UNIVERSITÀ DEGLI STUDI DI MILANO
FACOLTÀ DI SCIENZE E TECNOLOGIE

Corso di Laurea in Comunicazione Digitale

UN APPROCCIO PERCETTIVO PER IL RESTAURO DELLA DINAMICA AUDIO

Relatore:
Prof. Luca Andrea Ludovico

Correlatori:
Prof. Alessandro Rizzi
Dott. Giorgio Presti

Tesi di Laurea di:
Massimiliano Santamaria
Matr. n. 739767

Anno Accademico 2014/2015

Indice

Capitolo 1

Introduzione

L'algoritmo ACE (Automatic Color Equalization) propone un'innovativa strategia per l'elaborazione delle immagini digitali. Esso si basa su un modello computazionale del sistema visivo umano che fonde i due meccanismi di equalizzazione globale "Gray World" e "White Patch". Come accade con alcuni meccanismi di adattamento tipici della visione umana, ACE realizza un effetto di filtraggio locale prendendo in considerazione la distribuzione spaziale del colore nell'immagine. La sua peculiarità è quella di adattarsi ad un'ampia varietà di condizioni di luminosità.

Sostanzialmente l'obiettivo di ACE è quello di massimizzare la dinamica dell'immagine, così come il contenuto informativo della scena percepita. Sulla base di queste considerazioni, lo scopo di questa tesi è l'implementazione di tale algoritmo attraverso uno dei linguaggi di programmazione ad oggi disponibili per poter automatizzare il processo di restauro della dinamica in suoni digitalizzati che necessitano di tale intervento.

1.1 Obiettivi

Con il seguente elaborato si vuole indagare sulle metodologie utilizzate in ambito informatico per la rielaborazione della dinamica nell'audio, principalmente digitale. Inoltre verranno presentati i risultati di un test approfondito che è stato svolto su estratti audio utilizzando l'algoritmo ACE, originariamente studiato per la dinamica dell'immagine digitale. Tale algoritmo presenta una funzione differenza e una funzione distanza per confrontare i campioni, le quali sono state implementate in diverse varianti, e considerando diversi intervalli per limitare il confronto e velocizzare l'algoritmo, cercando di renderlo utilizzabile anche su macchine con potenza di calcolo meno elevata.

1.2 Struttura

L'elaborato in questione è suddiviso in quattro parti fondamentali: lo stato dell'arte, un primo approccio all'algoritmo, la descrizione di una sua implementazione più avanzata applicata allo spettro dell'audio ed infine il suo comportamento nella compressione della dinamica.

Il capitolo 2 tratta della conoscenza finora raggiunta sulla digitalizzazione dei segnali e sulla dinamica, il cosiddetto "stato dell'arte". Verrà quindi esplicata la funzione dei compressori e degli espansori, che vanno a modificare la gamma della dinamica.

Il capitolo 3 spiega in modo dettagliato l'algoritmo ACE, le sue funzioni e una sua prima implementazione tramite il linguaggio di programmazione Java a livello "mono-dimensionale", cioè lavorando sulla forma d'onda dei segnali audio.

Nel capitolo 4 verrà descritta l'applicazione dell'algoritmo allo spettro FFT utilizzando un ambiente per il calcolo numerico molto usato in ambito audio, cioè MatLab. Saranno inoltre presentati i risultati in forma grafica oltre ad un'analisi percettiva dei file audio ottenuti in output.

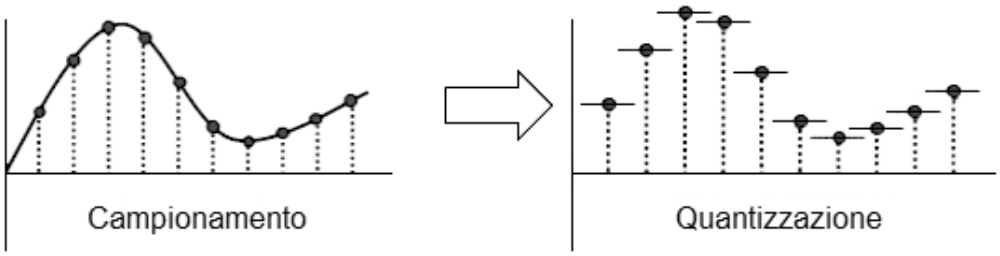
Il capitolo 5 racchiude la prima vera applicazione dell'algoritmo a livello di dinamica. Modellando inviluppo e ampiezza RMS dei campioni audio è stato possibile comprimere il suono e notare gli effetti che ACE ha in questo genere di applicazioni.

Infine il capitolo 6 è riservato alle conclusioni e alle indicazioni per poter migliorare l'utilizzo dell'algoritmo in futuro.

Capitolo 2

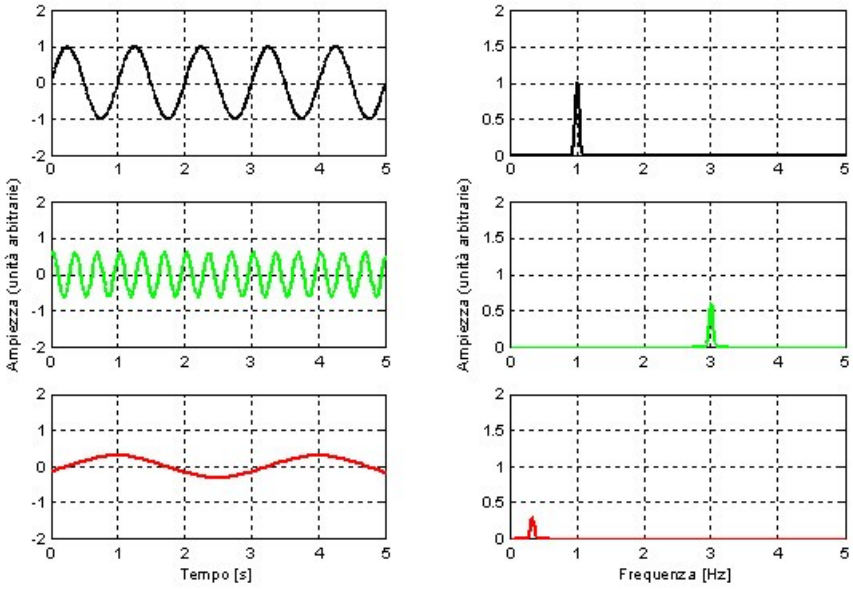
Segnali digitali e dinamica

Un segnale digitale, ovvero a tempo discreto $f(n\tau)$, si ottiene tramite il campionamento di un segnale analogico, cioè a tempo continuo, con passo di campionamento τ fissato. I campioni sono ricavati a istanti che sono multipli interi $n\tau$ di τ, con $-\infty < n < \infty$.
In sostanza, la conversione di un segnale analogico in un segnale digitale ci porta a descrivere un'onda come una sequenza di numeri, che rappresentano il valore assunto dal segnale ad intervalli regolari di tempo.
Affinché il segnale sia trasmissibile e codificabile con un numero finito di bit, ovvero in forma numerica, è però necessario che esso possa assumere solo un numero finito di valori di codominio discreti; ciò avviene tramite un successivo processo di quantizzazione. Questa operazione in generale introduce un errore irreversibile nel segnale quantizzato: dato il segnale quantizzato non è possibile ricostruire il segnale originale. Tuttavia è possibile controllare la qualità con cui il segnale quantizzato approssima quello analogico: un tipico indice di qualità è il rapporto segnale-rumore SQNR. Inoltre viene introdotto del rumore causato dalla riduzione del numero di bit, il quale si può mascherare grazie al "dithering" che rende aperiodici gli errori di arrotondamento.



2.1 La rappresentazione dello spettro

I principali metodi di analisi dei segnali possono essere riassunti nei concetti di "analisi nel dominio del tempo" e "analisi nel dominio della frequenza". Questi due modi di affrontare un problema sono tra loro intercambiabili, infatti, sotto opportune condizioni, nessuna informazione viene persa nel passare da un dominio all'altro. Il vantaggio che deriva dall'introduzione dei due domini è la possibilità di cambiare la prospettiva con la quale si osserva un dato fenomeno. In questo modo un problema che appare di difficile soluzione in un dominio può risultare molto più semplice nell'altro.



Lo strumento matematico che consente di trasferire lo studio dei segnali e dei sistemi dal dominio del tempo al dominio della frequenza è la trasformata di Fourier.

La trasformata di Fourier $X(f)$ di una funzione continua nel tempo $x(t)$ è data dalla relazione:

$$X(f) = \int_{-\infty}^{+\infty} x(t) e^{-i2\pi f t} dt$$

La trasformata di Fourier consente quindi di rappresentare un segnale continuo come somma di infiniti esponenziali periodici pesati da $X(f)df$. Tuttavia non può essere valutata per tutti i valori possibili della variabile continua f, perché questo significherebbe eseguire un numero infinito di volte i calcoli necessari per la sua implementazione. D'altra parte, dal punto di vista della conoscenza dell'informazione sullo spettro di un segnale campionato e troncato (quindi caratterizzato da N numeri), sarebbe strettamente sufficiente conoscere l'andamento dello spettro solo nell'intervallo di ripetizione in frequenza $(0 \div f_s)$. La trasformata discreta di Fourier (Discrete Fourier Transform, DFT) consente di valutare il contenuto armonico in tale intervallo mediante un numero N di componenti discrete.

$$X_k = \sum_{i=0}^{N-1} x_n e^{-ik\frac{2\pi}{N}n}$$

dove k=[0, N-1], i è l'unità immaginaria e $e^{-ik\frac{2\pi}{N}n}$ è una radice dell'unità primitiva N-esima. Nonostante i calcoli richiesti siano la ripetizione di semplici operazioni matematiche, qualora aumenti il numero di dati sui quali si deve operare, la pesantezza computazionale diviene elevata. Nel 1965 due ricercatori, Cooley e Tuckey, ridussero drasticamente il numero di operazioni matematiche richieste per eseguire la trasformata di Fourier attraverso lo sviluppo di un algoritmo oggi denominato Fast Fourier Transform (FFT). Per renderci

conto di quale sia l'ammontare del risparmio di operazioni tra la trasformata "classica" e quella "veloce" assumiamo che il segnale temporale in osservazione sia composto da N campioni. Utilizzando il metodo convenzionale il numero di operazioni matematiche da effettuare è pari a N^2, mentre usando l'algoritmo FFT il costo computazionale è ridotto a $\left(\frac{N}{2}\right) * log_2 N$. Ciò non costituisce una grande differenza per valore N piccolo, ma non appena N cresce tale differenza diviene molto significativa. [1]

$$spettro = fft(segnale)$$

L'ingresso è la sequenza di N valori campionati, mentre l'uscita è una sequenza di N numeri complessi.

Dal momento che ogni numero complesso consta di parte reale ed immaginaria (si hanno quindi in totale $2N$ numeri reali), e non è possibile ottenere informazioni maggiori rispetto a quelle di partenza (N), solo metà dei numeri restituiti avranno un significato. L'altra metà sarà ridondante. Schematizzando:

- Solo le componenti a frequenza positiva o nulla hanno significato fisico e quindi si trascurano quelle corrispondenti a frequenze negative.
- Per non perdere il contributo energetico associato alle componenti a frequenza negativa si moltiplicano per 2 le componenti a frequenza positiva (ciò non viene fatto per la componente a frequenza nulla poiché essa non ha una componente duale).
- Il comando fft non svolge la divisione per N che va dunque fatta dall'utente per normalizzare le ampiezze dello spettro.

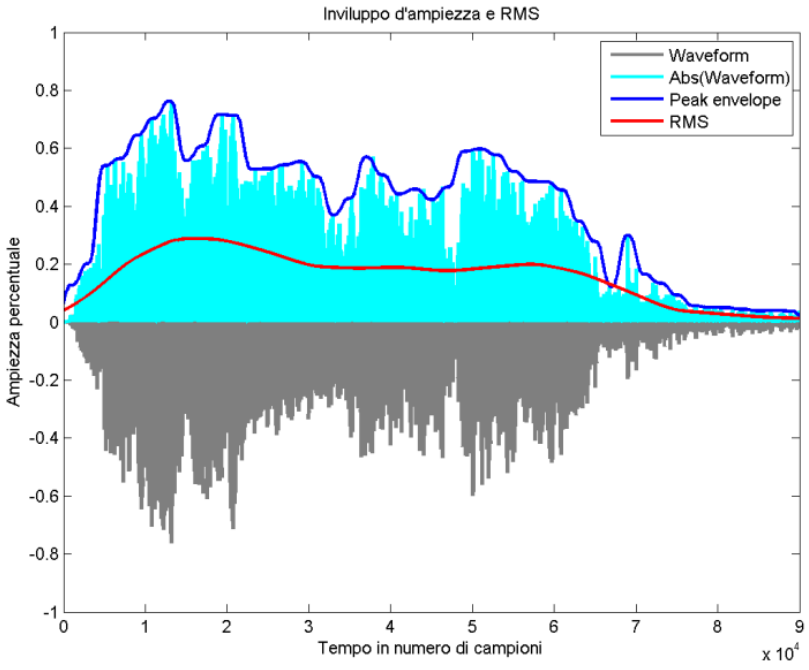
2.2 Il fattore di cresta

Il profilo del valore assoluto della forma d'onda di un segnale viene chiamato inviluppo e rappresenta il valore di picco del suono raggiunto nell'intorno di un dato istante. Esso quindi generalizza il concetto di ampiezza costante.

L'inviluppo consente di avere certamente un'idea dell'escursione dinamica di un suono, ma con forme d'onda complesse viene utilizzata generalmente l'ampiezza RMS (Root Mean Square) che è inequivocabile ed è significativa dal punto di vista fisico. La potenza media trasmessa da un'onda acustica è infatti proporzionale al quadrato dell'ampiezza RMS e non dell'inviluppo.

$$x_{rms} = \sqrt{\frac{1}{n}\sum_{i=1}^{n} {x_i}^2}$$

dove n è la finestra temporale.



Forma d'onda, inviluppo e ampiezza RMS di un segnale audio.

Dal rapporto tra inviluppo e RMS otteniamo il "fattore di cresta" che evidenzia come i picchi estremi sono posizionati nella forma d'onda:

$$C = \frac{|x|_{peak}}{x_{rms}}$$

Un fattore di cresta di 1 ad esempio indica che non ci sono picchi. Il rapporto potenza media/potenza di picco ($PAPR$), definito in [2], è dato dal quadrato dell'inviluppo (potenza di picco) diviso il quadrato del valore RMS (potenza media):

$$PAPR = \frac{|x|_{peak}{}^2}{x_{rms}{}^2} = C^2$$

Espressi in decibel, fattore di cresta e $PAPR$ si equivalgono a causa del modo in cui vengono calcolati i decibel per rapporti di potenza e rapporti d'ampiezza.

Grazie ad essi possiamo avere una stima della dinamica del materiale analizzato. Inoltre, in base alla dimensione della finestra temporale su cui viene calcolato l'RMS, si può parlare di dinamica a breve termine (finestre più piccole di una battuta musicale) o a lungo termine (finestre di alcune battute).

2.3 Impulso unitario e convoluzione

Un importante segnale è l'impulso unitario $\delta(n)$, che rappresenta nel tempo discreto ciò che l'impulso rappresenta nel tempo continuo. Tale segnale vale 1 per $n = 0$, in caso contrario vale 0:

$$\delta(n) = \begin{cases} 1, & n = 0 \\ 0, & n \neq 0 \end{cases}$$

Il comportamento di un sistema che è contemporaneamente lineare e tempo-invariante, è univocamente individuato dalla risposta del sistema al segnale impulsivo.
La convoluzione è una proprietà che ci permette di combinare due segnali $x(n)$ e $y(n)$ per ottenerne un terzo. Nel dominio del tempo tale convoluzione equivale al prodotto dei loro spettri (si veda [3]):

$$\sum_{k=-\infty}^{+\infty} x(k)y(n-k) \;\leftrightarrow\; X(z)Y(z)$$

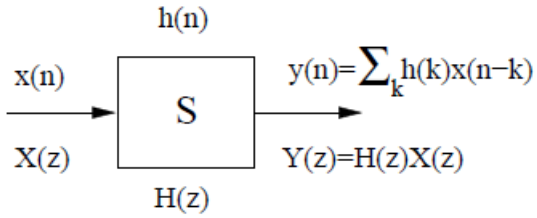
Dato un sistema lineare tempo-invariante a tempo discreto, l'uscita $y(n)$ del sistema che ha come ingresso il segnale $x(n)$ si ottiene dalla convoluzione di $x(n)$ con $h(n)$, dove $h(n)$ è la risposta del sistema all'impulso unitario $\delta(n)$:

$$y(n) = h(n) * x(n) = \sum_{K=-\infty}^{+\infty} h(k)x(n-k)$$

Grazie alla proprietà di convoluzione otteniamo che:

$$Y(z) = H(z)X(z)$$

dove $H(z)$, cioè la trasformata di z della risposta all'impulso $h(n)$, è detta "funzione di trasferimento del sistema".

