standard più diffuso per i CD musicali nella versione con 1,4 Mbit/s nella musica stereo e 16 bit per campione

[5] Comitato tecnico internazionale che fornisce gli standard negli algoritmi di compressione.
[6] Azienda svedese molto affermata che offre streaming audio di alta qualità a pagamento.
[7] Dall'inglese "trasmissione verso tutti i dispositivi in ascolto".
[8] Standard per modelli di contenimento dei dati suddivisi in "chunks" (frammenti).
[9] Pulse Code Modulation, è la modulazione del codice a impulsi ovvero un tipo di conversione da analogico a digitale, la quale offrendo diverse varianti, può generare notevoli gradi di compressione, come nel caso di ADPCM (Adaptive Differential PCM).

(frequenza di campionamento stereo 44'100 Hz) dove 1 minuto di musica occupa 10,7 MB.

- **AIFF:** *Audio Interchange File Format*, molto simile al formato WAV, sviluppato da Apple sulla base di un progetto di Electronic Arts.[10]
- **FLAC:** *Free Lossless Audio Codec*, open source come si può facilmente dedurre, creato nel 2001 per l'archiviazione nel lungo periodo in CD o PC, non differisce in qualità rispetto ai due precedenti e offre una compressione media di circa 2:1.
- **ALAC:** *Apple Lossless Audio Codec*, controparte Apple di FLAC, compressione leggermente inferiore rispetto al FLAC, ma meno diffuso.
- **APE:** *Monkey's Audio*, preferito dagli audiofili, supera leggermente la qualità del FLAC a parità di dimensioni del file, è inoltre ampiamente supportato su sistemi Windows.
- **OFR:** *OptimFROG*, codificatore gratuito sviluppato da Florin Ghido,[11] con il miglior tasso di compressione lossless che varia da 1.6:1 (musica rock "caotica") a 4:1 (musica classica melodica).

[10] Un'azienda che si occupa della creazione di videogiochi.
[11] Ricercatore e Professore associato presso l'Università di Tampere, Finlandia. Specializzato in compressione audio lossless.

# 2. Metodologia

Tutte le tecniche di compressione audio lossless, o quasi, hanno alla base il metodo che andremo ad analizzare, motivo per cui risulta conveniente descriverlo per primo e successivamente approfondire eventuali differenze tra i formati.

Al centro della riduzione del quantitativo in bit di un file audio vi è la conversione di notazione con cui vengono rappresentati i valori del segnale reso discreto: si utilizza un algoritmo di predizione per dividere la parte prevedibile da quella imprevedibile del segnale secondo le regole adottate, è naturale che maggiore sarà la complessità dell'algoritmo, maggiore sarà la capacità di predizione (nonché il tempo necessario ad eseguirlo). Si ricerca quindi il maggior numero di costanti ricorrenti nel file audio (al fine di eliminarne la ridondanza) e contemporaneamente si tenta di minimizzare la parte imprevedibile per poterla codificare col minimo quantitativo informazionale possibile. In termini più specifici chiamiamo *errore* la differenza tra la predizione ed il segnale reale: più piccolo sarà l'errore, minore sarà la quantità di dati da dover inviare. Ecco il motivo per cui per segnali ripetitivi (pieni di costanti ripetute) la compressione risulta molto grande.[12] Illustriamo uno schema per il calcolo dell'errore in figura 2.1.

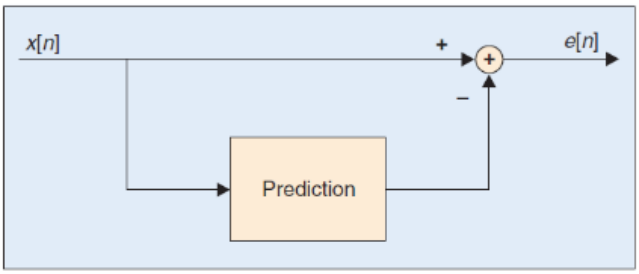


***Figura 2.1:*** *Schema riassuntivo per il calcolo e la trasmissione dell'errore (e[n]) a partire dal segnale originale segmentato (x[n]).*

Per gli studiosi di questo ambito nasce quindi il problema di sviluppare il miglior algoritmo predittivo, anche capace di modificarsi in base al segnale da analizzare, che sia però al contempo non troppo complesso, in modo da poter essere eseguito in tempi che giustificano il guadagno in termini di efficienza nella compressione. Tale complessità viene scelta in base alla potenza computazionale disponibile.[13]

---

[12] Ad esempio con l'algoritmo OptimFROG nella musica classica (armonica con diverse parti ripetute) possiamo ottenere una compressione 4:1 lossless.

[13] Una codifica che richiede molto tempo oggi, non si sarebbe potuta implementare ad esempio 40 anni fa, poiché avrebbe richiesto un multiplo di quel tempo, magari già notevole, sia in compressione che in decompressione.

Sarà poi possibile interessarsi alla codifica simbolica[14] migliore per la trasmissione dell'errore al fine di diminuire ulteriormente il quantitativo di bit necessario, nonché trovare l'unione più adatta tra tipo di algoritmo e codifica dell'errore che porterà alla performance più elevata per ogni diverso ambito di applicazione.

[14] La mappa biettiva tra valori del segnale e rappresentazione in memoria.

# 3. Modelli classici di codifica lossless

Nella "digitalizzazione" si prendono dei campioni ad intervalli di tempo regolari $T$, da cui possiamo ricavare la frequenza di campionamento $F = \frac{1}{T}$, utilizzando un numero discreto di livelli $L = 2^B$, dove $B$ è il numero di bit usati per rappresentare il campione. Sappiamo che un file musicale è usualmente dotato di due canali[15] quindi denotiamo i campioni del canale sinistro con $\{ x_0 , x_1 , \dots , x_{N-1} \}$ e quelli del destro con $\{ y_0 , y_1 , \dots , y_{N-1} \}$, mentre $N$ rappresenta la lunghezza dei vettori di campioni in un *frame,*[16] identica tra i canali.

## 3.1 Pre-processing

Prima di calcolare la predizione secondo uno degli algoritmi predittivi è possibile rimuovere delle informazioni superflue dal segnale digitalizzato: trattasi di una trasformazione biiettiva al fine di eliminare i bit inutilizzati meno significativi, ridurre l'intervallo di valori campionati al minimo e scalare i valori dei campioni in modo da poterli rappresentare con meno informazione.

### 3.1.1 Rimozione dei bit inutilizzati

Può accadere che il convertitore ADC (Analog-to-Digital Converter)[17] abbia una risoluzione, oppure un range operativo minore, del formato utilizzato per codificare il file digitalizzato del segnale originale. Alcuni bit salvati saranno quindi del tutto inutili e possono essere rimossi senza alcuna perdita di informazione nel modo seguente.
Utilizziamo lo pseudocodice per indicare le operazioni:

```
Set mask = 0;
For i = 0 to N-1 {
    mask = mask bitor xi
}
Set unused = "numero di bit nulli meno significativi in mask";
```

[15] Segnale audio "stereo".
[16] I campioni vengono suddivisi in blocchi (frame) per limitare la perdita di dati nel caso di una trasmissione rumorosa.
[17] Convertitore da segnale analogico a digitale. Contiene un campionatore che assegna ad ogni valore analogico, ad ogni intervallo prefissato, un simbolo discreto.

In questo modo otteniamo il numero di bit inutilizzati tra tutti i valori del segnale almeno all'interno di un frame, da eliminare poi nella rappresentazione binaria in memoria del nostro file compresso; ad esempio si potrebbe passare da una rappresentazione a 16 bit per valore, a una a 12 bit. Avendo eliminato 4 bit per ogni numero, il 25% delle cifre binarie, otterremmo già un certo grado di compressione.