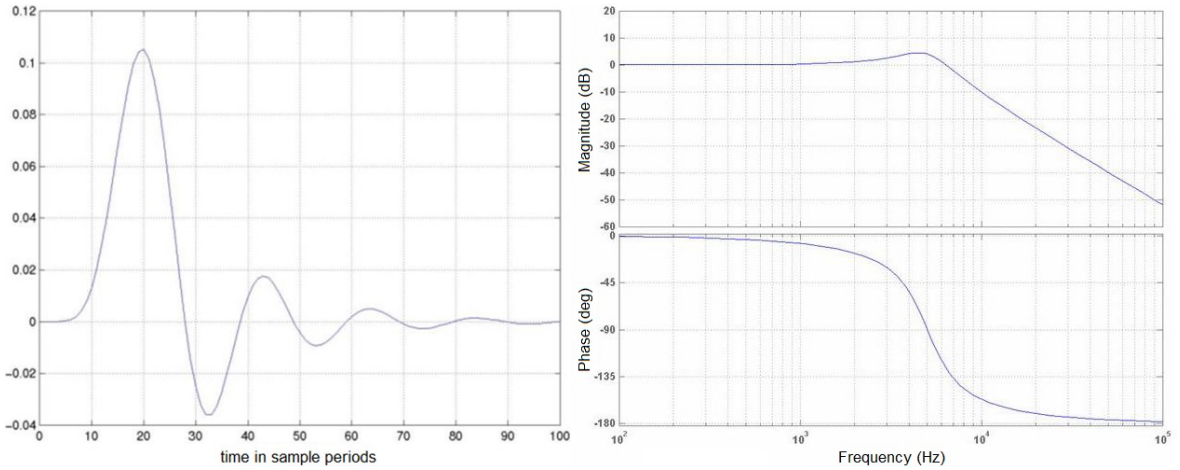


Funzione di trasferimento di un sistema S.

La conoscenza della funzione di trasferimento di un sistema lineare tempo-invariante permette di determinare molte proprietà del sistema, tra cui la stabilità e la risposta in frequenza. Perciò è importante sviluppare metodi per il calcolo della funzione di trasferimento.

2.4 I filtri

I filtri sono sistemi che alterano la magnitudine e la fase di ogni componente in frequenza di un segnale. In genere sono sistemi lineari che si possono descrivere grazie alla loro risposta all'impulso.



Risposta all'impulso di un filtro passa basso nel dominio del tempo e della frequenza.

Possiamo classificare i filtri in diversi modi a seconda di:

- **Tipologia di risposta (finita o infinita)**
  Un sistema lineare tempo-invariante a tempo discreto è detto filtro FIR (Finite Impulse Response) se la risposta $h(n)$ all'impulso unitario è finita nel senso che $h(n) = 0$ per $n < 0$ e per $n \geq M$, per un opportuno $M > 0$. Ricordando che l'uscita è ottenuta dalla convoluzione dell'ingresso con la risposta all'impulso, la relazione ingresso-uscita è data dalla seguente equazione alle differenze finite:

$$y(n) = \sum_{k=0}^{M-1} h(k) \cdot x(n-k)$$

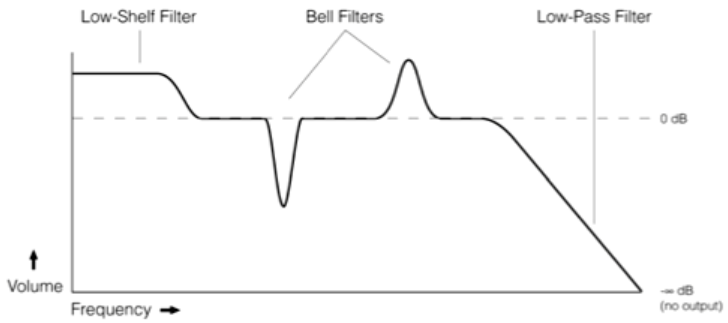
  Le caratteristiche più interessanti dei filtri FIR sono il fatto di essere sempre stabili e causali (la relazione ingresso-uscita permette di individuare univocamente la risposta del filtro all'impulso) e la possibilità di avere fase lineare.
  Un filtro lineare in cui la risposta all'impulso non è finita viene detto filtro IIR (Infinite Impulse Reponse). In questo caso, detta $h(n)$ la risposta all'impulso, la relazione ingresso-uscita è data da:

$$y(n) = \sum_{k=0}^{\infty} h(k) \cdot x(n-k)$$

Una sottoclasse particolare di filtri IIR sono i filtri ricorsivi, che ammettono una relazione ingresso-uscita del tipo:

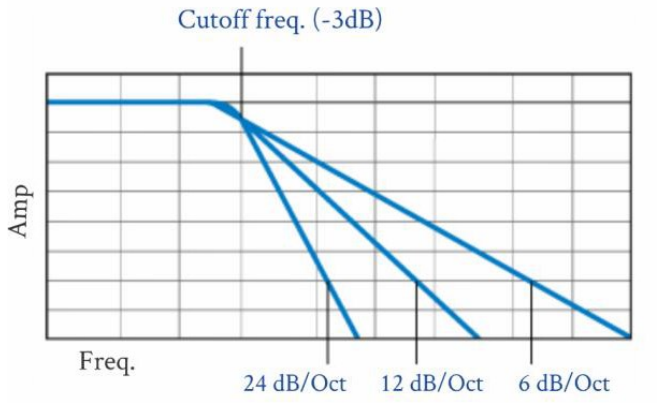
$$y(n) = \sum_{k=1}^{D} a(k) \cdot y(n-k) + \sum_{k=0}^{M} b(k) \cdot x(n-k)$$

- **Tipologia di curva**
  L'applicazione di un filtro comporta un'alterazione che si può identificare tramite l'andamento delle curve. In genere si utilizzano low-pass (passa basso), high-pass (passa alto), low-shelf, high-shelf e bells.



Funzione trasferimento in cui si possono identificare tre delle tipologie di curve conseguenti all'applicazione di filtri.

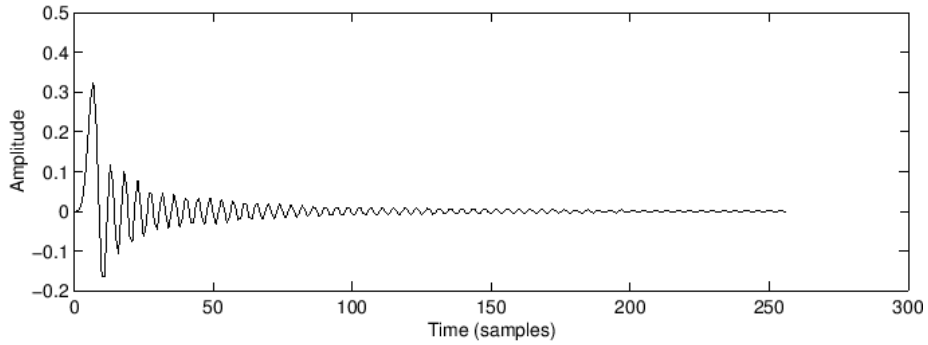
- **Pendenza**
  La pendenza di un filtro indica la quantità di attenuazione in funzione della distanza dalla frequenza di taglio. Identificando il decibel (dB) come unità di misura dell'intensità acustica e l'ottava come lo spazio che intercorre fra una data frequenza e il suo doppio o la sua metà, un filtro passa basso del primo ordine ha una pendenza di -6 dB per ottava, il che significa che un'ottava oltre la frequenza di taglio l'attenuazione sarà di -6 dB, 2 ottave oltre la frequenza di taglio l'attenuazione sarà di -12 dB, 3 ottave oltre la frequenza di taglio sarà di -18 dB, e così via.



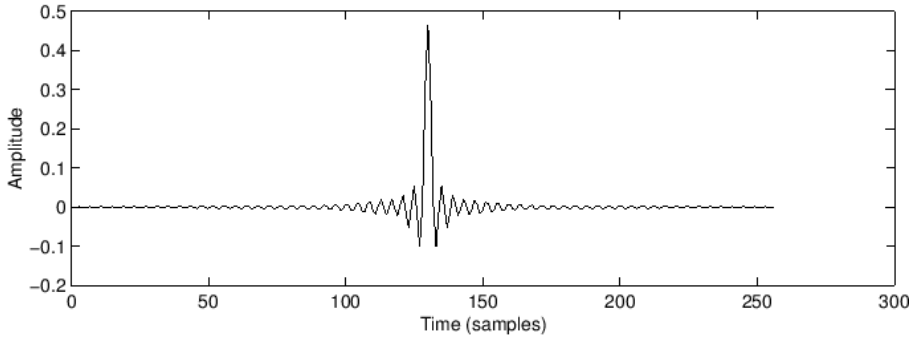
Filtro passa basso in cui la pendenza aumenta all'aumentare dell'ordine del filtro.

- **Implementazione**
  Esistono diversi modi di calcolare i coefficienti h_{0-k}, a_{0-k} e b_{0-k} dei filtri FIR ed IIR, attraverso cui si possono ottenere diversi comportamenti. Alcune implementazioni garantiscono una linearità nella banda passante a discapito della banda tagliata o viceversa, altre garantiscono una banda di transizione più stretta, cioè una pendenza più alta a discapito della linearità generale o viceversa. I metodi di implementazione di un filtro più usati in ambito audio sono i Butterworth e i Chebyshev di tipo 2.

- **Risposta in fase**
  Oltre ad alterare la magnitudine dello spettro, i filtri introducono anche un ritardo nella fase del segnale. Sostanzialmente continuano ad oscillare anche dopo che l'impulso è terminato. I filtri detti "non lineari in fase" introducono ritardo solo per le frequenze nei dintorni della frequenza di taglio, mentre con i filtri "lineari in fase" tutte le componenti di frequenza hanno un uguale ritardo nel tempo. In molte applicazioni, questo ritardo di gruppo costante è vantaggioso. Per contro, un filtro non lineare in fase ha un ritardo di gruppo che varia con la frequenza, portando come risultato ad una distorsione di fase.



Risposta di un filtro non lineare in fase.



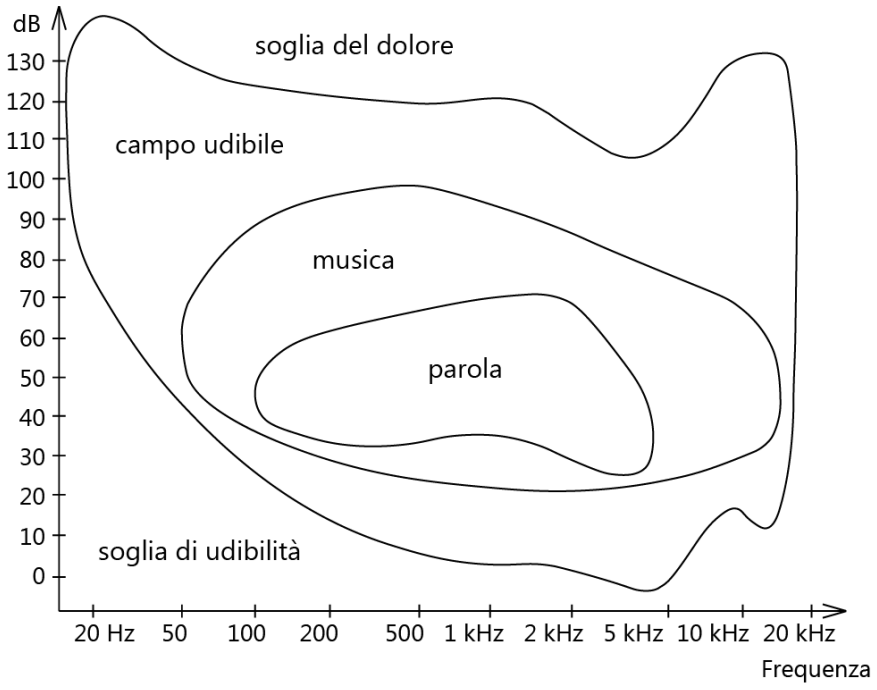
Risposta di un filtro lineare in fase

2.5 Il decibel

Un segnale può essere definito come la variazione di una grandezza nel tempo. Nel caso dei segnali audio questa grandezza è la pressione data dalle vibrazioni del mezzo attraverso cui si propaga il suono. Più le variazioni sono ampie, più il suono sarà intenso.
Le frequenze delle vibrazioni acustiche normalmente udibili dagli esseri umani, spaziano da circa 20 Hz a circa 22000 Hz. Per indicare il livello di pressione sonora (SPL=Sound Pressure Level) in acustica si utilizza il dB_{SPL} e si calcola nel seguente modo:

$$dB_{SPL} = 20 \cdot log_{10} \frac{P_{RMS}}{P_{Ref}}$$

P_{Ref} indica la pressione sonora corrispondente alla soglia di udibilità, pari a 29,7 µPa.
In realtà la soglia di udibilità dipende dalla frequenza del segnale. Attraverso lo studio delle curve isofoniche è possibile avere un'idea della gamma dinamica e del range di frequenze che un essere umano dall'udito sano può realmente percepire.
In ambito digitale, tuttavia, è più appropriato esprimere la pressione sonora come dB_{FS} (decibel full scale), cioè in relazione all'intervallo massimo reso disponibile dal numero di bit utilizzati. Lo 0 rappresenta il più alto valore possibile. Tutte le misurazioni espresse in dB_{FS} saranno quindi numeri negativi.



2.6 La gamma dinamica

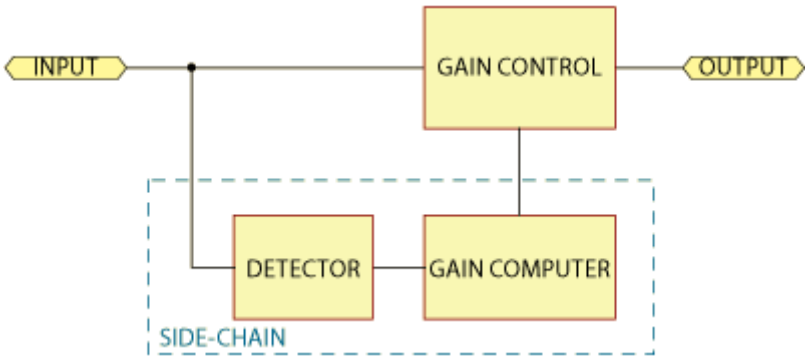
Il range dinamico di una traccia audio è dato dal rapporto tra il segnale più forte (full scale) e il segnale più silenzioso (rumore di fondo RMS), che teoricamente è causato dal rumore di quantizzazione.

$$DR = SNR = 20 \cdot log_{10}\left(2^n \sqrt{\frac{3}{2}}\right) \approx 6.0206 \cdot n + 1.761$$

Nel caso dell'audio con n=16 bit avremo quindi un range dinamico di 98.09 dB. Le ampiezze superiori a 0 causeranno una distorsione del segnale e quelle sotto a -98.09 dB_{FS} non saranno codificate.

I processori di dinamica alterano un segnale audio sulla base del suo contenuto di frequenze e dell'ampiezza dell'onda; da qui deriva il termina "dinamica" in quanto è un processo che cambia sempre.

I quattro effetti più comuni che modificano la dinamica sono i compressori, i limitatori, i gate e gli expander. Oltre a questi vi sono una gran quantità di processori special purpose: unità AGC, ducker, de-esser, leveler, feedback suppressor, exciter ed enhancer. Non importa quale sia il nome, tutti rientrano negli strumenti per il controllo automatico del volume o delle dinamiche di suono. Come illustrato in [4], tutti i processori di dinamica hanno una struttura comune: un controllo di guadagno (gain control) sul percorso del segnale principale ed una catena laterale contenente un rilevatore (detector) e un gain computer. Il rilevatore può campionare il segnale prima del controllo di guadagno o dopo di esso, ma per semplicità qui sarà mostrato il controllo fatto prima.



Struttura comune dei processori di dinamica.

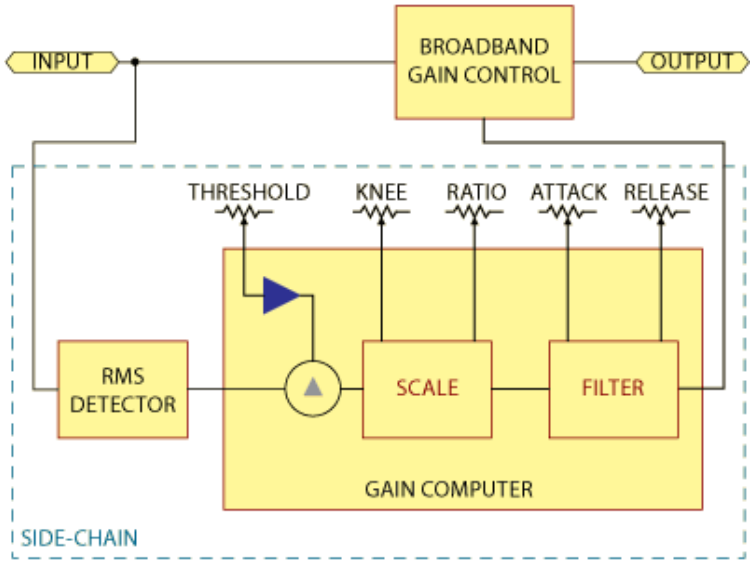
Il DSP (Digital Signal Processor) è un processore dedicato e ottimizzato per eseguire in maniera estremamente efficiente sequenze di istruzioni ricorrenti (come ad esempio somme, moltiplicazioni e traslazioni) nell'elaborazione di segnali digitali. Grazie ad esso è cambiata radicalmente la realizzazione di processori di dinamica. Nell'analisi di segnali analogici tradizionali non esiste il "guardare avanti" o l'analisi statistica del contenuto del segnale, ma solamente una funzione che risponda ad eventi già verificatisi.

2.7 I compressori

I compressori riducono il range dinamico del segnale che passa attraverso di essi, abbassando i segnali più forti in modo dinamico. Un compressore inizia ad agire abbassando il segnale di una quantità fissata dal ratio control quando il segnale in ingresso supera il livello fissato dal controllo di soglia (threshold).
In ambito digitale sono usati complessi algoritmo matematici ottimizzati per la dinamica della musica e dei segnali vocali. Prima dei compressori si utilizzavano mixer controllati da una persona in carne ed ossa, con l'atto di monitoraggio e regolazione costante chiamato "gain riding".
Il motivo principale di questa esigenza di compressione è la difficoltà che gli impianti audio hanno nel gestire la gamma completa di suoni. Infatti alzando il volume per poter sentire i segnali medi, insieme ad essi si presentano enormi picchi, che sono vitali per la musica, ma troppo grandi per essere gestiti dagli amplificatori e dagli altoparlanti. Ciò crea il cosiddetto "clipping", cioè una distorsione della forma d'onda che avviene quando l'amplificatore è sovrapilotato e tenta di rilasciare una tensione oltre la sua capacità massima. Una soluzione pensabile sarebbe quella di ridurre il volume per evitare sovraccarichi, ma ciò provocherebbe in passaggi dolci e quieti un abbattimento drastico del segnale vocale, che consentirebbe di udire poco o niente.
Per risolvere questo problema quindi viene utilizzato un compressore. Esso si utilizza semplicemente impostando un punto di soglia, oltre il quale tutto sarà abbassato di una certa quantità e quindi selezionare il rapporto (ratio) per definire quanto una certa quantità è in decibel. L'audio prima della soglia rimarrà invariato. Ad esempio per passare da 110 dB a 70 dB è necessario impostare il rapporto di compressione a 1.6:1 ($\frac{110}{70} = 1.6$).

2.8 La compressione broadband

La più semplice forma di compressione è quella broadband, cioè a banda larga, dove tutte le frequenze vengono compresse equamente e il side-chain è sensibile in modo uguale a tutte le frequenze. Si utilizza tipicamente un rilevatore RMS e i controlli base per controllare il guadagno sono threshold, ratio, attack e release.



Struttura di un compressore broadband.

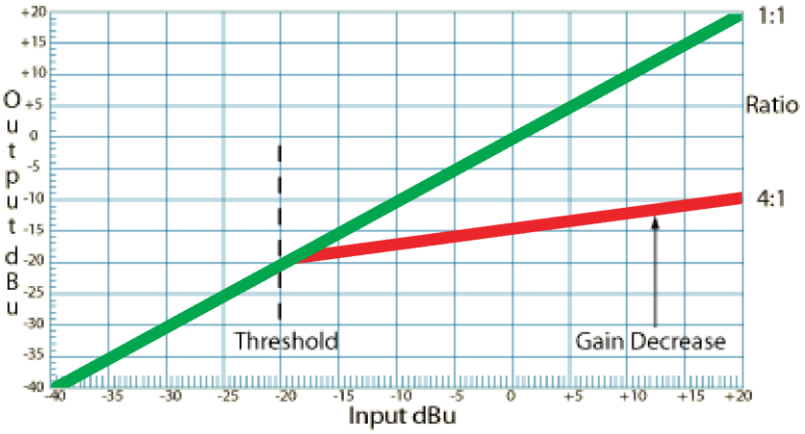
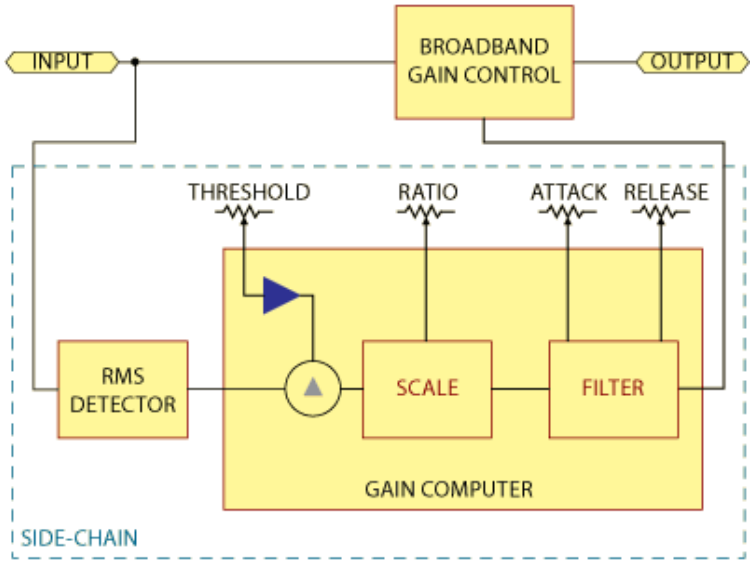


Grafico della risposta a un compressore broadband con threshold impostato a -20 dB e ratio a 4:1.

2.9 L'expander

L'expander è un utile complemento al compressore, in quanto espande il range dinamico del segnale che vi passa attraverso. Funge quindi come una sorta di compressore al contrario; ad esempio, passando un input compresso con una gamma dinamica di 70 dB ad un expander, questo potrà produrre un output da 110 dB. Il controllo di guadagno quindi svolge un lavoro diverso con la differenza tra la soglia e il livello del segnale in ingresso.

A differenza del compressore, l'expander riduce il guadagno dei segnali sotto la soglia. Così i suoni silenziosi risulteranno ancora più silenziosi, mentre come detto il compressore riduce i suoni troppo rumorosi che superano la soglia impostata.



Struttura di un expander.

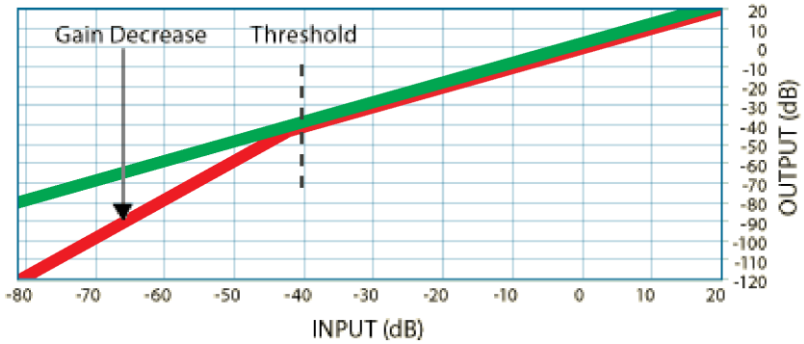


Grafico della risposta a un expander.

2.10 I controlli side-chain

I quattro principali parametri che controllano l'attività del circuito side-chain, come detto, sono threshold, ratio, attack e release. Molti controllori di dinamica offrono una regolazione manuale di questi parametri, mentre altri li impostano in automatico a valori ottimali.

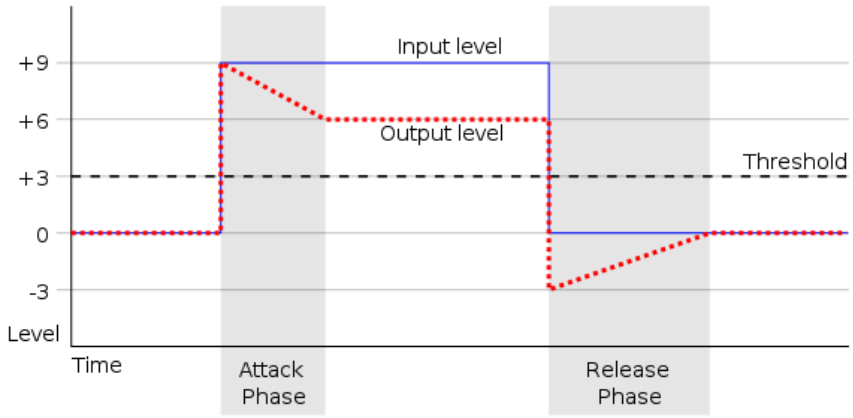
- **Threshold**
  È il punto di inizio della regolazione del guadagno. Quando il segnale di ingresso supera la soglia per i compressori, o è al di sotto della soglia per gli expander, il circuito di controllo side-chain inizia ad agire.
- **Ratio**
  Quando il segnale eccede il threshold, la variazione del volume dipende dal ratio. Un rapporto 1:1 implica che a un input con variazione di 3 dB seguirebbe un output con

variazione di 3 dB. Un ratio rigido è 10:1 il che significa che per un cambiamento di 10 dB all'ingresso vi è solo un cambiamento di un decibel all'uscita, il che implica un'elaborazione pesante.

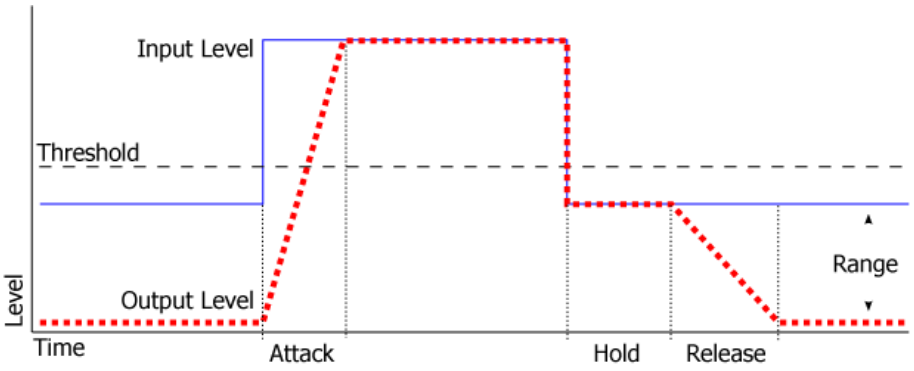
- **Gain**
  A volte vi si riferisce come "make-up gain". Controlla che l'output abbia il livello di volume desiderato con la compressione attiva. Il range preferito è tra -12 e 12 dB con una concentrazione maggiore intorno a 0 dB. Il guadagno può essere applicato direttamente al segnale principale o nel side-chain come controllo offset.

- **Attack**
  Indica quanto rapidamente la funzione risponde ad un aumento del livello di input oltre la soglia. Per i compressori ciò definisce quanto velocemente il guadagno è abbassato. Per expander e gate indica quanto velocemente il guadagno viene alzato.
  I tempi di attacco per i compressori in genere oscillano tra 25 ms e 500 ms. Per gli espansori questo range cambia tra 0 ms (attack istantaneo) e 250 ms, poiché oltre non è necessario.

- **Release**
  Definisce quanto velocemente la funzione risponde a un decremento nel livello di ingresso al di sotto della soglia. Per i compressori indica quanto rapidamente il guadagno viene rialzato dopo che questa elaborazione del segnale è finita. Invece per gate ed expander indica quanto velocemente il volume viene riabbassato.
  Valori di release tipici per compressori ed espansori sono tra i 25 ms e i 2 secondi.



Comportamento di un compressore nel tempo.



Comportamento di un gate nel tempo.

- **Knee**
  Nei compressori questa funzione controlla l'azione al punto di threshold. Un knee rigido (hard) non agisce finché il segnale un supera la soglia, per poi applicare una compressione completa. Un knee leggero (soft) invece riduce le distorsioni causate dalle transizioni brusche del passaggio tra segnale con ampia dinamica e segnale compresso.
  Mentre con metodi analogici è difficile stabilire una risposta accurata al knee leggero, implementazioni digitali consentono di porre il centro del knee esattamente sulla soglia con una funzione che crea una transizione graduale dal guadagno originario al rapporto specificato.
  Il soft knee possiede inoltre un parametro "span" che definisce quanti decibel sotto la soglia inizia la compressione e quanti decibel sopra la soglia viene raggiunto il rapporto di compressione specificato. Impostando un valore alto di knee si potrà applicare una compressione leggera che incrementerà fino a raggiungere il punto di threshold per poi continuare fino all'applicazione della compressione totale ai suoni di alto livello.
  Ad ogni modo se si richiede un alto livello di rumore giusto prima della compressione è meglio utilizzare un knee rigido.

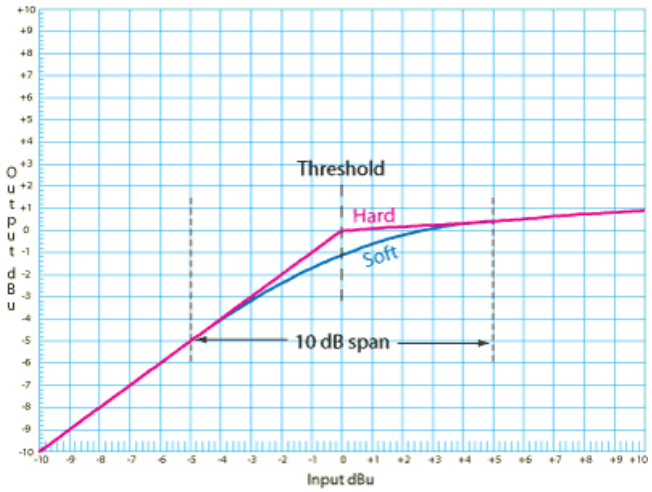
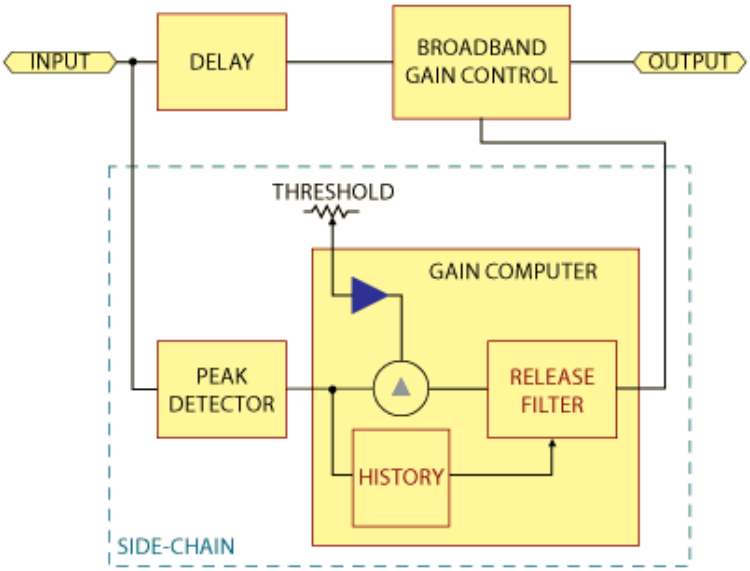


Grafico di un compressore con knee regolabile.

2.11 Il limiter

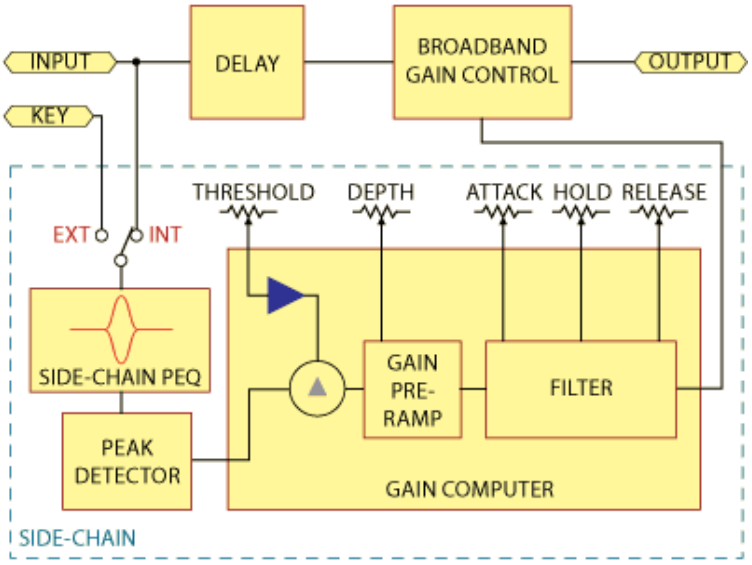
Il limitatore di picco viene generalmente applicato alla traccia master per evitare distorsioni dovute al clipping del segnale in uscita. Nonostante un compressore possa essere utilizzato come limitatore, impostando il rapporto ad un valore elevato come 10:1 e usando un tempo attack corto e release lungo, ciò non garantisce che il segnale non supererà la soglia con grandi valori. Per prevenire questa situazione il limiter offre un tempo "look-ahead" che assicura che il tempo di attack avrà raggiunto il suo valore obiettivo prima di agire sul picco. Ciò introdurrà una latenza del segnale in uscita, determinata dal parametro look-ahead.
Quindi, mentre il limiter agisce come un livellatore rigido sul segnale, il compressore svolge questo compito più gradualmente.



Struttura di un limiter con look-ahead.

2.12 Il gate

Il gate è per l'expander ciò che il limiter è per il compressore. Come l'expander infatti riduce il guadagno sotto una determinata soglia e come il limiter deve rispondere molto rapidamente ai cambiamenti di livello utilizzando un rilevatore di picco.
A differenza dell'expander però il rapporto è fisso a ∞:1 e viene utilizzata una intensità (depth) che determina quanti decibel il segnale viene attenuato quando è sotto il valore di threshold. Il gate generalmente è usato per rimuovere il rumore di fondo tra suoni elevati.



Struttura di un gate.

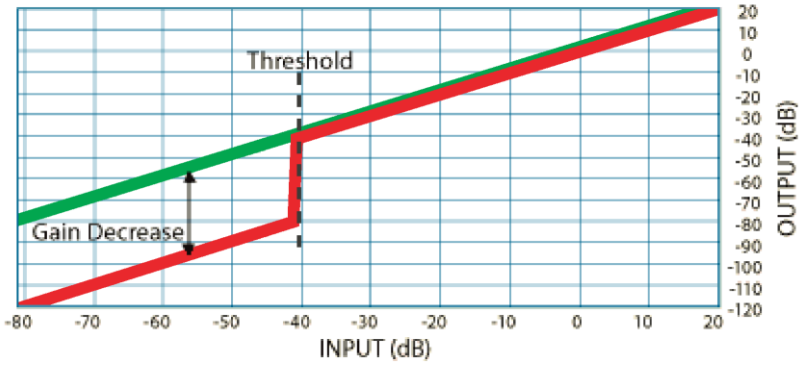


Grafico della risposta a un gate.

2.13 La compressione multibanda

Un'altra forma di compressione è quella multibanda che divide il segnale ricevuto in input in diverse bande di frequenze (da 2 a 6 in base al modello) e ne processa ognuna in modo indipendente. Quindi i controlli side-chain saranno ripetuti per ogni banda, rendendo facile uniformare le diverse frequenze dello spettro e scegliendo un attack e un release appropriato per ciascuna. Ad ogni modo tutto ciò può rendere il suono poco realistico a causa della disgiunzione delle diverse bande di frequenze. Perciò sarà necessario un intervento postumo di ricompressione per riconsolidare la dinamica dopo l'ottimizzazione multibanda.