### 3.1.2 Scala dei valori

Con un metodo più avanzato, sempre con l'obiettivo di ridurre al minimo l'ampiezza dei campioni e ottenere dunque un intervallo di valori minimo, vogliamo trovare due interi non negativi, α e β, con $\alpha \geq 2$ e $\beta < \alpha$ tali che $x_i = \alpha z_i + \beta$, con $\underline{z}$ sequenza di interi ($z_i = \frac{x_i - \beta}{\alpha}$), al fine di scalare i valori campionati $x_i$ e poter eliminare ulteriori bit nella loro rappresentazione in memoria, ovviamente salvando come informazione addizionale α e β. Ciò potrebbe e dovrebbe essere effettuato prima della rimozione di bit inutilizzati descritta nel paragrafo precedente, applicando l'algoritmo sul vettore di $z_i$, il quale verrà successivamente salvato nel file compresso.

Descriviamo il calcolo di questi coefficienti con lo pseudocodice seguente:

```
Set α = x1 – x0;
For i = 0 to N-1 {
    α = massimoComunDivisore(α, xi – x0);
}
Set β = (x1 – x0) modulo α
```

Le operazioni alla base di questo processo si possono schematizzare con la figura 3.1.

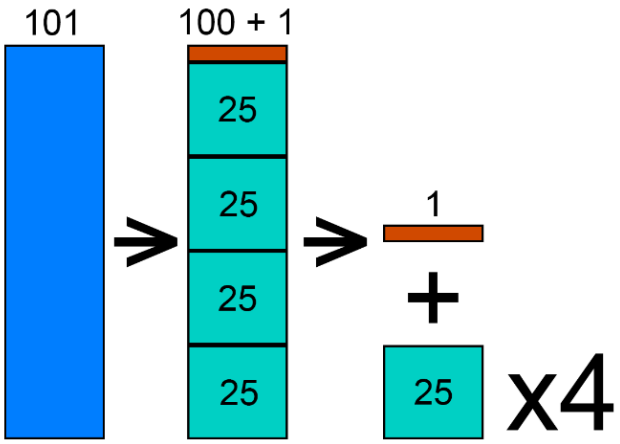


***Figura 3.1:*** *a titolo puramente esemplificativo il valore 101 viene codificato come 25x4 + 1 poiché, per un array di numeri rappresentato in questo modo, dove il resto 1 (= β) e il fattore moltiplicativo 4 (= α) rimangono costanti e la quantità moltiplicata varia, si ottiene un significativo guadagno nello spazio in memoria utilizzato.*

### 3.1.3 Gestione della virgola mobile

Possiamo applicare i metodi sopra indicati solo se siamo di fronte ad un tipo codifica in cui i valori segnale audio discreto siano rappresentati da numeri interi (senza cifre dopo la virgola), ma nulla ci impedisce di trasformare la notazione in virgola mobile[18] in quella intera tramite un'approssimazione ragionevolmente ponderata al fine di perdere il minimo quantitativo informazionale. Ad esempio moltiplicare i valori in modulo per 1000 o 10000, per poi arrotondarli a nessuna cifra decimale, offre già una buonissima approssimazione udibile solo all'orecchio di un esperto. Successivamente se si volesse mantenere la notazione di partenza nel file compresso, basterà effettuare le operazioni inverse per riottenere i valori in floating-point (seppur con minima approssimazione).
Al fine di chiarire ogni dubbio su tale metodo approfondiamo la notazione in virgola mobile maggiormente utilizzata nella compressione audio: il formato floating-point standard IEEE[19] a 32 bit, la quale presenta:

- 1 bit segno: $s$
- 8 bit esponente: $e$
- 23 bit mantissa: $m$

Abbiamo quindi un'altissima precisione dovuta ai 23 bit di mantissa dedicati alle cifre significative che possono raggiungere un'ampiezza descritta di $6.8 \cdot 10^{38}$ (mai utilizzata completamente nei file audio), assieme ai bit dell'esponente, contando sia i valori positivi che negativi. Ciò viene realizzato per applicazioni specifiche, come l'audio editing professionale, al fine di non perdere in qualità nelle trasformazioni applicate al segnale audio.[20]

## 3.2 Algoritmo predittivo

Ora possiamo concentrarci sullo studio della predizione al fine di diminuire l'entropia, ovvero l'informazione della variabilità del segnale di partenza, potendo mantenere quindi solo sequenze di bit non ripetitive secondo il nostro schema.[21]

[18] "Floating-point" in inglese.
[19] Institute of Electrical and Electronics Engineers, un'associazione internazionale con l'obiettivo della promozione delle scienze tecnologiche; rilascia gli standard per miriadi di prodotti.
[20] Se vi è più precisione ovvero meno distanza dal valore vero del segnale nei campioni, allora manipolando i valori salvati ci distanzieremo ancora meno dalla medesima trasformazione applicata idealmente al segnale originale.
[21] Tramite la definizione dell'algoritmo di predizione "decidiamo" quali dati sono prevedibili e quali no.

### 3.2.1 Predittori lineari fissi

I predittori più semplici sono i "polinomiali fissi" (lineari) ove i coefficienti non cambiano in ogni frame del file; allora definiamo i primi tre ordini nel modo seguente:

1° ordine: $\hat{x}_k = x_{k-1}$

2° ordine: $\hat{x}_k = 2x_{k-1} - x_{k-2}$

3° ordine: $\hat{x}_k = 3x_{k-1} - 3x_{k-2} + x_{k-3}$

Dove il simbolo "$\hat{}$" sta ad indicare il valore così predetto ed il grado dell'ordine derivante dal numero di campioni indietro nel "passato" rispetto al valore da predire ($\hat{x}_k$), nonché il fattore moltiplicativo del campione più "vicino al presente". La scelta dei coefficienti è basata sul peso da dare ai valori, deciso sulla "distanza nel tempo" dal campione predetto: i dati più vicini al "presente" dunque saranno pesati maggiormente ed in relazione all'ordine. Inoltre si può facilmente notare che sostituendo ad ogni $x_{k-t}$ , per *t = 1, 2, 3*, uno stesso valore, ad esempio $x_{k-t} = r$, otteniamo per ogni ordine $\hat{x}_k = r$, ovvero se un segnale è stato costante per gli ultimi campioni, la sua predizione del futuro campione sarà uguale al valore di quelli precedenti. A questo punto si prosegue nel calcolo della differenza tra il segnale campionato originale e la sua predizione, ottenendo così il "segnale" dei residui o errori.

$$e_k = x_k - \hat{x}_k \quad per\ 0 \leq k < N \tag{3.1}$$

Risulta chiaro a primo acchito che non possiamo predire i primi *g* campioni se *g* indica l'ordine del predittore utilizzato. Verrà allora generato un vettore $\underline{e}$ con le prime *g* componenti identiche rispetto al segnale originale e solo per le altre si potrà operare il calcolo di predizione.

Dunque nel file compresso si dovrà salvare solo l'informazione relativa all'errore e ai coefficienti di predizione, tramite una codifica efficiente che discuteremo successivamente. Al decodificatore basterà quindi riunire le informazioni così divise tramite la somma (3.2) per riottenere il segnale originale, dopo aver ricostruito la predizione ripercorrendo le operazioni del codificatore all'inverso.

$$x_k = \hat{x}_k + e_k \tag{3.2}$$

### 3.2.2 Predittori adattivi in avanti

Se assumiamo che il segnale sia stazionario in un frame, solitamente un intervallo di pochi millisecondi (10-50),[22] risulta vantaggioso calcolare quali coefficienti predittivi utilizzare per ogni frame. A questo punto sorgono alcune limitazioni da dover bilanciare: la grandezza dei frame per ottenere la migliore predizione contro la loro quantità totale e quindi la mole di informazioni addizionali diverse per ogni frame, nonché la precisione dei coefficienti predittivi che inficia nella quantità di bit necessari da memorizzare.

Possiamo utilizzare la cosiddetta "segmentazione adattiva" per la compressione: il nostro scopo è suddividere il segnale in frame quasi-stazionari,[23] non obbligatoriamente di lunghezza identica, per ottenere dei coefficienti predittivi ottimali;[24] perciò non ci limitiamo a trovare la grandezza ottimale dei segmenti, ma possiamo dividere il segnale in frame di lunghezza diversa, al fine di guadagnare ulteriormente spazio, anche se a scapito della semplicità, la quale diminuisce, obbligandoci ad aumentare i tempi di computazione.