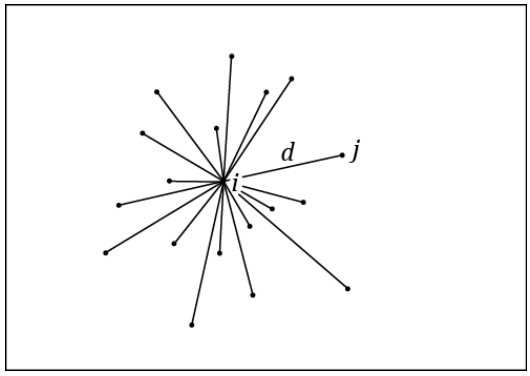
Capitolo 3

Un primo approccio all'algoritmo ACE

Per migliorare la dinamica dell'audio si è scelto di utilizzare un algoritmo denominato ACE, sigla per Automatic Color Equalization. Tale algoritmo è stato originariamente concepito per l'adattamento cromatico/spaziale delle immagini.
Data un'immagine in input, ne viene migliorata la costanza di luminosità e colore, e viene applicato un contrasto controllato. Sostanzialmente ogni singolo pixel viene modificato sulla base del contenuto dell'immagine.

3.1 L'algoritmo

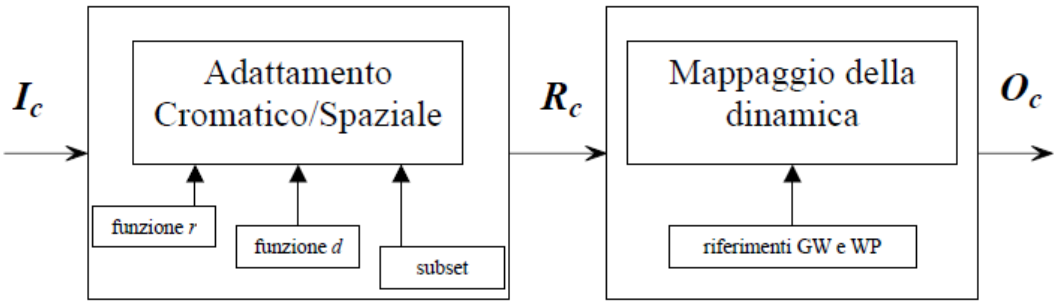
Nello specifico, l'algoritmo ACE agisce in modo tale che il valore di ogni pixel i dell'immagine venga ricalcolato in base ai valori di un insieme di punti j posizionati ad una certa distanza d da i, come descritto in [5].



$$R_c(i) = \sum_{j \in Subset, j \neq i} \frac{r(I(i) - I(j))}{d(i,j)}$$

Si noti che l'algoritmo non richiede nessun tipo di supervisione o di precalcolo statistico. La funzione differenza $r(\cdot)$ fa una stima della luminosità relativa dei pixel, mentre $d(\cdot)$ è una funzione distanza che pesa la quantità dei contributi locali o globali.
Il ricalcolo può essere eseguito prendendo in considerazione tutti pixel dell'immagine oppure un suo sottoinsieme, cioè confrontando ogni singolo punto con quelli che si trovano nei pressi di quest'ultimo. In ogni caso non verrà mai considerato il caso in cui $j=i$ perché si tradurrebbe in un confronto di i con se stesso, quindi inutile.
L'algoritmo prevederebbe inoltre un secondo stadio in cui avviene un mappaggio della dinamica prodotta dal primo stadio in quella dell'immagine finale. Ciò permetterebbe di

compensare l'effetto bordo, cioè il fatto che i pixel vicino al bordo verrebbero ricalcolati su un vicinato non omogeneamente distribuito.

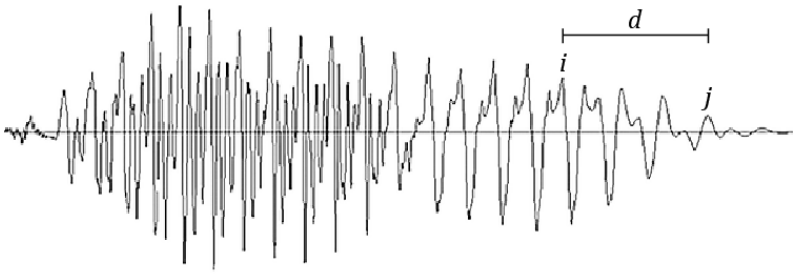


Schema base di ACE. I indica l'immagine in input, R il risultato intermedio mentre O rappresenta il risultato finale; il pedice c denota il canale cromatico (RGB) poiché l'algoritmo viene applicato separatamente sui tre canali cromatici.

All'equazione precedente si introduce quindi un denominatore per bilanciare il filtraggio in funzione della posizione di i.

$$R_c(i) = \frac{\sum_{j \in Subset, j \neq i} \frac{r(I(i) - I(j))}{d(i,j)}}{\sum_{j \in Subset, j \neq i} \frac{r_{max}}{d(i,j)}}$$

r_{max} è il valore massimo assunto dalla funzione $r(\cdot)$.
In ambito audio, per poter applicare l'algoritmo, si procederà prendendo in considerazione un campione i di un'onda, con una determinata ampiezza, ed un insieme di altri punti j della stessa onda, distanti quindi da i nel tempo.



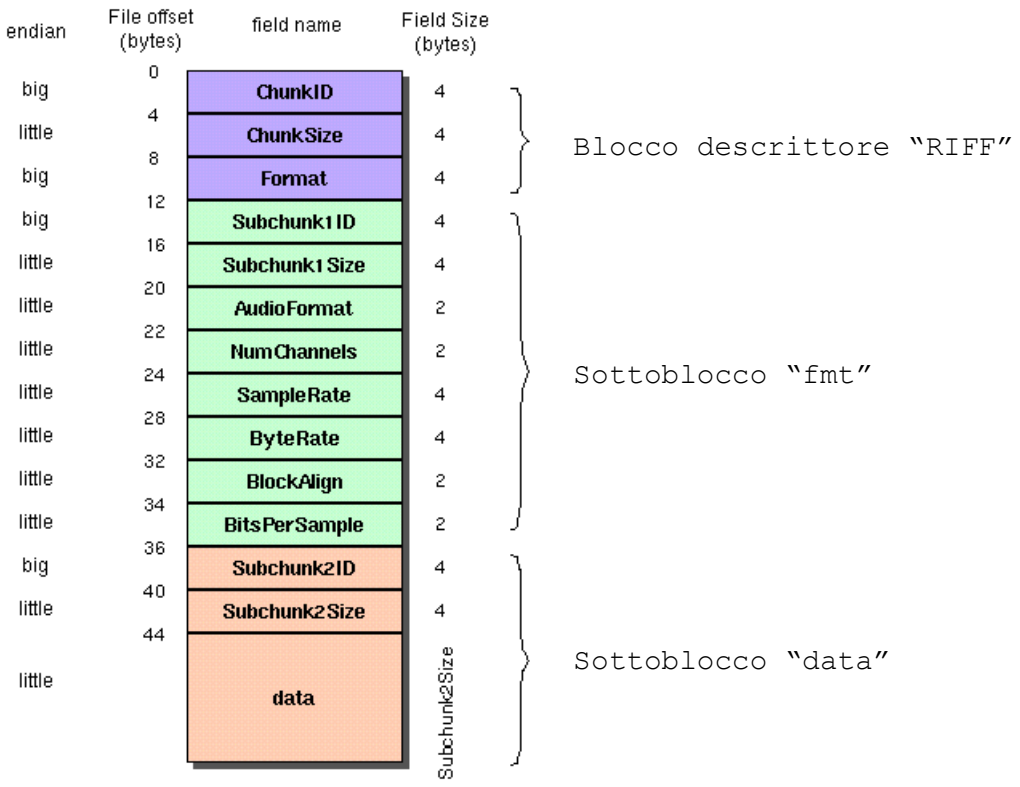
Si passerà così da un caso bidimensionale (l'applicazione ad una matrice) ad uno monodimensionale (l'applicazione ad un array).
La fase di mappaggio finale, essendo poco consona nel caso dell'audio poiché produrrebbe un output distorto e con basso volume, sarà invece sostituita. Si effettuerà infatti una normalizzazione studiata ad hoc per la rappresentazione audio a cui l'algoritmo verrà applicato.

3.2 Introduzione a Java

Java è un linguaggio di programmazione utilizzato per sviluppare applicazioni per una vasta gamma di ambienti, dai dispositivi consumer ai sistemi aziendali eterogenei.
Come ogni linguaggio di programmazione, il linguaggio Java ha una propria struttura, regole di sintassi e un paradigma di programmazione. Il paradigma di programmazione del linguaggio Java si basa sul concetto di programmazione orientata agli oggetti (si veda [6]). Inoltre è un derivato del linguaggio C, quindi le sue regole di sintassi presentano un aspetto molto simile a quello del C.
Strutturalmente, il linguaggio Java inizia con i pacchetti. All'interno dei pacchetti vi sono le classi, e all'interno delle classi vi sono metodi, variabili, costanti, e così via.
I packages Java Sound in particolare permettono di gestire la qualità dell'audio e di riprodurre o memorizzare, in maniera piuttosto semplice, file sonori.

3.3 La struttura di un file wave

Il WAVEform audio file format è un formato audio di codifica digitale sviluppato da Microsoft e IBM. È una variante del formato RIFF di memorizzazione dei dati. I dati vengono salvati in "chunk" ovvero blocchi, come descritto in [7].
Analogamente ai file RIFF, ogni WAVE inizia con un header che descrive la tipologia di file, per poi essere seguito da informazioni sul formato e infine dai dati sui campioni audio.

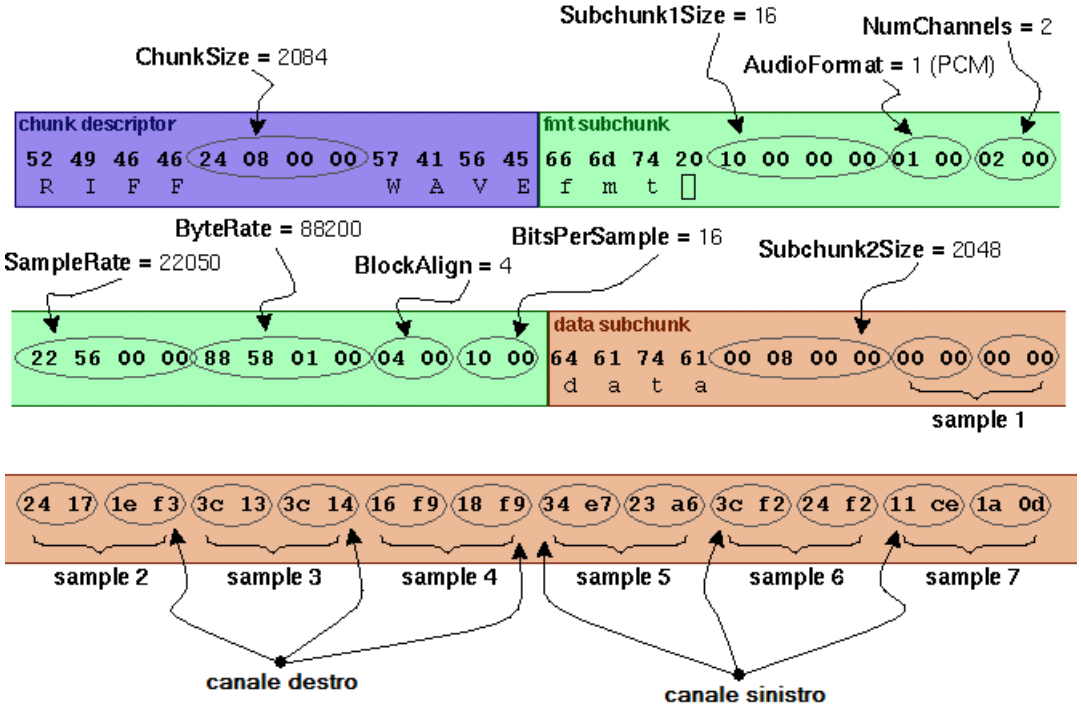


I dati di questo formato, essendo progettato per computer che utilizzano processori Intel o compatibili, vengono memorizzati con la notazione little endian. Poiché basato sullo standard RIFF il formato supporta varie modalità di immagazzinamento dei dati, tuttavia il più diffuso è il metodo PCM.

Il PCM provvede a salvare i dati audio senza nessun tipo di compressione dati, quindi la forma d'onda viene memorizzata così com'è, seppur digitalizzata. Inoltre la registrazione può essere caratterizzata da altri parametri: il numero di bit di codifica (generalmente 8, 16 o 24) e la frequenza di campionamento (11, 22, 44.1, 48, 96 o 192 kHz).

All'interno del blocco "data", dopo le dimensioni dell'informazione, vi sono i sample cioè il suono vero e proprio che, per i file audio stereo, presenta un'alternanza tra campione del canale sinistro e il parallelo del canale destro.

Per chiarire meglio segue un'illustrazione dei primi 72 byte di un file WAVE, qui convertiti in numeri esadecimali.



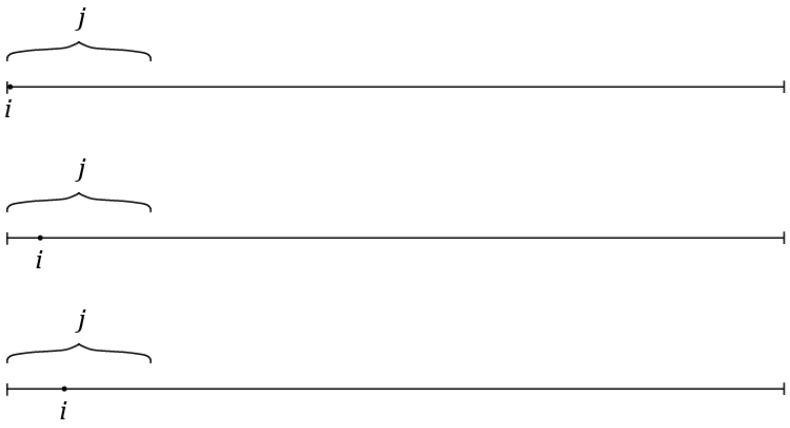
3.4 Implementazione

Per prima cosa è stato ricreato in Java l'algoritmo ACE di base in modo tale che fosse applicabile a file audio mono, quindi con un singolo canale.

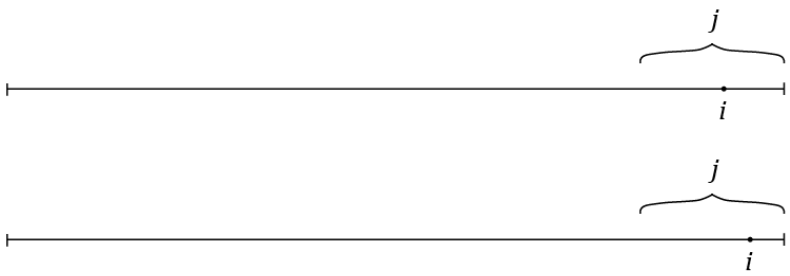
$$\text{out}[i] = \sum_{j \neq i} \frac{\text{in}[i] - \text{in}[j]}{i - j}$$

```
for (int i=0; i<l.length; i++)
      for (int j=0; j<l.length; j++)
            if (j!=i)
                  out[i] += (in[i]-in[j])/(i-j);
```

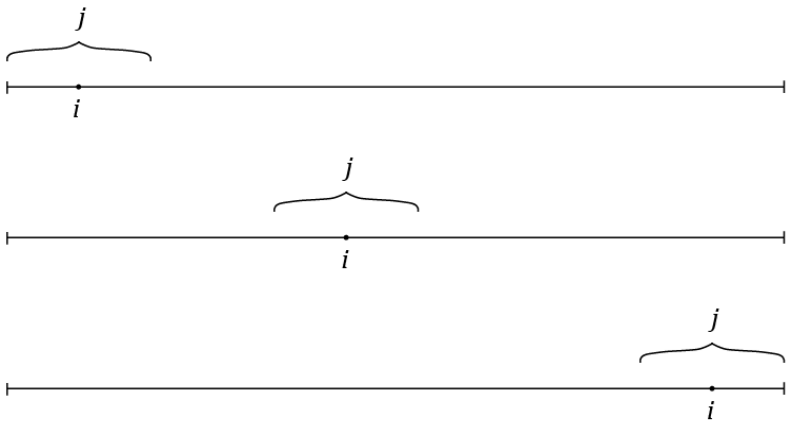
Sono stati quindi aggiunti dei metodi che utilizzano i recenti packages di Java dedicati all'audio in modo che fosse letto un file wave come array di byte, convertito in un array di double ed esportato in un nuovo file wave in seguito all'applicazione dell'algoritmo.
Un problema subito evidente è stato quello che, eseguendo il confronto di ogni campione *i* con tutti gli altri campioni *j* del file audio ricevuto in input, il processo risultava irrimediabilmente lento. È stato quindi ridotto il numero dei valori *j* confrontati con il campione *i* utilizzando una finestra mobile.



Intervallo valori iniziali fisso.



Intervallo valori finali fisso.



Intervallo mobile per tutti i valori centrali.

Finché non si raggiunge la metà dell'intervallo (`range/2`) alle estremità dell'array di valori, la finestra rimane fissa. Da tale valore in poi la finestra si muoverà lungo tutto l'array.

```
for (int i=0; i<in.length; i++) {

      // Intervallo iniziale
      if (i<range/2) {
            for (j=0; j<=range; j++)
                  if (j!=i)
                        out[i] += (in[i]-in[j])/(i-j);
      }
      // Intervallo finale
      else if (i>=in.length-(range/2)) {
            for (j=in.length-(range-1); j<in.length; j++)
                  if (j!=i)
                        out[i] += (in[i]-in[j])/(i-j);
      }
      // Intervallo centrale mobile
      else {
            for (j=i-(range/2); j<i+(range/2); j++)
                  if (j!=i)
                        out[i] += (in[i]-in[j])/(i-j);
      }
}
```

Si è quindi introdotto nel ciclo `for` principale un double, per salvare in esso il massimo valore del nuovo array. L'intero array è poi stato normalizzato dividendolo per il valore massimo precedentemente salvato, in modo da eliminare picchi troppo alti e quindi distorsioni nell'audio.
Infine si è introdotta una lettura dei canali del file audio di input in modo tale che, se il numero di canali fosse stato 2 (stereo), l'algoritmo sarebbe stato applicato a due array distinti, creati secondo quanto visto in [7], per poi essere ricongiunti in un unico array finale.

3.5 La funzione distanza

Dopo alcune verifiche, la distanza precedente $(i - j)$ è stata riscritta come una funzione vera e propria, tramite la chiamata di un metodo che consente di scegliere tra tre diverse distanze: quella base, una distanza euclidea e un'esponenziale inversa.

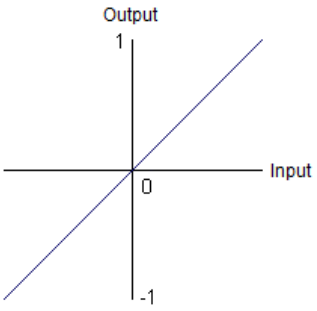
Linear	Euclidean/Absolute	Inverse exponential
d	$\|d\|$	$\frac{1}{e^{-\alpha E d}}$

```
public static double distance(double i, double j, int d, double a){

      // Distanza lineare
      if (d==0)
            return i-j;

      // Distanza assoluta
      else if (d==1)
            return Math.abs(i-j);

      // Distanza esponenziale inversa
      else if (d==2)
            return 1/Math.exp(-a*Math.abs(i-j));

      return 0;
}
```

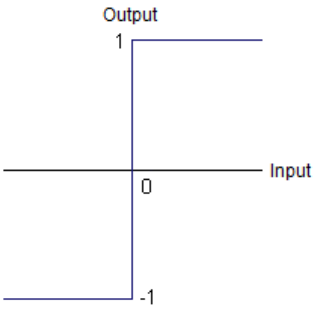
La α nella distanza inverse exponential è un parametro che ha valore compreso tra 0 e 1.

3.6 La funzione differenza

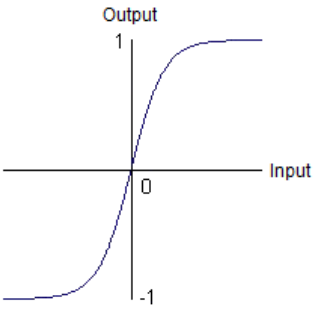
Oltre alla funzione distanza si è scelto di testare diverse varianti della funzione a numeratore nell'algoritmo, cioè la differenza. Sono state utilizzate la versione lineare, una con signum e una versione smussata con tangente iperbolica, precedentemente testata in [8].



Linear



Signum



Hyperbolic tangent

```
public static double difference(double in[], int i, int j, int diff){

      // Differenza lineare
      if (diff==0)
            return in[i]-in[j];

      // Differenza signum
      else if (diff==1)
            return Math.signum(in[i]-in[j]);

      // Differenza tangente iperbolica
      else if (diff==2)
            return Math.tanh(in[i]-in[j]);
      return 0;
}
```

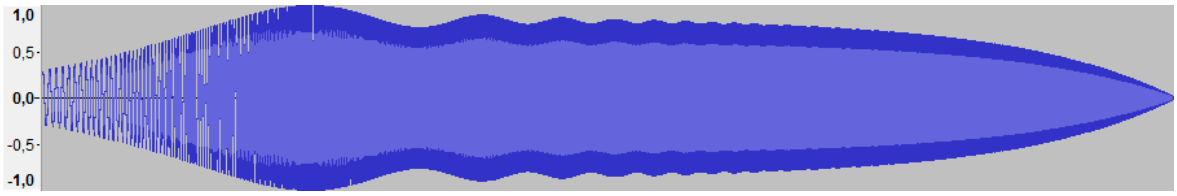
La funzione signum restituisce 0 se l'argomento è 0, 1.0 se è maggiore di 0 e -1.0 se è minore di 0. Essa quindi portava l'ampiezza dell'inviluppo al massimo in tutto il file preso a campione, creando un'audio distorto e non utilizzabile. La variante con tangente iperbolica invece non apportava alcun cambiamento rispetto alla versione lineare.

Queste varianti della funzione differenza si riveleranno inefficaci anche nei test su spettro, inviluppo ed RMS. Per questo motivo si è scelto di concentrare le prove sulla versione con funzione differenza lineare.

3.7 Test sweep

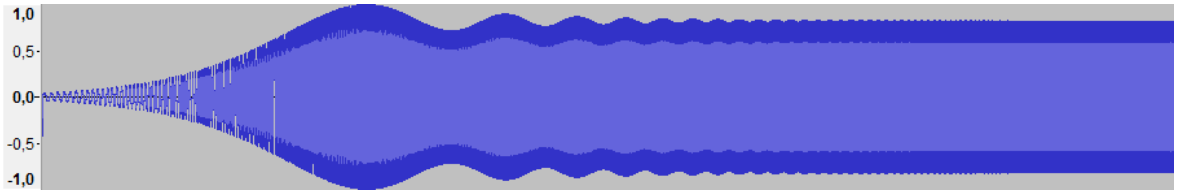
Il primo test è stato effettuato utilizzando come input uno sweep, cioè un suono esteso che partendo dalle basse arriva fino alle alte frequenze.
Come range entro cui confrontare i campioni si è partiti da 100 per arrivare fino a 5000. Il risultato, salendo col numero di campioni, è stato un aumento considerevole delle basse frequenze e in dose più leggera delle medie e delle alte.

Onda sweep originale.

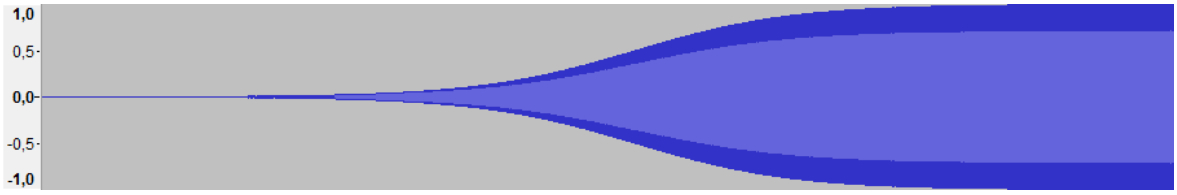


Onda risultante dall'applicazione dell'algoritmo base nel range 500.

Onda risultante dall'applicazione dell'algoritmo con distanza euclidea nel range 500.

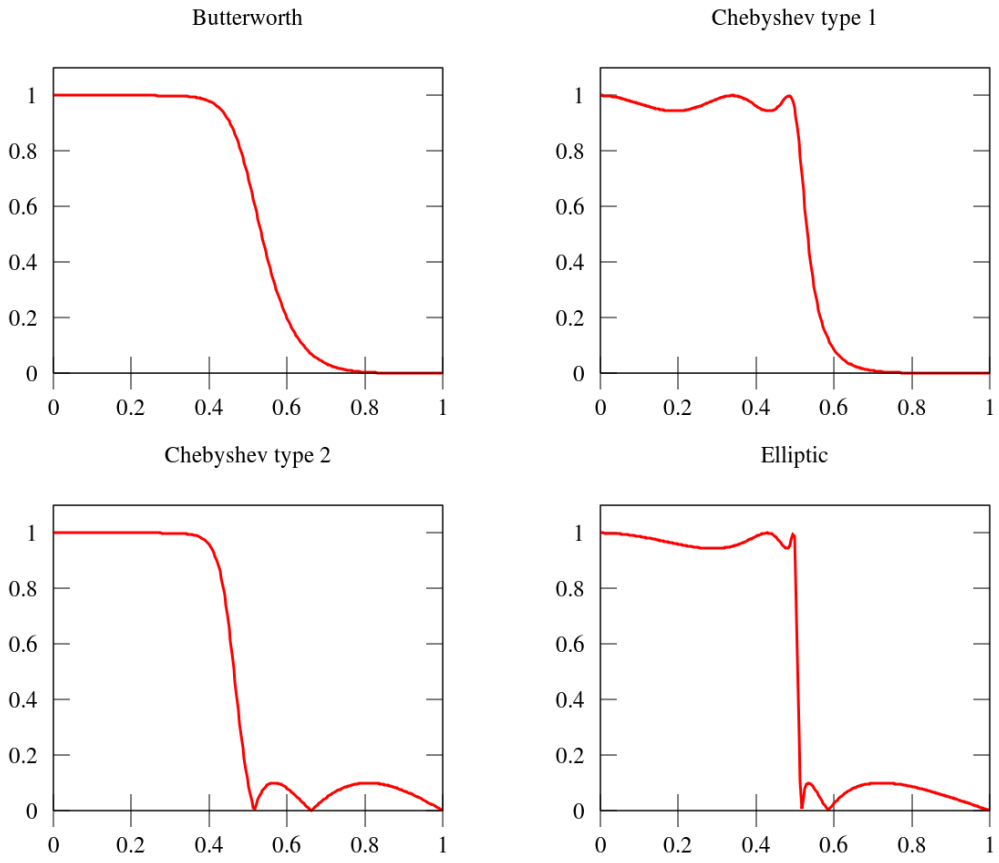


Onda risultante dall'applicazione dell'algoritmo
con distanza inverse exponential α=0 nel range 500.



Onda risultante dall'applicazione dell'algoritmo
con distanza inverse exponential α=0.1 nel range 500.

In dettaglio, andando ad analizzare le curve dei file di output, si è riscontrato che, con tutte e tre le funzioni distanza, si è ottenuta un'onda modellata come un filtro di Chebyshev di tipo 1, cioè con una banda di transizione più stretta a discapito della linearità generale. Tuttavia, incrementando il parametro dell'inverse exponential, l'onda assumeva una forma più simile a quella dell'applicazione di un filtro Butterworth, quindi con una banda di transizione più larga e la banda passante lineare. Più il valore α era vicino a 1 più il taglio si spostava a destra, lasciando passare esclusivamente le alte frequenze.



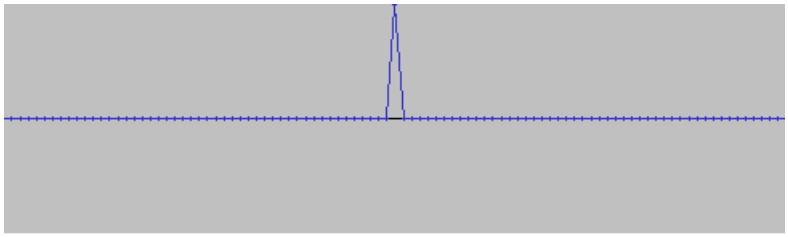
Principali tipologie di filtro usato in ambito musicale.

Tuttavia, analizzando lo spettro FFT dei segnali in uscita, si è notato che la distorsione introdotta era stata minima quindi, evidentemente, l'algoritmo non lavorava tanto sulla gamma della dinamica quanto piuttosto sulle frequenze fungendo da filtro passa alto, passa basso o passa banda.

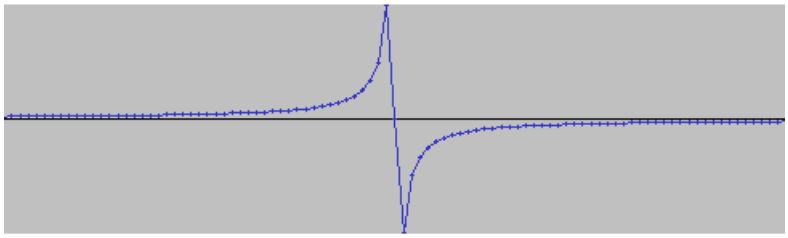
3.8 Test impulso unitario

Una seconda verifica è stata effettuata con un impulso unitario (o Delta di Dirac) ovvero un segnale che è uguale a zero ovunque tranne in un singolo punto in cui è uguale a 1. La sua principale caratteristica è che esso contiene l'intero spettro delle frequenze, quindi è ideale per analizzare il comportamento di un sistema dinamico lineare come l'algoritmo che si è scelto di utilizzare. L'output di tale sistema, quando è soggetto ad un input a Delta di Dirac, viene chiamato risposta all'impulso (IR).

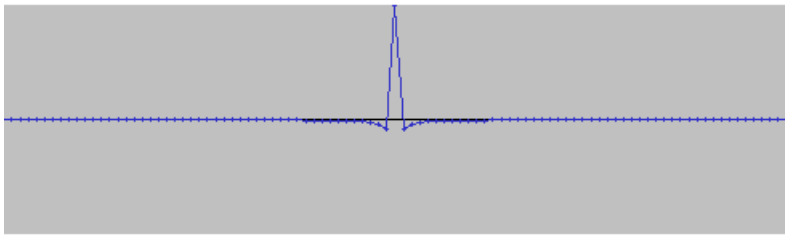
Delta di Dirac.



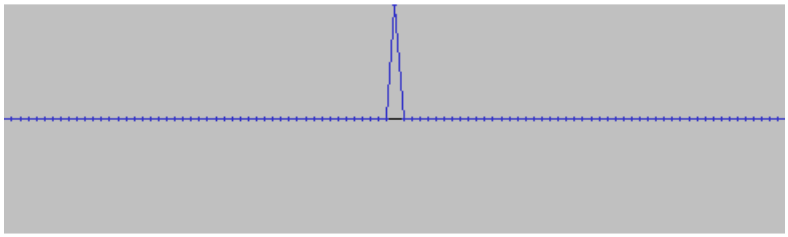
IR con distanza lineare e range 500.



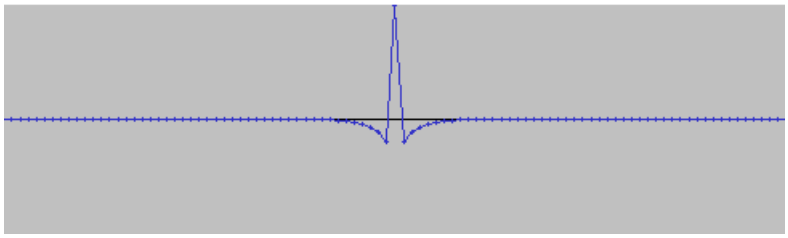
IR con distanza euclidea e range 500.



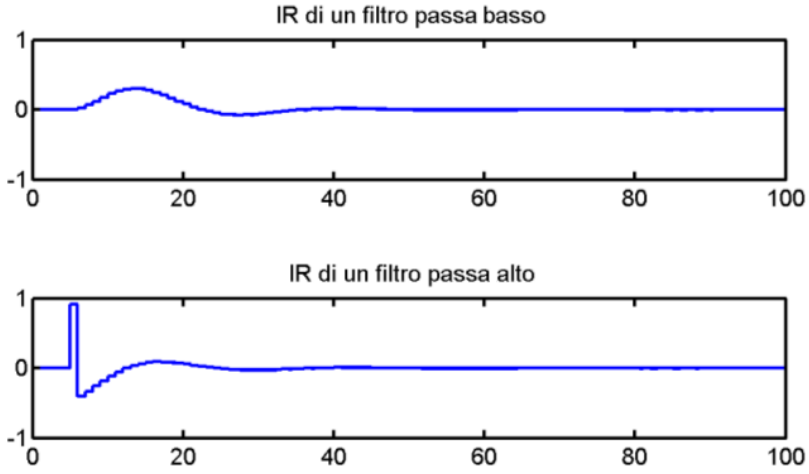
IR con distanza inverse exponential α=0 e range 500.



IR con distanza inverse exponential α=0.5 e range 500.



In particolare, l'applicazione dell'algoritmo con distanza euclidea e con distanza esponenziale inversa a parametro maggiore di 0, portava ad un risultato del tutto similare a un filtro passa alto.