Dunque risultano efficienti 2 metodi:

dato $G$ come grandezza di *griglia* (costante prefissata)

1. Definiamo $R$ come potenza di due di partenza adeguatamente scelta, dunque partendo da $2^R G$ verifichiamo se comprimere il frame in questione in 2 segmenti di metà lunghezza ($2^{R-1}G$) risulta migliore.[25] In caso positivo lo stesso processo è applicato ricorsivamente a tutti i mezzi segmenti fino a raggiungere una lunghezza di frame minima $G$.
2. Dato un numero arbitrario $k < k_{max}$ massima lunghezza di frame, sfruttiamo un algoritmo in programmazione dinamica per variare $k$ al fine di trovare la compressione migliore per il segmento successivo di lunghezza $kG$.

Focalizziamoci ora nel trovare i coefficienti di predizione migliori per ogni frame: la funzione da minimizzare è la seguente:

$$J(w) = \sum_{i=p}^{q} (x_i - \sum_{j=0}^{M-1} w_j \cdot x_{i-j-1})^2 \tag{3.3}$$

[22] Con buona qualità (frequenza di campionamento a 44100 Hz) otteniamo 441 campioni in un frame da 10 millisecondi, sufficienti per qualsiasi tipo di predizione.
[23] Il segnale è quasi identico se traslato nel tempo di una specifica costante.
[24] Se il segnale è stazionario nel frame è anche notevolmente più prevedibile.
[25] Controllo di tipo iterativo.

Ovvero la sommatoria degli scarti al quadrato tra il valore vero di cui studiare la predizione ($x_i$) e il valore predetto ($\sum_{j=0}^{M-1} w_j \cdot x_{i-j-1}$), trattasi del classico metodo dei minimi quadrati[26] dove *p* è il campione di inizio frame, *q* quello di fine e $\underline{w}$ il vettore di coefficienti del predittore lineare di ordine *M*. Il termine "$x_{i-j-1}$" indica semplicemente i primi *M* valori precedenti a $x_i$. Si noti che i coefficienti $w_j$ non sono altro che dei pesi applicati ai campioni "vicini" al campione da predire, una sorta di media pesata di cui minimizzare lo scarto con il valore vero "futuro".

Per trovare la soluzione ottima possiamo avvalerci dell'algoritmo di Levinson-Durbin che sfrutta l'autocorrelazione del vettore $\underline{r}$ di lunghezza *M+1* così definito:

$$r_j = \sum_{i=p}^{q} x_i \cdot x_{i-j}, \qquad per\ j = 0, 1, \dots, M \tag{3.4}$$

Considerando nulli i valori di $x_i$ fuori dall'intervallo definito, possiamo ricavare dunque un vettore di coefficienti intermediari di riflessione[27] $\underline{k}$ da cui calcolare matematicamente i coefficienti di predizione $\underline{w}$; nella pratica quest'ultima operazione viene effettuata con dei confronti in tabella al fine di risparmiare tempo computazionale.[28]

Per quanto riguarda specificatamente la decisione sulla precisione dei coefficienti effettuiamo la scelta ottima in questo modo: quantizziamo i coefficienti in un certo numero di bit frazionari *cb* decisi a priori tali che

$$k_j = floor(2^{cb} \cdot k_j + 0.5) \tag{3.5}$$

ovvero si scala alla potenza di 2 corrispondente e si arrotonda all'intero più vicino. Solitamente il numero di bit frazionari *cb* dei coefficienti di riflessione varia tra 5 e 8.

A questo punto abbiamo ottenuto, per ogni frame del segnale, una quantità *M* di coefficienti del predittore diversi. Non resta che memorizzare queste informazioni ad ogni segmento.

[26] Un metodo di regressione lineare, usato ampiamente per stimare l'andamento di un fenomeno di cui si conoscono i dati statistici.

[27] Mediante l'algoritmo di Schur, è inoltre disponibile un comando apposito in MATLAB.

[28] Computabile con l'algoritmo di ricorsione di Durbin.

### 3.2.3 Predittori adattivi all'indietro

Ove non possiamo accedere all'informazione del segnale nel "futuro", cioè non abbiamo già la conoscenza di tutto il frame "a portata di mano", risulta estremamente vantaggioso calcolare i coefficienti predittivi sulla base del "passato", ovvero dei campioni precedenti all'interno dell'ultimo segmento. In questo modo non è nemmeno necessario trasmettere i coefficienti utilizzati come informazioni addizionali, poiché è possibile ricalcolarli identici anche in una trasmissione a remoto (ad esempio in un servizio di streaming dove non si conosce il segnale completo fintanto che non si ha atteso la sua intera durata in tempo reale). Di contro il decodificatore, quindi il ricevitore nell'esempio precedente, deve ripercorrere tutti i processi del codificatore (trasmettitore) inversamente, per ogni frame, al fine di poter usufruire del dato originale (come nel caso dei predittori fissi); ciò rischia di rallentare il servizio a causa del maggior tempo computazionale richiesto.[29] Per ovviare a tale problema si opta per un calcolo semplicistico dei coefficienti, come il metodo dei minimi quadrati già descritto, o la sua versione normalizzata o ricorsiva. Riportiamo un esempio di tale procedura in pseudocodice nel paragrafo seguente.

```
For j = 0 to M-1 {                                  //Inizializzazione coefficienti
   Set w_j = 0;
   Set e_j = x_i;
}
For i = M to N-1 {
   Set p = 0;
   For j = 0 to M-1 {                                //Calcola predizione per x_i
      Set p = p + w_j • x_i-j-1;
   }
   Set e_i = x_i - p;
   For j = 0 to M-1 {                               //Imposta I coefficienti del predittore
      Set w_j = w_j + μ • e_i • x_i-j-1;
   }
}
```

Il parametro $\mu$ è una costante molto piccola tale che $\mu < \frac{1}{2Mr_0}$ ($r_0$ è la prima componente del vettore di autocorrelazione $\underline{r}$) e l'array $\underline{e}$ contiene gli errori sulla predizione per ogni campione (indichiamo il *valore i-esimo* con la notazione $e_i$).

Per quanto riguarda le prestazioni notiamo che in questo algoritmo vengono eseguite:

[29] La complessità dell'algoritmo ed il numero di calcoli da effettuare aumentano, di conseguenza è necessario più tempo di elaborazione.

- $2M + 2(N - M) + 2(N - M)M \approx 2(M + N + MN) \approx \theta(MN)$ assegnazioni
- $2(N - M)M \approx \theta(MN)$ addizioni
- $3(N - M)M \approx \theta(MN)$ moltiplicazioni

Tenendo conto che per necessità di bassa latenza non si usa un grado *M* maggiore di 10, *N* risulta quindi molto più grande di *M,*[30] si può affermare che l'algoritmo esegue in toto $\theta(MN) \approx \theta(N)$ operazioni.

### 3.2.4 Predizione stereo

È possibile implementare un'ulteriore tecnica di predizione se il segnale è dotato di entrambi i canali, sinistro e destro, basandoci sul fatto che solitamente l'informazione dei due canali non differisce molto. Utilizzando dunque 2 vettori dei coefficienti predittivi: $\underline{w}$ e $\underline{w}'$, di ordine *M* e *L*, rispettivamente per i due canali,[31] possiamo ottenere le predizioni nella maniera seguente:

$$\widehat{x_i} = \sum_{j=0}^{M-1} w_j \cdot x_{i-j-1} + \sum_{j=0}^{L-1} w_{j+M} \cdot y_{i-j-1} : campione\ futuro\ can.sx \qquad (3.6)$$

$$\widehat{y_i} = \sum_{j=0}^{M-1} w'_j \cdot y_{i-j-1} + \sum_{j=0}^{L-1} w'_{j+M} \cdot x_{i-j} : campione\ futuro\ can.dx \qquad (3.7)$$

Risulta, come anche intuitivamente si deduce, che nel caso di segnali quasi identici nei canali, implementando questa compressione, si dimezza ulteriormente la grandezza del file finale. È peculiare l'omissione della sottrazione unitaria al pedice di $x_{i-j}$ nel "campione futuro canale destro", essa trova spiegazione nel fatto che è possibile utilizzare il valore $x_i$ dato che questo valore è già stato calcolato precedentemente nel canale sinistro. Ciò realizza quindi un abuso di notazione in quanto tale valore necessiterebbe della dicitura "con cappello" $\widehat{x_i}$ poiché trattasi di un valore predetto, non vero.[32]

[30] Ricordiamo che *N* è la grandezza del blocco di campioni ed *M* l'ordine del predittore polinomiale.
[31] I vettori di predizione avranno entrambi lunghezza *M*+*L*, al fine di contenere *M* valori di predizione per il proprio canale ed *L* per l'altro.
[32] Ciò comporta che verranno usati i campioni più vicini di un intervallo di tempo del canale sinistro per la predizione del destro, uno shift della selezione che include anche il campione al tempo "presente" (quello calcolato precedentemente con l'elaborazione del canale sinistro).