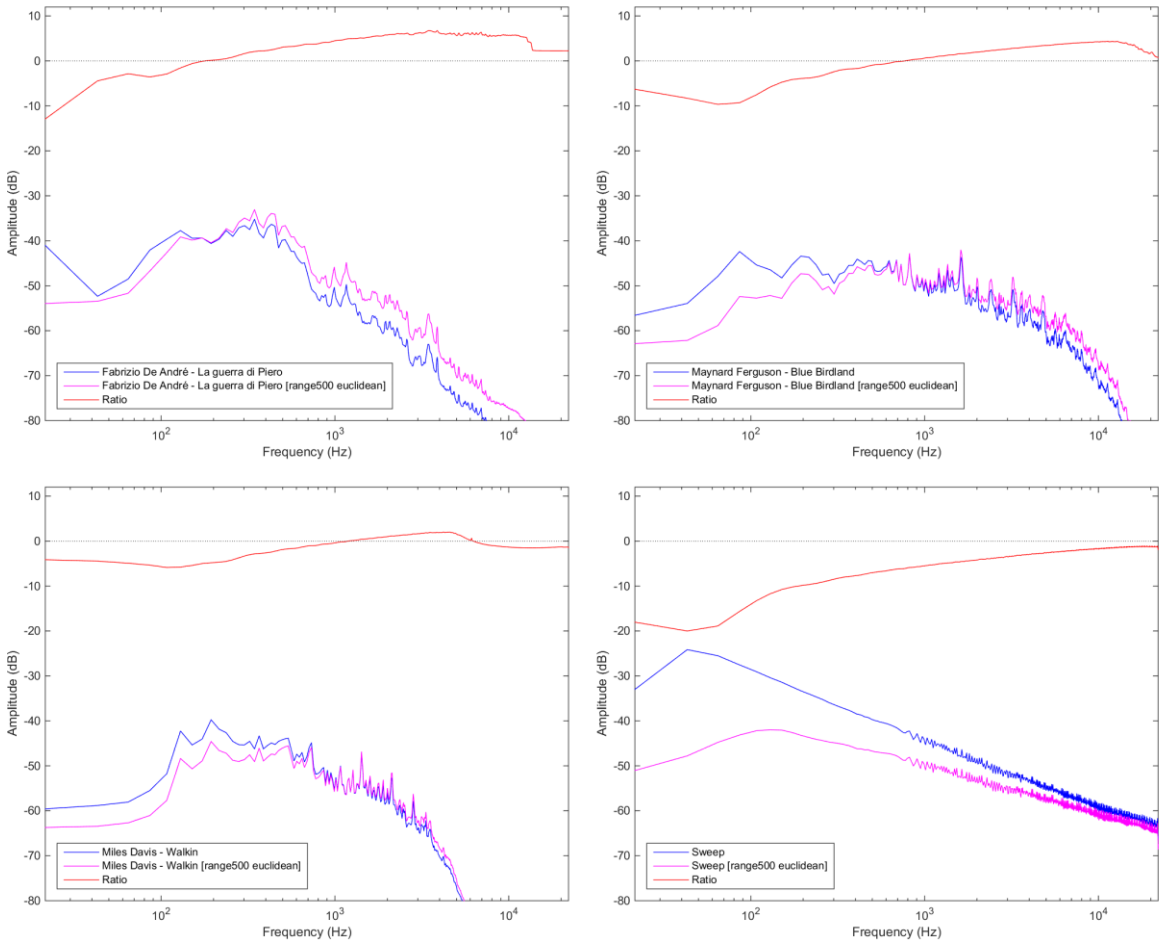
Risultato dell'applicazione di un semplice passa basso e passa alto alla Delta di Dirac.

3.9 Test file musicale

Sono stati inoltre eseguiti dei test su file audio musicali. Il primo input utilizzato è stato estratto dal videoclip della canzone "La guerra di Piero" di Fabrizio De André.
Tale clip è stata digitalizzata da una vecchia VHS, il cui nastro si è inevitabilmente deteriorato nel tempo.

Il file wave utilizzato quindi presentava un suono uniforme, senza una netta distinzione tra voce e accompagnamento strumentale.
Applicando le diverse varianti dell'algoritmo i risultati migliori sono stati ottenuti con un range di 5000 campioni. In particolare si è ottenuta un'esaltazione e una giusta calibrazione di tutte le frequenze con la distanza inverse exponential a parametro 0, oppure applicando prima l'algoritmo lineare e poi quello con distanza euclidea al risultato ottenuto.
Andando ad analizzare graficamente la variazione delle diverse frequenze, si è constatato che l'applicazione dell'algoritmo sull'onda di diversi input musicali produceva un output modellato in modo similare ai test precedenti eseguiti sullo sweep e sull'impulso unitario.

Ogni funzione distanza agisce quindi come descritto nelle pagine precedenti, tuttavia si possono riscontrare leggere variazioni nel taglio delle frequenze dovute alla capacità dell'algoritmo di adattarsi al contenuto dell'input utilizzato.



Confronto risposta in frequenza con distanza euclidea a range 500 di tre tracce musicali e dello sweep (in basso a destra). In rosso viene indicato il rapporto output/input che partendo dalle basse frequenze a sinistra arriva fino alle alte a destra.

Ad ogni modo sulla forma d'onda l'algoritmo ha assunto solamente la funzione di un equalizzatore e non sono state notate variazioni dal punto di vista della dinamica.

Capitolo 4

L'algoritmo ACE applicato allo spettro FFT

Compresa la necessità di lavorare sullo spettro FFT dei file audio, si è scelto di utilizzare l'ambiente scritto in C denominato MatLab. Tale ambiente comprende l'omonimo linguaggio di programmazione creato dalla MathWorks, il quale fornisce numerosi metodi di calcolo numerico per analizzare dati, sviluppare algoritmi e creare modelli.

4.1 La scelta di MatLab

Il linguaggio MatLab include funzioni matematiche che supportano comuni operazioni scientifiche. Le principali funzioni matematiche utilizzano librerie ottimizzate per il processore, per consentire una rapida esecuzione di calcoli vettoriali e matriciali. Tra i diversi metodi disponibili sono inclusi quelli per l'analisi di Fourier.

Inoltre sono forniti strumenti per acquisire, analizzare e visualizzare i dati, consentendo una comprensione approfondita dei dati in una frazione del tempo che servirebbe usando linguaggi di programmazione tradizionali.

L'elemento base di MatLab è la matrice, ovvero un array a due dimensioni composto da m righe e n colonne. Ad esempio per creare una matrice A così definita:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

sarà necessario digitare: $A = [1\ 2\ 3; 4\ 5\ 6; 7\ 8\ 9]$

Una volta indicata la matrice, questa sarà automaticamente memorizzata nel Workspace e vi potremo fare riferimento semplicemente digitando A. Inoltre sarà possibile accedere ad un particolare elemento di essa specificando la sua locazione. Scrivendo quindi: $A(2,1)$ otterremo il numero 4, cioè l'elemento nella seconda riga della prima colonna.

Per determinare le dimensioni della matrice potremo infine usare il comando $size(A)$.

La tabella seguente raccoglie le funzioni per la costruzione di matrici esplicate in [9].

$eye(n)$	matrice identità di dimensione $n \times n$
$zeros(m,n)$	matrice di 0 di dimensione $m \times n$
$ones(m,n)$	matrice di 1 di dimensione $m \times n$
$diag(v)$	matrice diagonale con diagonale $\{v_k\}_{k=1\ldots n}$
$rand(m,n)$	matrice di numeri casuali di dimensione $m \times n$

Si vuole infine illustrare l'operatore : , uno dei più importanti in MatLab. Esso appare in diversi contesti:

- Per creare una riga di elementi equidistanti:

$\gg v1 = 1:10$

$$v1 = 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10$$

$\gg v2 = 100:-7:50$

$$v2 = 100\ 93\ 86\ 79\ 65\ 58\ 51$$

- Per estrarre la riga i-esima di una matrice A si scrive: $A(i,:)$
- Per estrarre la colonna j-esima di una matrice A si scrive: $A(:,j)$
- Per eliminare la riga i-esima di una matrice A si scrive: $A(i,:) = [\,]$
- Per eliminare la colonna i-esima di una matrice A si scrive: $A(:,i) = [\,]$
- Per scrivere una matrice A di dimensioni $m \times n$ come un vettore colonna di mn elementi si scrive: $A(:)$

4.2 Conversione di un segnale monofonico tramite FFT

Tramite un numero ridotto di righe di codice nel linguaggio MatLab è possibile estrapolare da un segnale audio a singolo canale il relativo spettro FFT (trasformata veloce di Fourier):

```
1.    input = wavread ('Test.wav');
2.    framesize = 1024;
3.    bin_utili = floor(framesize/2) + 1;
4.    frames = buffer(input, framesize);
5.    window = hanning(framesize);
6.    frames = bsxfun(@times,frames,window);
7.    complex = fft(frames)/framesize;
8.    magnitudine = abs(complex(1:bin_utili,:));
9.    image (magnitudine,'CDataMapping','scaled');
10.   axis xy
```

Segue la spiegazione dei comandi precedenti:

1. Carico un file nel vettore 'input'
2. Decido la risoluzione temporale (in campioni)
3. Calcolo il numero di quanti con cui verrà campionato lo spettro
4. Divido il segnale di input in segmenti
5. Creo una maschera per portare a zero la testa e la coda dei segmenti
6. Moltiplico ogni segmento con la maschera
7. Calcolo la FFT e scalo il risultato (un side-effect dell'algoritmo)
8. Metto in 'magnitudine' il livello di ogni sinusoidale del segnale
9. Visualizzo come colore la potenza delle sinusoidi nel tempo
10. Oriento gli assi in modo da avere x=frequenza e y=tempo

Poiché lo spettro FFT di un file audio è equiparabile ad una immagine, potremo applicare l'algoritmo ACE nel modo in cui è stato concepito, ovvero il confronto dei punti di una matrice per la regolazione cromatica delle immagini, che nel caso dell'audio corrisponderà ad una modifica della gamma dinamica.

4.3 Confronto monodimensionale e bidimensionale

In questo caso, al posto di un range fisso per il confronto, si è scelto di utilizzare una percentuale relativa alla finestra di risoluzione temporale. Se ad esempio si sceglie 10 come range, il confronto verrà eseguito, per ogni singolo punto a della finestra, con tutti i punti b compresi entro una matrice/finestra rettangolare ridotta pari al 10% dell'originale, in cui a rappresenta il punto centrale. È stata inoltre introdotta la possibilità di scegliere tra altre due tipologie di confronti, in questo caso monodimensionali. Una nel dominio della frequenza, confrontando solo i valori aventi la stessa coordinata y, compresi nel range del punto a in questione. Un'altra nel dominio del tempo, variando le y per i punti di confronto e mantenendo fisse le x.
Tutto questo è stato reso possibile grazie a una variabile che a seconda del valore rimandava a diverse porzioni di codice, così da rendere selezionabile la tipologia di confronto.

```
% Variabili da modificare
framesize = 1024;
range = 10;
q = 2;

if range<1 || range>100
    error('Range must be between 1 and 100.');
end

% L'ammontare della percentuale per le x
rx = floor((range/100)*size(magnitudine,1));

% Faccio in modo che il numero di punti x della finestra ridotta sia pari
if mod(rx,2)~=0
    rx = rx+1;
end

% L'ammontare della percentuale per le y
ry = floor((range/100)*size(magnitudine,2));

% Faccio in modo che il numero di punti y della finestra ridotta sia pari
if mod(ry,2)~=0
    ry = ry+1;
end

switch q
      % Confronto sulla stessa riga
      case 0
      ...
      % Confronto sulla stessa colonna
      case 1
```

```
      ...
      % Confronto sulla matrice intera
      case 2
      ...
      otherwise
      error('Unexpected window type.');
end
```

4.4 La funzione distanza nel caso monodimensionale

L'algoritmo ACE, nella tipologia di confronto con valori x o valori y fissi, è stato applicato utilizzando le tre diverse varianti della distanza a denominatore già utilizzate nell'applicazione alla forma d'onda.

Manhattan/Linear	Euclidean/Absolute	Inverse exponential
d	$\|d\|$	$\frac{1}{e^{-\alpha E d}}$

```
A = magnitudine(ax,ay);
B = magnitudine(bx,by);
switch d

      % Distanza lineare
      case 0
      out(ax,ay) = out(ax,ay) + ((A-B) / (ax-bx));

      % Distanza assoluta
      case 1
      out(ax,ay) = out(ax,ay) + ((A-B) / abs(ax-bx));

      % Distanza esponenziale inversa
      case 2
      out(ax,ay) = out(ax,ay) + ((A-B) / (1/exp(-k*abs(ax-bx))));

      otherwise
      error('Unexpected distance function.');

end
```

4.5 La funzione distanza nel caso bidimensionale

Nel caso del confronto entro una matrice le funzioni sono state riscritte nella loro versione bidimensionale. Tuttavia la versione Manhattan $dx + dy$ portava ad un azzeramento dei campioni, per questo si è scelto di sostituirla con una funzione che selezionava il massimo tra la distanza delle x e la distanza delle y dei due punti dello spettro a confronto.

Maximum	Euclidean	Inverse exponential
$Max(dx, dy)$	$\sqrt{dx^2 + dy^2}$	$\frac{1}{e^{-\alpha Ed}}$

In questo caso Ed nell'inverse exponential non sarà più il semplice valore assoluto della distanza, ma rappresenterà l'intera funzione euclidea $\sqrt{dx^2 + dy^2}$.

```
A = magnitudine(ax,ay);
B = magnitudine(bx,by);
switch d

      % Distanza massima
      case 0
      out(ax,ay) = out(ax,ay) + ((A-B) / max(ax-bx,ay-by));

      % Distanza euclidea
      case 1
      out(ax,ay) = out(ax,ay) + ((A-B) / sqrt((ax-bx)^2+(ay-by)^2));

      % Distanza esponenziale inversa
      case 2
      out(ax,ay) = out(ax,ay) + ((A-B) /
                  (1/exp(-k*sqrt((ax-bx)^2+(ay-by)^2))));
      otherwise
      error('Unexpected distance function.');

end
```

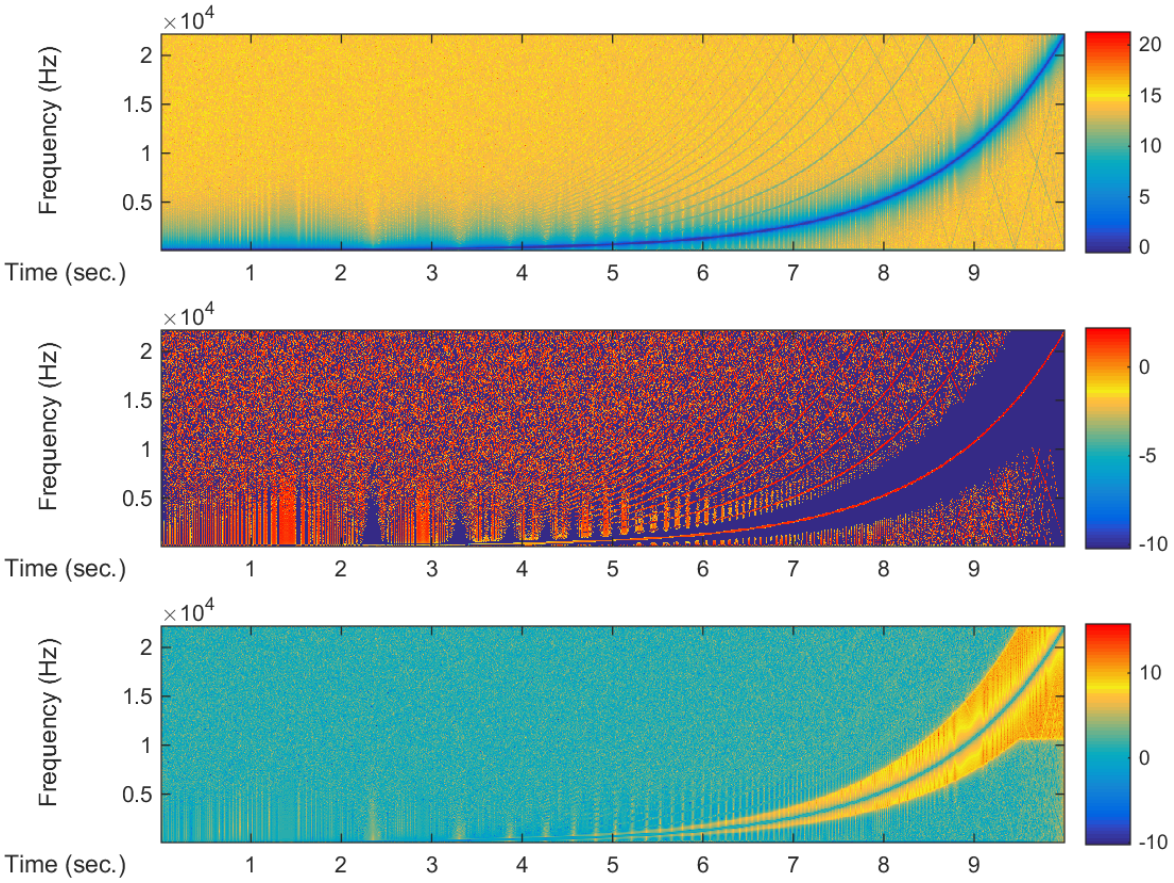
Le variabili modificabili per i test sono diventate quindi le seguenti:

```
% Variabili da modificare
framesize = 1024;
range = 10;             % Intero [1,100]
q = 2;                  % 0 = riga       1 = colonna       2 = matrice
d = 0;                  % Intero [0,2]
k = 0;                  % Double [0,1]   se d = 2
```