# 4. Codifica entropica

Ora che abbiamo diviso l'informazione ridondante da quella imprevedibile,[33] non rimane altro che salvare in memoria i residui della predizione (gli errori), nel modo più efficiente possibile: la codifica entropica. Allora riportiamo innanzitutto in figura 4.1 un riassunto dei blocchi operativi fondamentali discussi.

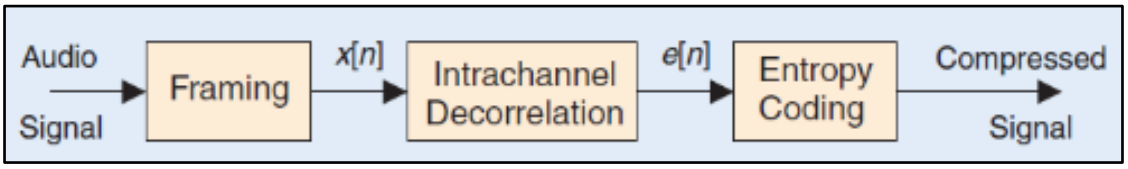


***Figura 4.1:*** *Blocchi operativi fondamentali della compressione audio lossless. Partendo dal segnale audio digitalizzato, avviene la suddivisione in frame per poi calcolare l'errore sulla predizione nel blocco di decorrelazione intra-canale ed infine salvare i residui e[n] con codifica entropica.*

Ma prima di parlare di questo tipo di codifica cerchiamo di chiarire il concetto di entropia e come questa sia collegata alla rappresentazione delle informazioni.

## 4.1 Entropia

All'interno dell'ambito della teoria dell'informazione, l'entropia è definita come la quantità informazionale media che un messaggio di qualsiasi tipo porta con sé. Non è equivalente all'ammontare di parole o dati che vengono trasmessi, anzi è completamente scorrelata da questo valore: essa dipende direttamente dall'imprevedibilità dell'informazione acquisita grazie al messaggio in questione. Diamo la formula (4.1) e discutiamone un esempio.

$$H(x) = \sum_{a \epsilon \mathcal{A}_x} p_x(a) \cdot \log_2 \frac{1}{p_x(a)} = E[i_x(x)] \tag{4.1}$$

dove $x$ è una variabile aleatoria discreta, $\mathcal{A}_x$ il suo alfabeto (la lista di tutti i possibili valori che può assumere), $a$ un evento appartenente all'alfabeto, $p_x(a)$ la probabilità che quell'evento si verifichi, $\log_2 \frac{1}{p_x(a)}$ è la quantità di informazione che un evento porta con sé. La dicitura $E[\cdot]$ indica il "*valore atteso di*", ovvero il valore che ci si aspetta avrà quella data funzione solitamente.

[33] Effettuando delle operazioni simili a quelle utilizzate nella progettazione di circuiti elettronici in regime sinusoidale per rimuovere la media da un segnale e mantenere solo la parte variabile, per studiarle separatamente.

Ora veniamo ad un esempio per comprendere appieno il concetto di informazione ed entropia di un messaggio, al fine di poterlo utilizzare nell'ambito del nostro studio: notiamo che nella giornata odierna il tempo è sereno, una persona ci dice la stessa cosa (ovvero che il tempo è sereno), la quantità di informazione del suo messaggio è pari a 0, poiché l'evento che ha descritto è ovvio (con probabilità 100%). Se invece questa persona ci avverte che tra pochissimo pioverà, contando che questo sia vero, allora l'informazione aumenta notevolmente poiché la probabilità che ciò accada è davvero scarsa. Nel caso sapessimo che la probabilità di cambio meteo in poco tempo sia del 5% e del 95% quella con cui il meteo rimanga identico a quello già presente, avremmo un alfabeto semplicistico con soli due eventi possibili e l'informazione di questo messaggio risulterebbe: $i = \log_2 \frac{1}{0.05} = 4{,}32\ bit$ e l'entropia di questo tipo di messaggi,[34] assumendo che vi sia una fonte di previsioni come la persona citata, diventerebbe: $H = 0.05 \cdot \log_2 \frac{1}{0.05} + 0.95 \cdot \log_2 \frac{1}{0.95} = 0{,}29\ bit/simbolo$. Questo ultimo valore indica quindi quanti bit sono necessari in media[35] per codificare l'informazione relativa agli eventi:

- Tra poco il meteo rimarrà invariato
- Tra poco il meteo cambierà

Trattandosi di soli due eventi, ci bastano due simboli per codificare tutte le informazioni, inoltre il 95% delle volte ci sarà sempre la previsione del meteo sereno, quindi se assegniamo "1" e "0" ai due eventi, il 95% dei messaggi conterrà un "1" e il restante 5% uno "0".

[34] Notare che l'entropia è una *media*, assume pertanto significato quando ci aspettiamo che i messaggi siano più d'uno.

[35] Ovvero dopo una lunga serie di questi messaggi, otteniamo questa media; naturalmente un valore frazionario non rappresenta una quantità concreta, ma solo un dato astratto.

### 4.1.1 L'entropia nell'assegnazione dei simboli

Nel caso avessimo più eventi da simboleggiare, ovvero un alfabeto più ampio, ad esempio di 4 simboli, è ovviamente necessario utilizzare sequenze di 2 cifre assegnando ad esempio:

- A = 00
- B = 01
- C = 10
- D = 11

E i messaggi risulterebbero del tipo: D,C,D,B = 11,10,11,01 = 11101101 concatenati. Ma accorgendoci che un valore occorre più volte degli altri (da verificare in media), ovvero i simboli non sono equiprobabili, possiamo decidere di impiegare meno bit per codificare il valore più ricorrente, in modo da diminuire la lunghezza della stringa finale. Questa tecnica di compressione dell'informazione è chiamata codifica entropica. Diamo un esempio semplice tramite il codice unario:

- A = 0001
- B = 001
- C = 01
- D = 1 (maggiore probabilità)

Allora il messaggio precedente risulterà: D,C,D,B = 1,01,1,001 = 1011001 risparmiando un bit.

Chiaramente dobbiamo poter distinguere un simbolo dall'altro, quindi si parla di codifiche a prefisso, un tipo di codici che permettono di identificare inequivocabilmente i simboli anche se sono descritti con *parole*[36] di lunghezza variabile senza il bisogno di informazioni addizionali. Ciò viene realizzato, come dice il nome stesso, grazie ai prefissi univoci.[37]

[36] Qui intese come sequenze di cifre binarie.
[37] Un esempio di prefix-code (codice a prefisso) molto efficiente è l'Huffman.

## 4.2 Codifica Golomb e Golomb-Rice

Tornando quindi al problema di trovare un modo di rappresentare i residui *"x"* della predizione in modo efficiente, notiamo sperimentalmente che essi hanno una distribuzione probabilistica di tipo laplaciano bilaterale simmetrico,[38] è quindi inizialmente utile togliere il segno da questi valori per riorganizzarli in una scala unilaterale, dove i valori risultano distribuiti come in una variabile aleatoria geometrica unilaterale. Eseguiamo questa conversione con la funzione (la moltiplicazione per 2 è possibile mediante un singolo shift)

$$u(x) = \begin{cases} 2x & per\ x \geq 0 \\ -2x - 1 & per\ x < 0 \end{cases} \tag{4.2}$$

dove la quantità di bit per codificare $x$ è la stessa per la codifica di $u(x)$.