4.6 Gestione dei valori negativi sull'output

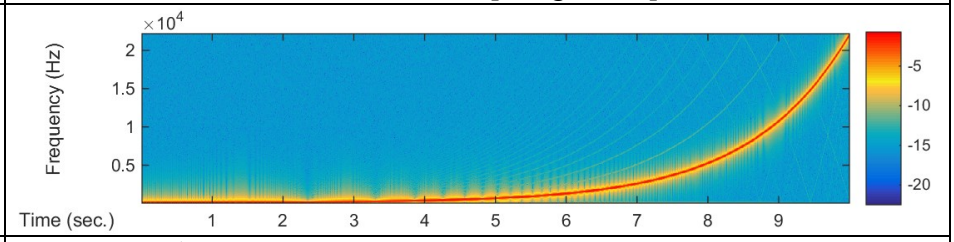
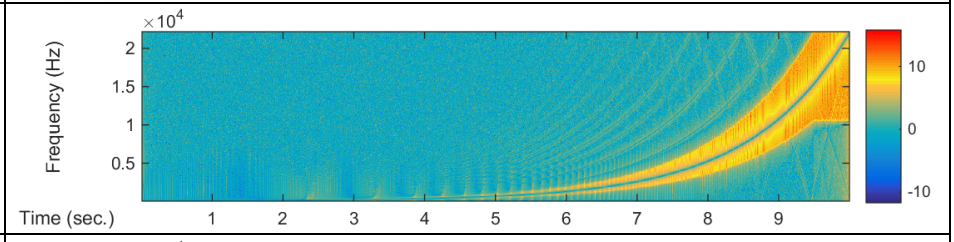
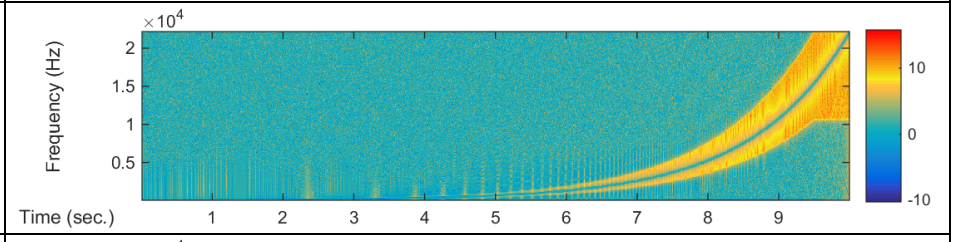
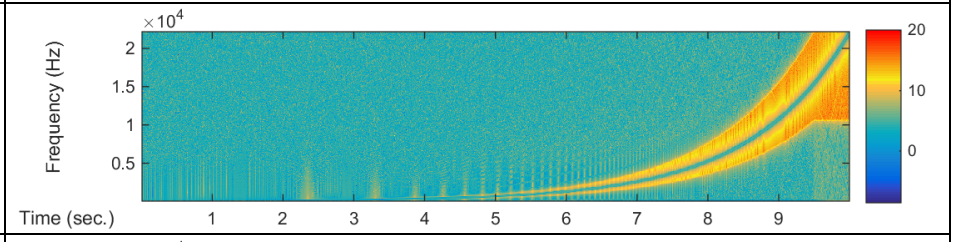
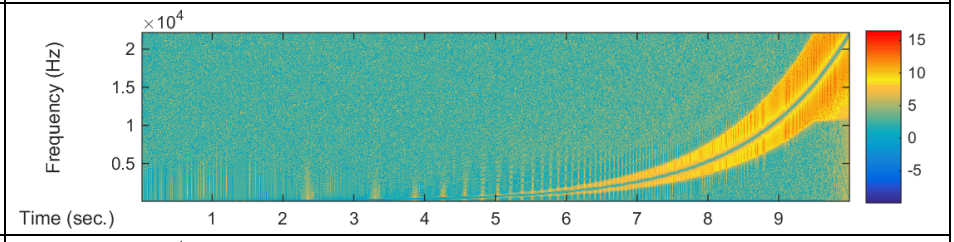
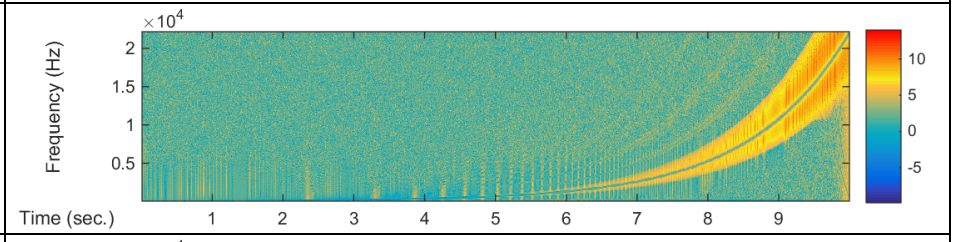
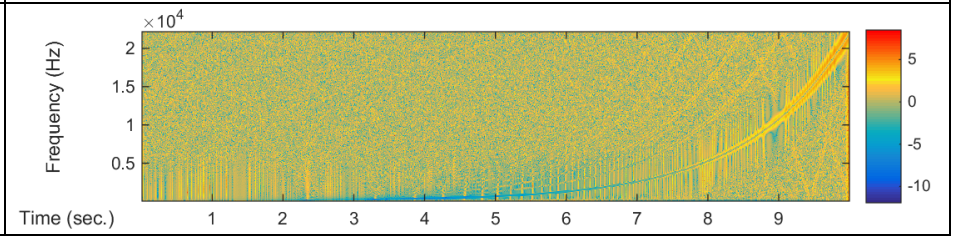
Il problema principale nell'applicazione dell'algoritmo allo spettro è stato quello di comprendere come gestire i valori che in seguito all'applicazione dell'algoritmo diventavano minori di zero. Dopo alcuni tentativi che portavano ad un audio quasi totalmente distorto, si è scelto di considerare tali numeri come positivi applicando il valore assoluto.

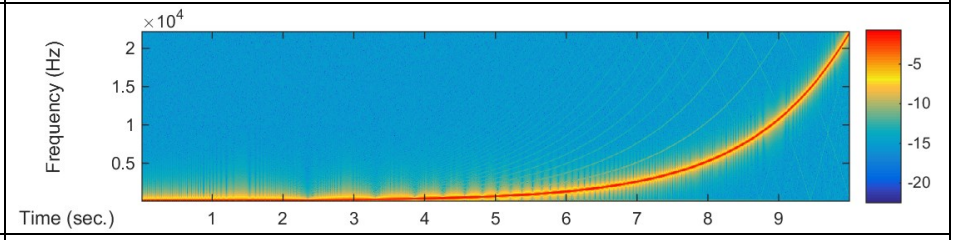
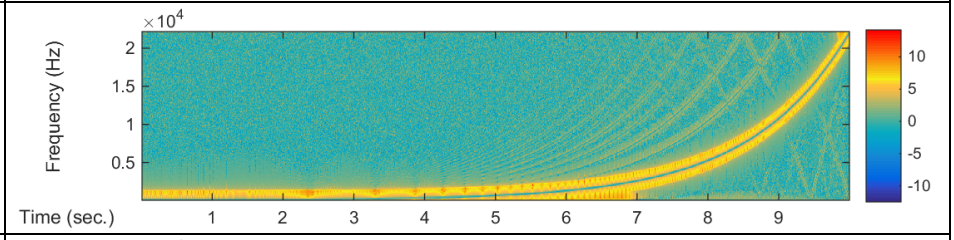
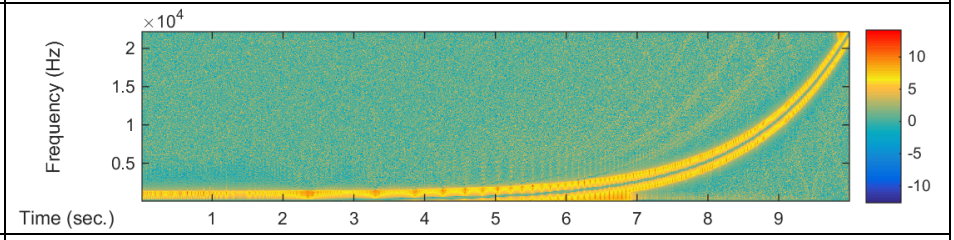
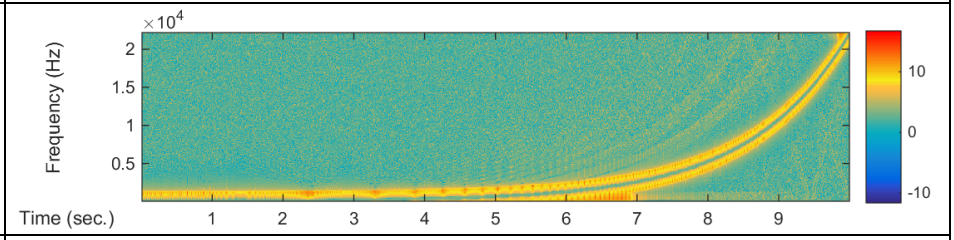
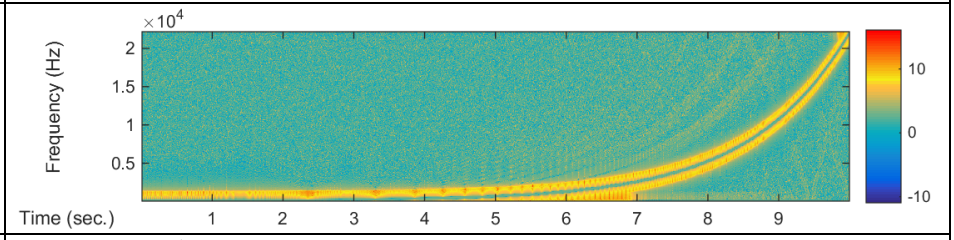
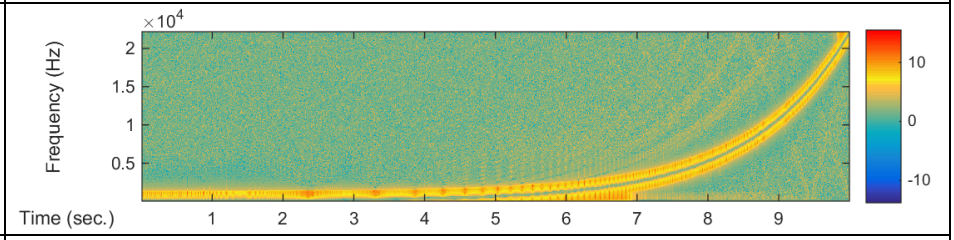
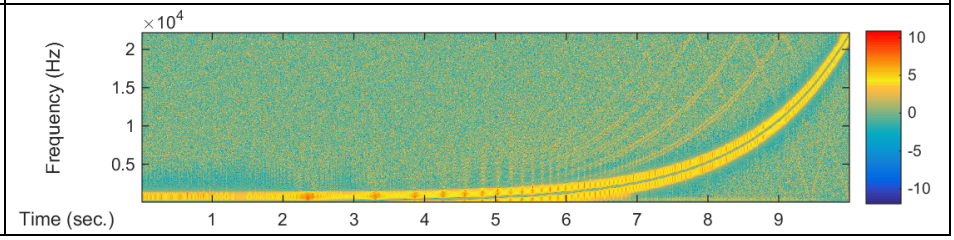
```
% Offset (aggiunge rumore)
outMin = min(min(out));
out = out - outMin;
gain = (max(max(in))/max(max(out)));
out = out .* gain;

% Troncamento (considera numeri negativi come silenzio)
out(out<0) = 0;

% Foldback (considera numeri negativi come positivi)
out = abs(out);
```

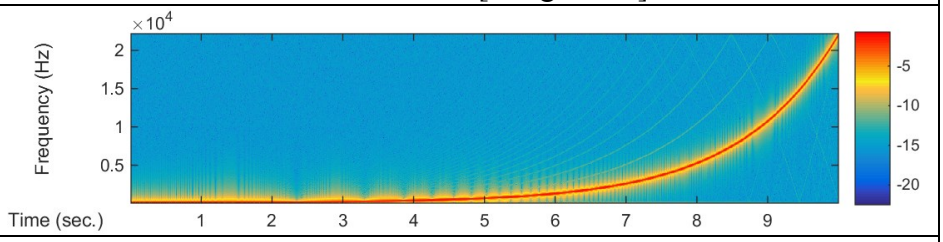
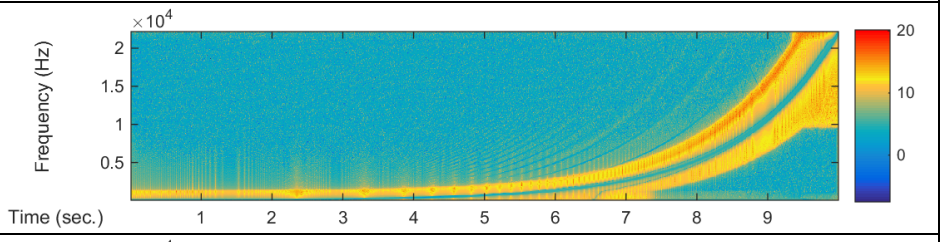
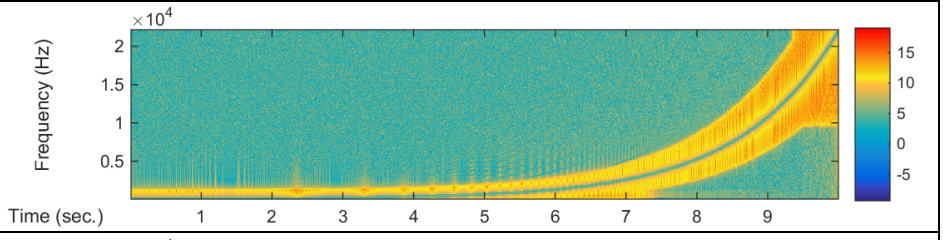
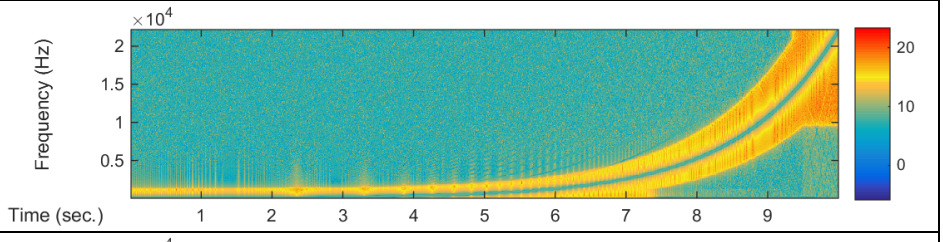
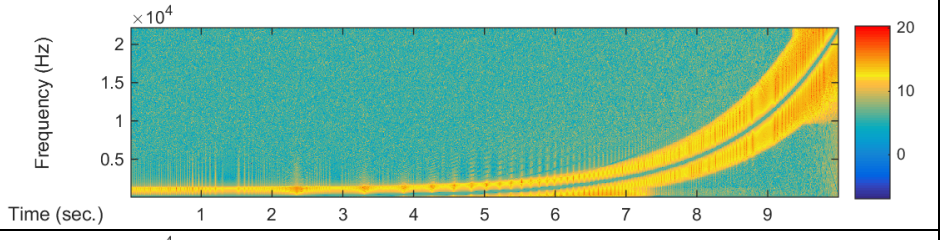
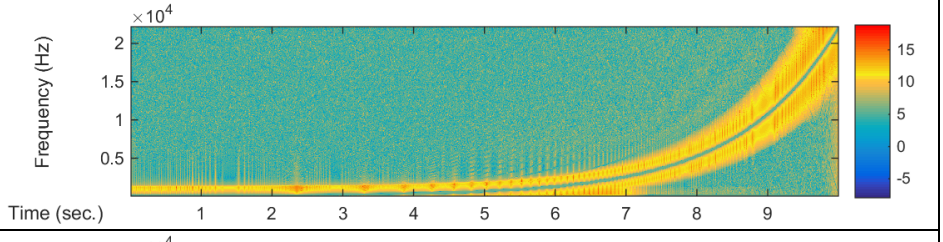
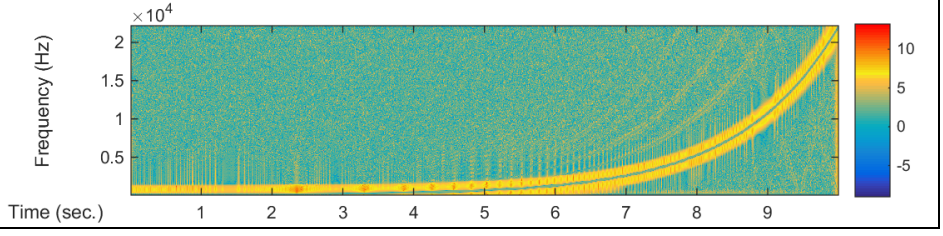


Spettro del rapporto output/input di un segnale sweep utilizzando una finestra di 512 e un range del 10%. Dall'alto verso il basso: offset, troncamento e foldback.

4.7 Rapporto output/input sweep

		Framesize 512 [Range 10%]
Time	ORIGINAL INPUT	
	OUT/IN RATIO Linear	
	OUT/IN RATIO Absolute	
	OUT/IN RATIO Inverse exponential α=0	
	OUT/IN RATIO Inverse exponential α=0.05	
	OUT/IN RATIO Inverse exponential α=0.1	
	OUT/IN RATIO Inverse exponential α=0.5	

		Framesize 512 [Range 10%]
Frequency	ORIGINAL INPUT	
	OUT/IN RATIO Linear	
	OUT/IN RATIO Absolute	
	OUT/IN RATIO Inverse exponential α=0	
	OUT/IN RATIO Inverse exponential α=0.05	
	OUT/IN RATIO Inverse exponential α=0.1	
	OUT/IN RATIO Inverse exponential α=0.5	

		Framesize 512 [Range 10%]
Time-Frequency	ORIGINAL INPUT	
	OUT/IN RATIO Maximum	
	OUT/IN RATIO Euclidean	
	OUT/IN RATIO Inverse exponential α=0	
	OUT/IN RATIO Inverse exponential α=0.05	
	OUT/IN RATIO Inverse exponential α=0.1	
	OUT/IN RATIO Inverse exponential α=0.5	

I test sono stati eseguiti con un range pari al 5, 10 e 20% della dimensione della finestra. La finestra (framesize) è stata settata a 128, 512 e 1024. Il primo risultato evidente è stato che, aumentando la dimensione della finestra, le modifiche allo spettro si espandevano. Si è quindi scelto di mantenere una finestra con dimensione media di 512.

La seguente tabella riassume gli effetti notati in seguito all'applicazione sullo sweep e sui brani musicali.

Tempo	Espansione in larghezza picchi inviluppo, ma frammentata. Crea effetto eco.
Frequenza	Incremento dell'RMS in altezza.
Matrice (Tempo-Frequenza)	Espansione sia dell'RMS che dell'inviluppo in quantità maggiore rispetto all'applicazione monodimensionale.
Incremento framesize	Espansione dei cambiamenti nello spettro.
Incremento range	Ulteriore espansione dei cambiamenti nello spettro.

Dal punto di vista percettivo l'applicazione di ACE allo spettro si è rivelata inefficace, poiché incrementava il rumore di fondo senza portare miglioramenti significativi alla dinamica o alla gamma delle frequenze. Inoltre, nella maggior parte dei casi, introduceva delle distorsioni e dei click dovuti alla presenza dei valori negativi che, come detto precedentemente, si è scelto di trattare come positivi.