Successivamente possiamo ragionevolmente optare per un codice Golomb o Golomb-Rice, ottimi per le distribuzioni geometriche unilaterali. Per quanto riguarda il salvataggio dei valori il codice Golomb utilizza un parametro *s*, prefissato, per dividere ogni valore $u$ in $m = floor(u/s)$ blocchi di grandezza fissa e $l = u(x)\ mod\ s$ residui in modo che risulti $u = s \cdot m + l$, analogamente alla fase di pre-processing di scala dei valori. A questo punto il numero $m$ è salvato in codice unario[39] (in modo da identificare facilmente la fine del numero avvalendosi dello "0" finale) ed $l$ in codifica binaria troncata. Quest'ultimo valore viene salvato nel seguente modo:

definiamo $b = floor(\log_2 s)$ allora

- Se $l < 2^{b+1} - s$ viene memorizzato *l* in binario con *b* bit.
- Se $l \geq 2^{b+1} - s$ viene memorizzato $l + 2^{b+1} - s$ con *b+1* bit.

A questo punto viene codificata solamente l'informazione relativa ad *m* ed *l*, poiché *s* è una costante decisa a priori. Riportiamo ad esempio la tabella 4.1 al fine di chiarire maggiormente la rappresentazione dei residui secondo questo codice.

[38] Una distribuzione simile a quella esponenziale.
[39] Ovvero avremo una quantità *m* di "1" seguita da un bit nullo.

| x | s = *5* | s = 8 | s = 10 | x | s = *5* | s = 8 | s = 10 |
|---|---|---|---|---|---|---|---|
| **0** | 0:00 | 0:000 | 0:000 | 10 | 110:00 | 10:010 | 10:000 |
| **1** | 0:01 | 0:001 | 0:001 | 11 | 110:01 | 10:011 | 10:001 |
| **2** | 0:10 | 0:010 | 0:010 | 12 | 110:10 | 10:100 | 10:010 |
| **3** | 0:110 | 0:011 | 0:011 | 13 | 110:110 | 10:101 | 10:011 |
| **4** | 0:111 | 0:100 | 0:100 | 14 | 110:111 | 10:110 | 10:100 |
| **5** | 10:00 | 0:101 | 0:101 | 15 | 1110:00 | 10:111 | 10:101 |
| **6** | 10:01 | 0:110 | 0:1100 | 16 | 1110:01 | 110:000 | 10:1100 |
| **7** | 10:10 | 0:111 | 0:1101 | 17 | 1110:10 | 110:001 | 10:1101 |
| **8** | 10:110 | 10:000 | 0:1110 | 18 | 1110:110 | 110:010 | 10:1110 |
| **9** | 10:111 | 10:001 | 0:1111 | 19 | 1110:111 | 110:011 | 10:1111 |

***Tabella 4.1:*** *Esempio di codice Golomb con diversi valori di s (x rappresenta il valore da codificare).*

Alternativamente, il codice Golomb-Rice che è maggiormente utilizzato, limita il parametro *s* ad una potenza di due, quindi $s = 2^p$, con *p* intero positivo, per migliorare l'efficienza computazionale dell'algoritmo di codifica eliminando la necessità di una divisione ($u/s$) che viene effettuata tramite shift binari.[40] Nonostante ciò in linea teorica sarebbe migliore un algoritmo senza questo limite.[41]

Per trovare il valore ottimo di *p* e quindi anche di *s* abbiamo due strade: adattivo in avanti o all'indietro. Nel primo caso la computazione è semplificata e quindi migliore in termini di tempo di esecuzione con le relative conseguenze, nel secondo è richiesta almeno un'analisi ricorsiva.

1. $p \approx \log_2 E[u(x)] \rightarrow s \approx E[u(x)]$, controllando anche i valori "vicini" al valore atteso intero per poi selezionare il migliore per la compressione. Necessita quindi di un'analisi statistica su tutti i valori $u(x)$.
2. Ricorsione di "$U_{i+1} = \beta U_i + (1-\beta)u(x_i)$" con $0 < \beta < 1$, dunque $s_i = U_i$ e $p_i = floor(\log_2 E[U_i])$. Ad ogni ricorsione si mantiene la coppia migliore per la compressione di valori $s_i$ e $p_i$.

Successivamente a questa conversione in specifici formati, si giunge alla codifica ultima. Descriviamo anche in pseudocodice l'algoritmo Golomb-Rice, il quale risulta più "comodo" dal punto di vista della complessità e della computazione. Ciò può venire

[40] Ad esempio 42/4 = 42/$2^2$ => 101010/100 = 1010 = $10_{10}$ (resto non rappresentato). Notiamo quindi che il numero in binario della decina corrisponde al 42 con il doppio ($2^2$) spostamento verso destra di ogni cifra binaria in 42 (eliminando le 2 cifre meno significative).

[41] La codifica binaria del residuo è più semplice.

eseguito per ogni frame o addirittura sul segnale intero, senza segmentazione quindi, per risparmiare informazioni addizionali, ma a scapito dell'efficienza in compressione: si avranno residui più grandi nonché maggiormente variabili e saranno necessari più bit per rappresentarli in memoria.

```
Set s = *calcolo di s con uno dei metodi discussi*;          //calcolo notazione
Write s;                                                      //salva sul file
For (i = 0; i < u.length; i++) {          //u è il vettore di campioni secondo (2.6)
   Set m = floor(u(i)/s);
   Set l = u(i) mod s;
   Set mcoded = "stringa di m bit a 1 seguita da uno 0";      //codifica unaria
   Set b = floor(log2(s));
   If (l < 2^(b+1) – s) {                                     //codifica binaria troncata
      Set lcoded = "(l) in binario con b bits";
   } else {
      Set lcoded = "(l+2^(b+1) – s) in binario con (b+1) bits";
   }
   Write mcoded + lcoded";
}
```

Nello sviluppo sul codice reale sarà ovviamente necessario implementare una categorizzazione più dettagliata nonché efficiente delle informazioni da salvare nel futuro file compresso. Al momento della decodifica sarà sufficiente eseguire le operazioni inverse, rispettando la formattazione utilizzata per il salvataggio delle informazioni addizionali (ad esempio la lunghezza dei frame, il parametro *s*, ecc.), nonché distinguere accuratamente la fine e l'inizio della rappresentazione dei valori per via della codifica binaria troncata[42] e di quella unaria, i quali possono essere fonte di errori irreparabili, ma facilmente riconoscibili, a causa della lunghezza variabile.

### 4.2.1 Conclusioni

Abbiamo affrontato tutti i presupposti teorici fondamentali alla creazione di codifiche audio lossless efficienti, discutendo delle fasi di:

1. Pre-processing – eliminazione delle informazioni superflue
2. Predizione – diminuzione variabilità del segnale
3. Codifica entropica – compressione efficiente dei dati finali

ora possiamo dedicarci allo studio delle loro relative prestazioni.

[42] Si può eventualmente sostituire questa codifica con un'altra a prefisso e quindi a lunghezza variabile, come ad esempio quella di Huffman.

# 5. Codificatore lossless in MATLAB

Al fine di approfondire maggiormente le tecniche discusse abbiamo voluto realizzare un programma di codifica lossless dei file audio, il quale seppur utilizzando i metodi più basilari, mostra il rendimento ottenuto mettendo in pratica gli argomenti trattati. Vaglieremo dunque i risultati per poi confrontarli con un algoritmo open-source largamente diffuso di cui abbiamo già accennato nella prima sezione: FLAC (Free Lossless Audio Codec).

## 5.1 Struttura del programma

Abbiamo deciso di omettere le operazioni di pre-processing in quanto considerate apportare un guadagno trascurabile secondo il nostro obiettivo: un esempio dimostrativo con tecniche basilari. Dunque abbiamo utilizzato dei predittori a coefficienti fissi di ordine variabile da 1 a 5 per la predizione del segnale ed una codifica Golomb-Rice senza rappresentazione binaria troncata,[43] con scelta del fattore *s* come valore atteso del segnale. Abbiamo inoltre voluto mantenere due versioni di questa fase finale al fine del confronto: la prima salva l'informazione relativa al fattore moltiplicativo *m* con una codifica binaria ordinaria, ma evitando di aggiungere i bit più significativi che non vengono mai utilizzati tra tutti i campioni computati, la seconda versione invece implementa una codifica unaria per questo fattore *m*, variando quindi la lunghezza delle parole di codice utilizzate.

## 5.2 Test di compressione

Utilizzando un set di 9 spezzoni audio di diverso genere, scaricabile gratuitamente da *soundexpert.org* alla voce *"audio samples"*, intendiamo ora analizzare i risultati del nostro algoritmo e confrontarli con quelli del codificatore FLAC.[44]

Riportiamo la tabella 5.1 per mostrare più in dettaglio le caratteristiche di questi campioni audio di alta qualità (frequenza di campionamento 44'100 Hz / 16 bit per campione) salvati senza compressione in formato WAVE.

---

[43] Sostituita con una codifica binaria a lunghezza fissa per i residui *l*.

[44] Abbiamo utilizzato il codificatore FLAC offerto dalle librerie già presenti in MATLAB (comando: "audiowrite(finename.flac, array, frequency)").

| Nome | Grandezza (MB) | Durata (s) | Descrizione | Genere |
|---|---|---|---|---|
| Bah.wav | 1,59 | 9 | Pianoforte - J.S. Bach | Classica |
| Bas.wav | 2,00 | 11 | Canto generico | Lirica |
| Cst.wav | 1,09 | 6 | Nacchere e chitarra | Tradizione spagnola |
| Fms.wav | 1,86 | 11 | Discorso francese maschile | Parlato |
| Glk.wav | 2,57 | 15 | Xilofono | Scala delle note |
| Hrp.wav | 1,47 | 8 | Clavicembalo | Scala delle note |
| Lob.wav | 1,19 | 7 | "You were here" - Lobo | Lo-Fi |
| Mof.wav | 1,79 | 10 | "Music from the balcony" - Mike Oldfield | Elettronica |
| Qrt.wav | 2,06 | 12 | Quartetto cantato | Lirica |

***Tabella 5.1:*** *Elenco del set di campioni audio di piccola durata.*

Definiamo ora il rapporto di compressione come:

$$r = \frac{grandezza\ file\ originale}{grandezza\ file\ compresso} \tag{5.1}$$

e riportiamo la tabella 5.2 al fine di una rapida comprensione dell'efficienza raggiunta dal nostro algoritmo: in essa troviamo la grandezza dei file originali ed i rapporti di compressione *r* relativi alla codifica ottenuta con polinomi di predizione di ordine differente. Possiamo interpretare quest'ultimi valori come il numero di volte con cui la grandezza del file originale è stata compressa.

| Nome | Dimensione originale (KB) | Rapporto r sul file codificato con ordine | | | | |
|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 |
| bah | 1634 | 1,57 | 1,96 | 2,09 | 1,43 | 1,35 |
| bas | 2056 | 1,50 | 1,80 | 2,02 | 1,37 | 1,31 |
| cst | 1124 | 1,42 | 1,41 | 1,35 | 1,31 | 1,24 |
| fms | 1913 | 1,61 | 1,72 | 1,64 | 1,45 | 1,38 |
| glk | 2653 | 1,49 | 1,65 | 1,74 | 1,37 | 1,30 |
| hrp | 1515 | 1,44 | 1,38 | 1,31 | 1,26 | 1,23 |
| lob | 1222 | 1,46 | 1,44 | 1,39 | 1,33 | 1,26 |
| mof | 1835 | 1,46 | 1,47 | 1,43 | 1,34 | 1,26 |
| qrt | 2115 | 1,50 | 1,80 | 1,91 | 1,37 | 1,31 |
| **Media** | | **1,50** | **1,63** | **1,65** | **1,36** | **1,29** |

***Tabella 5.2:*** *Elenco rapporti "r" sui file codificati con predittore di ordine variabile e codifica unaria di m.*

Un risultato che risulterà con molta probabilità controintuitivo a molti è la diminuzione di efficienza in compressione media al di sopra del terzo ordine; ciò sta a significare che generalmente la variazione del segnale negli ultimi 5 campioni[45] è troppo ampia per poterla adeguatamente sfruttare con dei predittori a coefficienti fissi.
Notiamo quindi che il predittore di ordine 3 si rivela mediamente migliore ed è dunque quello consigliato qualora non si volesse implementare una scelta sull'ordine da utilizzare per codificare una generica traccia audio.
A questo punto verifichiamo il cambiamento apportato nel rendimento dalla codifica unaria del coefficiente *m* tramite la tabella 5.3 (polinomio di ordine 3).

| Nome | Dimensione originale (KB) | Fattore di compressione r con codifica | | Differenza |
|---|---|---|---|---|
| | | Unaria | Binaria | |
| bah | 1634 | 2,09 | 1,46 | 0,64 |
| bas | 2056 | 2,02 | 1,33 | 0,69 |
| cst | 1124 | 1,35 | 0,89 | 0,46 |
| fms | 1913 | 1,64 | 1,14 | 0,50 |
| glk | 2653 | 1,74 | 1,01 | 0,73 |
| hrp | 1515 | 1,31 | 0,89 | 0,42 |
| lob | 1222 | 1,39 | 1,00 | 0,39 |
| mof | 1835 | 1,43 | 0,94 | 0,49 |
| qrt | 2115 | 1,91 | 1,23 | 0,68 |
| **Media** | | **1,65** | **1,10** | **0,56** |

***Tabella 5.3:*** *Confronto nei rapporti di compressione tra codifica unaria e binaria del coefficiente m.*

Innanzitutto notiamo che nei casi senza codifica unaria ove $r < 1$ il file codificato addirittura si espande e diviene leggermente più grande di quello originale. Risulta dunque doveroso implementare un qualche tipo di codifica purché non semplicemente binaria nel salvataggio finale dei dati per garantire almeno di non appesantire il file.
Dunque la semplice aggiunta della codifica unaria di *m* apporta un guadagno medio di *0,56* nel rapporto di compressione, un risultato che reputiamo dovuto in gran parte alla presenza di numerosi momenti di silenzio all'interno delle canzoni: questi spezzoni, ove i campioni digitalizzati hanno valore nullo, sono codificabili mediante un singolo bit a "0" nella codifica unaria;[46] al contrario in quella binaria si è obbligati ad utilizzare la stessa quantità di bit usata per gli altri campioni non nulli. Da quest'ultima deriva un

[45] Ricordiamo che l'ordine del predittore si riferisce a quanti campioni "indietro nel tempo" vengono utilizzati per calcolare il campione futuro.
[46] Senza contare i bit *l* del resto.

notevole spreco, ma anche la codifica unaria ha dei lati negativi: se la scelta del coefficiente *s* con cui avviene la divisione dei valori non è effettuata adeguatamente, le quantità *m* da codificare potrebbero essere molto grandi e richiederebbero numerosissimi bit a "1" consecutivi da scrivere nel file codificato.[47]

Infine, dato che i file "cst.wav" e "hrp.wav" dimostrano le difficoltà più grandi in compressione del nostro algoritmo, ne riportiamo i diagrammi temporali in figura 5.1 e 5.2 al fine di analizzarlo e comprendere la natura di questa riduzione in efficienza.

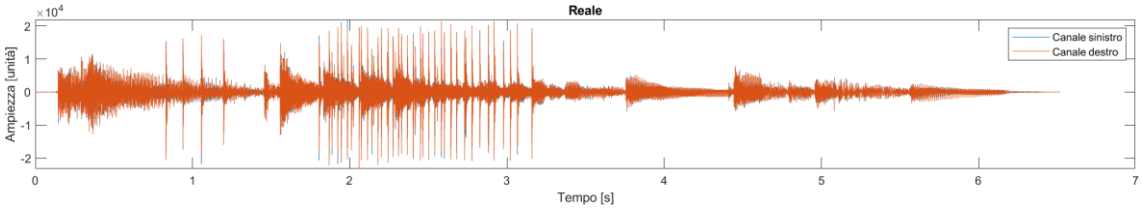


***Figura 5.1:*** *Diagramma del file audio "cst.wav". Asse X=Tempo [s] | Asse Y=Ampiezza [unità], Blu = Canale sinistro | Arancione = Canale destro.*

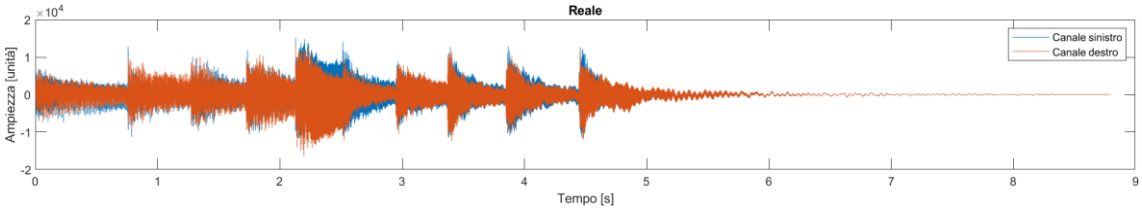


***Figura 5.2:*** *Diagramma del file audio "hrp.wav". Asse X=Tempo [s] | Asse Y=Ampiezza [unità], Blu = Canale sinistro | Arancione = Canale destro.*

In entrambi questi file audio troviamo diversi picchi repentini, i quali non sono prevedibili in quanto non seguono l'andamento lineare dei campioni precedenti e rendono l'errore massimo sulla predizione molto grande; di conseguenza la quantità di bit necessaria a codificare quest'informazione può aumentare notevolmente.

Contrariamente nella traccia audio "bah.wav" descritta in figura 5.3 non vi sono questi picchi, anzi il segnale presenta un andamento piuttosto poco variabile o comunque con cambiamenti più graduali (musica melodica) molto più semplici da prevedere; allora l'errore sarà molto minore, il che renderà necessario meno contenuto informativo.

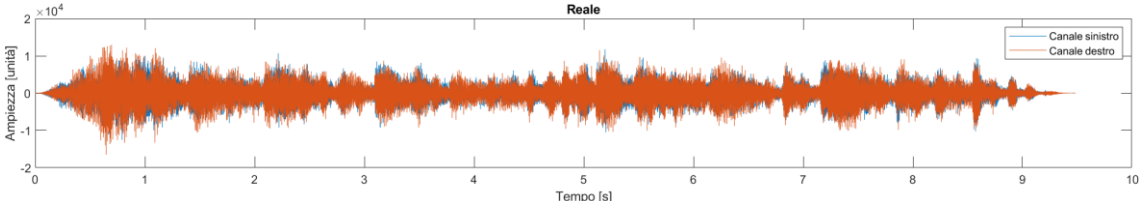


***Figura 5.3:*** *Diagramma del file audio "bah.wav". Asse X=Tempo [s] | Asse Y=Ampiezza [unità], Blu = Canale sinistro | Arancione = Canale destro.*

[47] Ad esempio per $s = 16$, un valore $u(x) = 8000$ renderebbe $m = 500$, il quale sarebbe codificato con 500 bit a 1 consecutivi (codifica unaria).

Riportiamo infine le figure 5.4 e 5.5 relative agli errori calcolati con polinomio predittivo di ordine 3 dei file "bah.wav" e "hrp.wav". Notiamo l'ampiezza del valore massimo raggiunto dai due errori: nel primo caso troviamo circa $10^3$, nel secondo arriviamo addirittura a necessitare una scala di ampiezza che raggiunge $10^5$, una differenza di ben due ordini di grandezza.

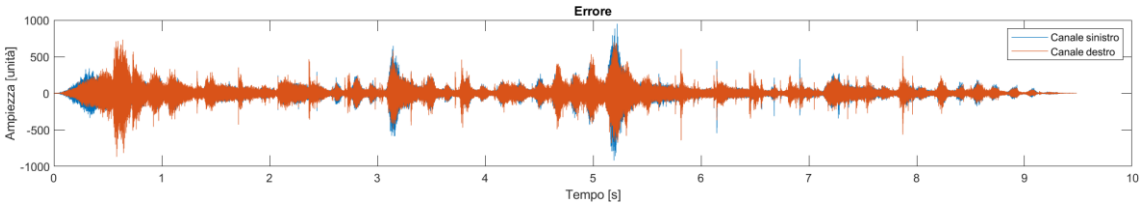


***Figura 5.4:*** *Diagramma dell'errore sulla predizione in "bah.wav". Asse X=Tempo [s] | Asse Y=Ampiezza [unità], Blu = Canale sinistro | Arancione = Canale destro.*

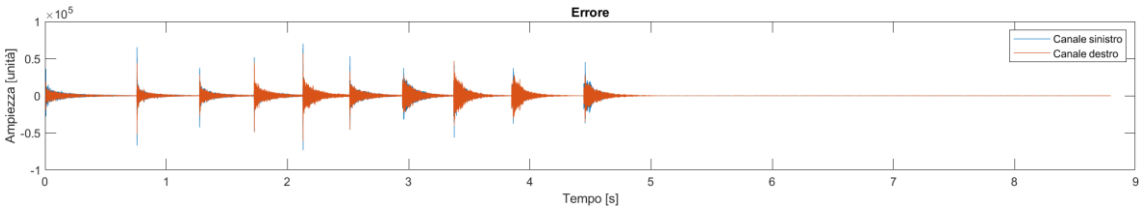


***Figura 5.5:*** *Diagramma dell'errore sulla predizione in "hrp.wav". Asse X=Tempo [s] | Asse Y=Ampiezza (x$10^5$) [unità], Blu = Canale sinistro | Arancione = Canale destro.*

Per questo motivo troviamo una differenza nel rapporto di compressione di ben *0,78* tra la codifica dei due file.[48]

## 5.3 Confronto con lo standard FLAC

Con l'obiettivo di dare una scala di misura con cui valutare le prestazioni del nostro algoritmo dimostrativo, proponiamo un confronto con lo standard ben affermato tra gli audiofili digitali: FLAC.

Dunque riportiamo la tabella 5.4 ove vengono affiancati i risultati della miglior versione del nostro codificatore e dello standard FLAC.

[48] Con polinomio di predizione d'ordine 3 e codifica unaria di *m*.

| Nome | CSEncoder | FLAC | Differenza |
|---|---|---|---|
| bah | 2,09 | 2,31 | 0,22 |
| bas | 2,02 | 2,50 | 0,48 |
| cst | 1,42 | 1,92 | 0,50 |
| fms | 1,72 | 2,39 | 0,68 |
| glk | 1,74 | 4,36 | 2,62 |
| hrp | 1,44 | 2,00 | 0,56 |
| lob | 1,46 | 1,76 | 0,30 |
| mof | 1,47 | 1,89 | 0,42 |
| qrt | 1,91 | 2,37 | 0,45 |
| **Media** | **1,70** | **2,39** | **0,69** |

***Tabella 5.4:*** *Confronto in fattore di compressione "r" tra l'esempio da noi realizzato (CSEncoder) e lo standard FLAC.*

Ricordando che in FLAC troviamo la segmentazione del file audio, la previsione adattiva con coefficienti variabili nonché quella stereo, la codifica RLE (Run Length Coding)[49] e Verbatim,[50] possiamo ritenere che il nostro programma raggiunge pienamente l'obiettivo desiderato con uno scarto medio di solo *0,69* in tasso di compressione.

### 5.3.1 Test su audio di lunga durata

Concludiamo la fase di testing con delle prove su file di più lunga durata, al fine di verificare il corretto funzionamento del nostro algoritmo anche con grandi quantità di dati, nonché controllare se lo scarto di efficienza con lo standard FLAC rimane costante o varia.

A tal proposito affianchiamo i risultati della versione migliore in media del nostro programma (con predittore di ordine 3 e codifica unaria di *m*) alle prestazioni del FLAC. Diamo innanzitutto la lista delle tracce audio utilizzate in tabella 5.5.

[49] Una codifica che tiene conto delle ripetizioni consecutive di un dato. Ad esempio "5888884" diventa, a livello logico, "5x1+8x5+4x1".

[50] Una tecnica che comprende quali parti di un set di dati sono prevedibili e quali no, allora opera in modo diverso in base a questa conoscenza.

| Nome | Grandezza (MB) | Durata (min) | Genere | Descrizione |
|---|---|---|---|---|
| Bring me to life.wav | 47,4 | 4:12 | Rock | Evanescence, Fallen |
| Allegro con fuoco.wav | 149,8 | 13:18 | Orchestra classica con ritmo veloce | Gustavo Dudamel Dvorak, Sinfonia No.9, 4° Movimento |
| Hotel California.wav | 75,0 | 6:40 | Rock leggero | Eagles, Hotel California |
| In my mind.wav | 38,9 | 3:26 | Elettronica con voce | Gigi D'Agostino & Dynoro |
| Allegro appassionato.wav | 43,5 | 3:51 | Orchestra classica con ritmo leggero | Jacqueline du Pré, Saint-Saëns, Op.43 |
| Nessun dorma.wav | 36,6 | 3:15 | Lirica | Pavarotti, Turandot, Concerto 1994 |
| Rachmaninoff No.2 Op.18.wav | 425,3 | 37:48 | Concerto completo pianoforte | Rachmaninoff, concerto completo |
| Time.wav | 58,3 | 5:10 | Colonna sonora di musica classica | Hans Zimmer, Inception |

***Tabella 5.5:*** *Elenco tracce audio per test di lunga durata.*

E ora riportiamo i risultati in termini di rapporti *r* nella tabella 5.6.

| Nome | CSEncoder | FLAC | Differenza |
|---|---|---|---|
| Bring me to life | 1,26 | 1,59 | 0,33 |
| Allegro con fuoco | 1,55 | 2,14 | 0,60 |
| Hotel california | 1,24 | 1,61 | 0,37 |
| In my mind | 1,26 | 1,75 | 0,49 |
| Allegro appassionato | 2,19 | 2,83 | 0,65 |
| Nessun dorma | 1,78 | 2,62 | 0,84 |
| Rachmaninoff No.2 Op.18 | 2,26 | 3,05 | 0,79 |
| Time | 1,80 | 2,69 | 0,89 |
| **Media** | **1,67** | **2,29** | **0,62** |

***Tabella 5.6:*** *Rapporti di compressione nei campioni audio di lunga durata.*

Notiamo quindi un lievissimo miglioramento nelle prestazioni del nostro programma rispetto al FLAC ovvero la differenza media del coefficiente *r* è scesa da *0,69* a *0,62*. Imputiamo questo relativo peggioramento dello standard FLAC alla tecnica della segmentazione, la quale con segnali di lunga durata costringe ad avere un gran numero di frame, ognuno con i suoi parametri diversi. In ogni caso la perdita di efficienza dovuta a questo fenomeno è trascurabile se comparata al guadagno ottenuto, ovvero un *r* medio di *2,29*.

# Conclusioni

Ci eravamo preposti l'obiettivo di studiare i metodi di compressione audio senza perdite, approfondendoli in modo da riuscire a creare un programma che riuscisse in questo intento. A tal fine abbiamo descritto le tre fasi fondamentali di questo tema, discutendole con l'intenzione di rendere chiarezza anche ad un lettore lontano da questo settore, ma senza rimuovere la trattazione dei dettagli tecnici, la quale risulta essenziale per una comprensione completa e feconda.

Riteniamo di aver realizzato questi obiettivi e anche avendo creato un algoritmo di compressione utilizzando le tecniche più basilari, abbiamo ottenuto un buon risultato in termini di efficienza se rapportato alla complessità implementata.
I problemi sorti con più frequenza nello sviluppo di questo programma riguardavano le diverse tipologie di cambi di notazione dei valori utilizzati durante le varie fasi di computazione: da un tipo di codifica all'altra, nonché utilizzare un intervallo numerico più ampio di quello permesso dai bit utilizzati per la rappresentazione dei campioni audio "grezzi", al fine di poter eseguire moltiplicazioni e somme senza errori. Evidenziamo quindi la necessità di una conversione dei valori ricavati dal segnale originale ed una successiva riconversione al momento della codifica entropica e della decodifica.

In conclusione possiamo affermare che rimane un notevole margine di miglioramento nelle prestazioni del nostro algoritmo, molte tecniche ancora da includere, ma che sono state già studiate ed approfondite e pertanto possiamo ritenerci soddisfatti nell'aver costruito delle solide basi per un futuro sviluppo di questo progetto.

# Riferimenti bibliografici

Mat Hans & Ronald W. Schafer, “Lossless compression of digital audio”, IEEE signal processing magazine, 2001

Florin Ghido, “An Introduction to Lossless Audio Compression”, Department of Signal Processing - Tampere University of Technology

Solomon W. Golomb, “Run Lengths Encodings”, IEEE Transactions of information theory”, pp. 399-401, 1966

Benvenuto Nevio & Zorzi Michele, “Principles of Communications Networks and Systems”, pag. 164, 2011

## Sitografia:

“Computation linear prediction coefficients” - https://ccrma.stanford.edu/~jos/sasp/Computation_Linear_Prediction_Coefficients.html (ultimo accesso: 12/10/2021)

Florin Ghido, “What is OptimFROG?” - http://losslessaudio.org/ (ultimo accesso: 8/10/2021)

“Compressione audio lossy e lossless” - https://leganerd.com/2011/07/23/compressione-audio-lossy-e-lossless/ (ultimo accesso: 11/10/2021)

Chris Neel - “Tutto sulla compressione MP3” - http://www.chrisneel.it/articolo/tuttosump3 (ultimo accesso: 5/10/2021)

Vincenzo Zaglio, “MP3, WMA, formati a confronto” - https://www.01net.it/mp3-e-wma-formati-a-confronto/ (ultimo accesso: 6/10/2021)

“Cosa significa Ogg Vorbis?” - https://www.wikibit.it/o/cosa-significa-ogg-vorbis-3704/ (ultimo accesso: 7/10/2021)

“L’algoritmo di compressione del sistema dts” - https://www.digitalvideoht.it/tecnica/tecnica-generale/lalgoritmo-compressione-del-sistema-dts.html (ultimo accesso: 7/10/2021)

“Cos’è il formato WAV?” - https://www.apowersoft.it/cose-il-formato-wav.html, 2017 (ultimo accesso: 7/10/2021)

“Flac cos’è e perché è meglio dell’mp3” - https://www.tabletprezzi.it/flac-cos-e-mp3/ (ultimo accesso: 7/10/2021)

“Formati audio digitali Lossless e Lossy” - https://cuffiemigliori.it/guida/formati-audio-digitali-lossless-lossy/ (ultimo accesso: 7/10/2021)

"Definizione audio di Monkey: che cos'è il formato APE?" - https://comeaprire.com/blog/2020/10/02/definizione-audio-di-monkey-che-cose-il-formato-ape/
(ultimo accesso: 8/10/2021)

"I formati e i codec di compressione audio e video" - https://www.fastweb.it/internet/i-formati-e-i-codec-di-compressione-per-audio-e-video/
(ultimo accesso: 11/10/2021)

"Lossless predictive audio compression" - https://en.wikipedia.org/wiki/Lossless_predictive_audio_compression
(ultimo accesso: 4/10/2021)

Salvatore Aranzulla, "Migliori formati audio" - https://www.aranzulla.it/migliori-formati-audio-1031352.html
(ultimo accesso: 5/10/2021)

Fabio Biscaro, "Numeri binari in notifica IEEE 754 a 32 bit" - https://www.youtube.com/watch?v=N6U_xUdUWC8
(ultimo accesso: 21/10/2021)

"WAVE PCM file format" - http://soundfile.sapp.org/doc/WaveFormat/
(ultimo accesso: 22/10/2021)

"WAV specifications" - https://en.wikipedia.org/wiki/WAV#Specification
(ultimo accesso: 22/10/2021)

"Golomb coding" - https://en.wikipedia.org/wiki/Golomb_coding
(ultimo accesso: 26/10/2021)

"How FLAC works" - https://hydrogenaud.io/index.php?topic=110551.0
(ultimo accesso: 12/11/2021)

"Sound Samples" - http://soundexpert.org/sound-samples
(ultimo accesso: 13/11/2021)