Capitolo 5

Inviluppo e ampiezza RMS

Un'ulteriore serie di verifiche è stata effettuata sull'inviluppo dei file audio, ovvero il profilo che caratterizza l'andamento del suono. Per fare ciò è stata utilizzata una funzione che estrae l'inviluppo dall'input in modo più o meno preciso a seconda della finestra "windowsize" specificata. Più il valore di quest'ultima sarà alto più l'inviluppo risultante assumerà un aspetto smussato.

```
function output = envel(input,windowsize,smth)
if nargin < 3; smth = 1; end;
frames = buffer(input,windowsize);
envelope = max(abs(frames),[],1);
envelope = repmat(envelope,windowsize,1);
envelope = envelope(:);
if smth == 1
    output = smoothfit(envelope,windowsize,@max);
else
    output = envelope;
end
l = length(input);
output = output(1:l);
end
```

In questo caso l'algoritmo è stato applicato, come per la forma d'onda, unicamente nella versione ad una dimensione trattandosi di singoli punti per ogni istante di tempo.

5.1 Normalizzazione e regolazione dei silenzi

L'inviluppo risultante dall'applicazione di ACE si è rivelato essere d'aspetto più simile a quello di una forma d'onda. Inoltre presentava alcuni valori che andavano sotto lo zero, per cui si è proceduto ad una normalizzazione per rendere il nuovo inviluppo compreso tra 0 e 1. Per poter utilizzare quest'ultimo in modo efficace si è quindi deciso di applicare al file audio il rapporto tra l'inviluppo modificato e l'inviluppo originale.

Il file in output presentava un incremento delle sonorità a volume più basso. Tuttavia, più il suono si presentava basso, più tale suono veniva incrementato. In questo modo i silenzi del file originale si presentavano come un forte fruscio. Di conseguenza si è proceduto a introdurre una regolazione dei silenzi impostando un valore `rate` oltre il quale si decideva di considerare l'incremento del volume come silenzio. Tutti i valori oltre tale soglia venivano riflessi verticalmente, ma con una normalizzazione, in modo che non scendessero sotto lo

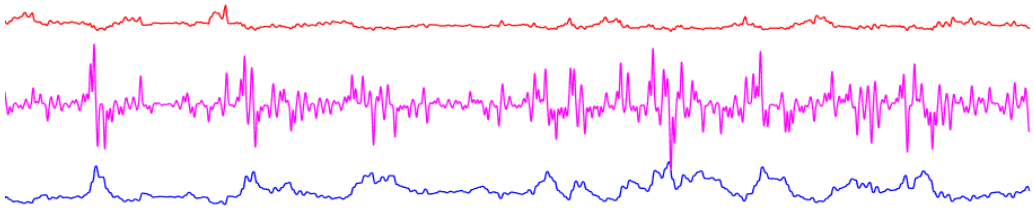
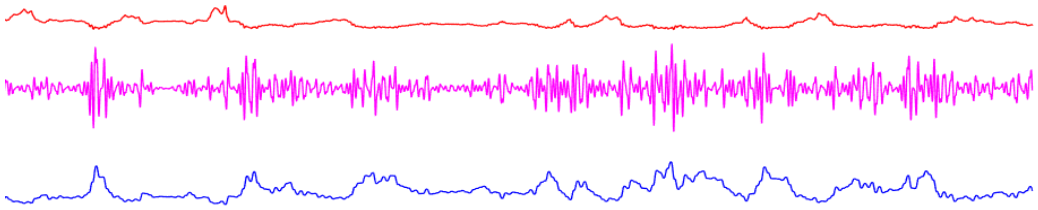
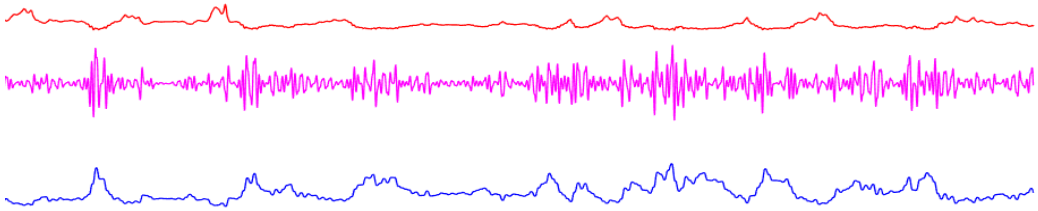
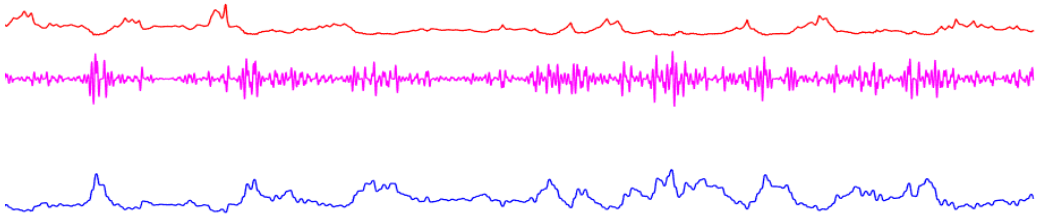
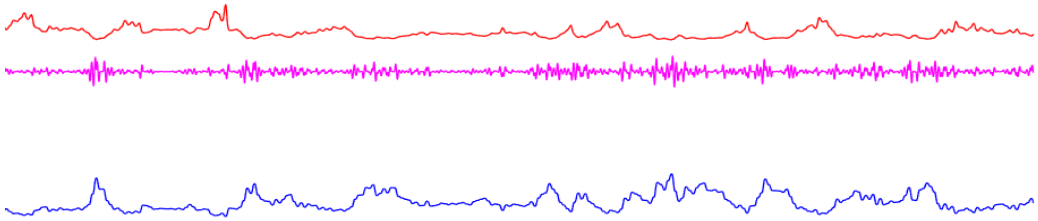
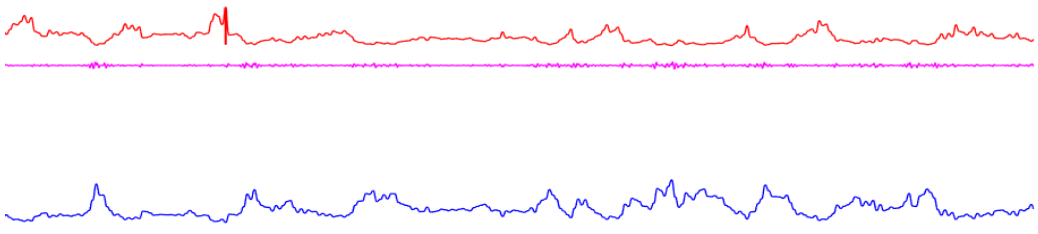
zero. In questo modo, ad esempio, nei file musicali che terminano sfumando gradualmente verso il silenzio, veniva mantenuta tale peculiarità.

```
% Normalizzazione dell'inviluppo in output in modo che i valori siano
% compresi tra 0 e 1
out = out - min(out);
out = out./max(out);
gainChange = out./in;

% Regolazione dei silenzi
rate = 10;
maxGain = max(gainChange);
gainRate = maxGain/rate;
for i=1 : length(gainChange)
    if gainChange(i)>rate
        gainChange(i) = rate-(gainChange(i)/gainRate);
    end
end

% Applicazione delle modifiche dell'inviluppo al segnale in ingresso
output = gainChange.*input;
```

5.2 Test ACE su inviluppo

Poiché la curva dell'inviluppo di segnali come lo sweep o l'impulso unitario è poco significativa, l'applicazione di ACE su questi risultava comprensibilmente inefficace. Si sono svolti quindi una serie di test su file musicali ed in particolar modo sul brano Blue Birdland del trombettista jazz Maynard Ferguson.

La tabella che segue riporta l'inviluppo in ingresso, in uscita e il rapporto tra essi di un estratto di 8 secondi del brano.

	Framesize 512 [Range 500]
LEGEND	Maynard Ferguson – Blue Birdland [8 sec.] OUTPUT / INPUT RATIO OUTPUT INPUT
Linear	
Absolute	
Inverse exponential α=0	
Inverse exponential α=0.05	
Inverse exponential α=0.1	
Inverse exponential α=0.5	

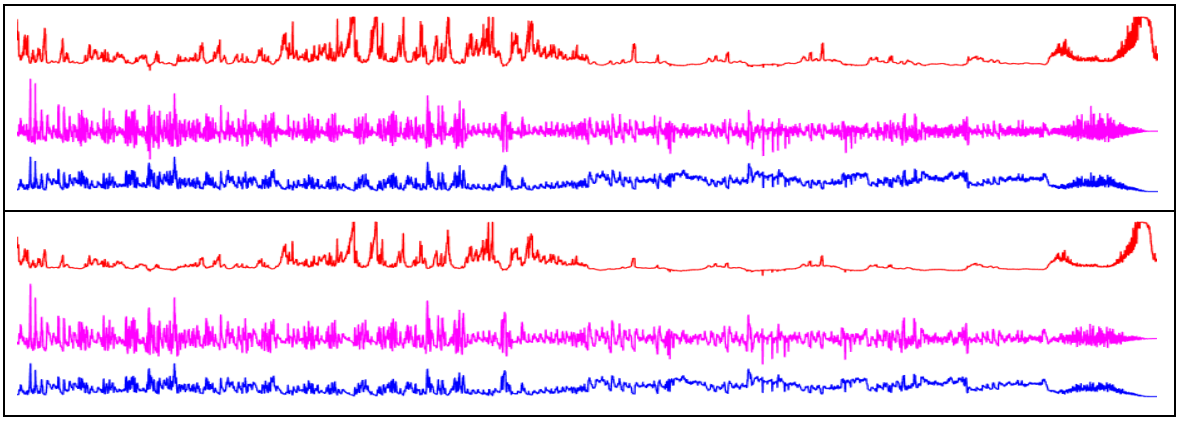
Nel caso dell'inviluppo il problema principale è risultato essere il tempo di elaborazione eccessivamente lungo, in parte dovuto all'ambiente di calcolo MatLab e in parte alle scelte fatte nello scrivere il codice dello script. Ad ogni modo i file in output acusticamente risultavano non troppo diversi da quelli che produrrebbe un semplice compressore impostando il valore di threshold in modo adeguato. Si è quindi scelto di procedere ad ulteriori verifiche concentrandosi sull'ampiezza RMS.

5.3 Root Mean Square

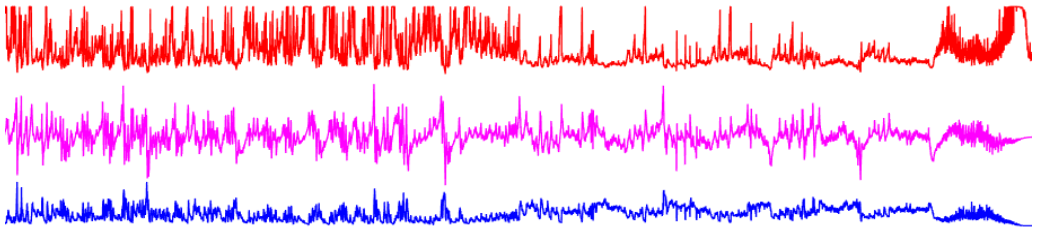
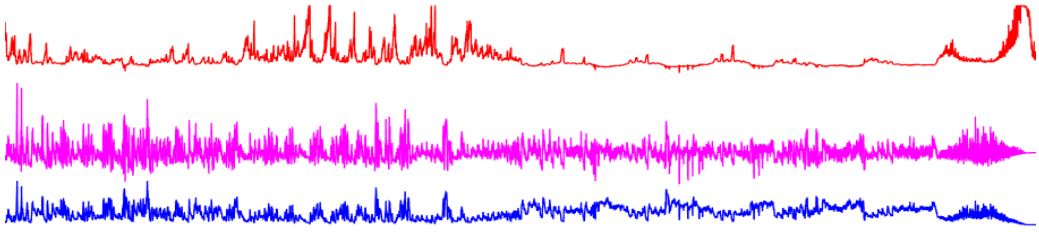
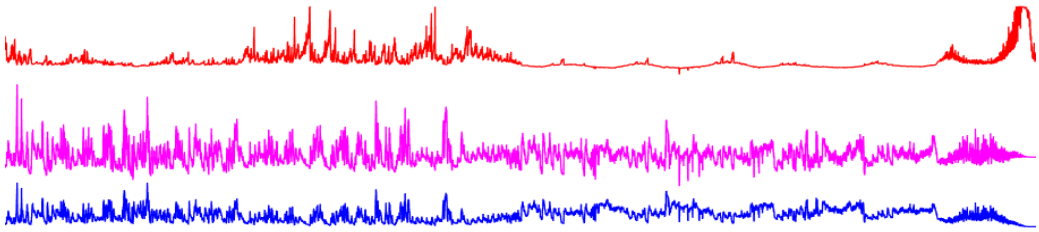
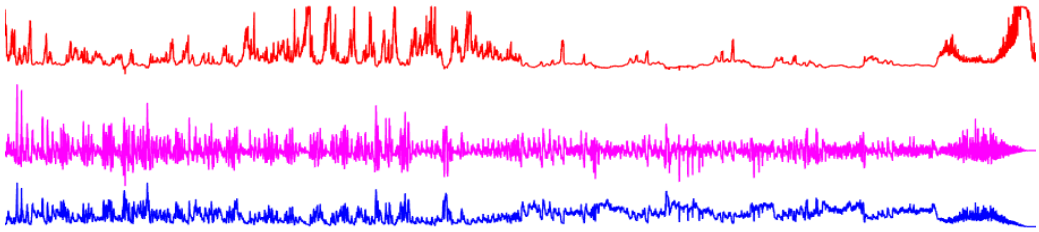
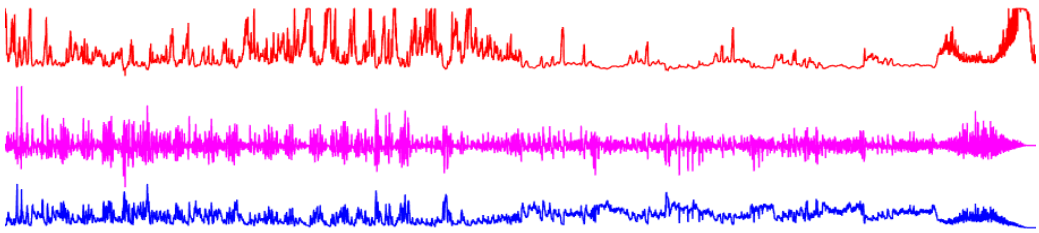
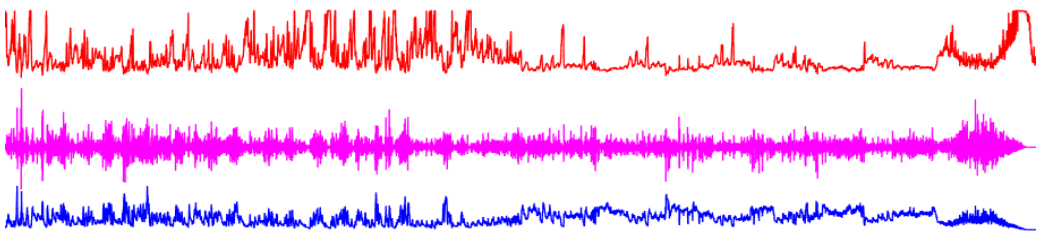
In ultima analisi si è andato a testare se ACE potesse risultare efficace nell'applicazione al Root Mean Square dell'audio. Dopo aver letto quindi il file, e memorizzata la parte audio in una variabile `data`, si è proceduto a dividere l'input in frames e a calcolare l'RMS di ognuno di essi prima di applicarvi l'algoritmo.

```
% Suddivisione dell'input in frames e calcolo dell'RMS di ogni frame
framesize = 512;
overallInputRms = mean(data.^2).^0.5;
inputFrames = buffer(data, framesize, framesize/2);
in = mean(inputFrames.^2,1).^0.5;
```

Il numero di campioni sottoposti all'algoritmo veniva quindi ampiamente ridotto e controllato con un parametro framesize, consentendo di processare un file musicale della durata di 5 minuti in meno di 2 secondi.
Si è quindi eseguito nuovamente il test su Blue Birdland, questa volta in versione integrale. Data la velocità dello script in questo caso si è eseguito sia un test con ACE limitato a un range di 500 campioni sia confrontando ogni singolo campione con tutti gli altri. La differenza del risultato di queste due versioni tuttavia risultava veramente sottile con l'RMS. Si sono invece riscontrate differenze abbastanza notevoli modificando il framesize. Impostandolo a 512 si otteneva un output con picchi maggiori rispetto a un settaggio a 1024.



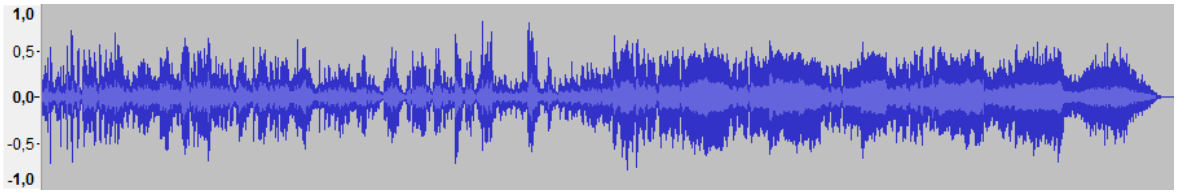
Ratio, output e input di Blue Birdland utilizzando un framesize prima di 512 e poi di 1024, applicando ACE con inverse exponential α=0.05 e confronto completo (senza range).

	Framesize 512 [Range 500]
LEGEND	Maynard Ferguson – Blue Birdland [00:01:43] OUTPUT / INPUT RATIO OUTPUT INPUT
Linear	
Absolute	
Inverse exponential α=0	
Inverse exponential α=0.05	
Inverse exponential α=0.1	
Inverse exponential α=0.5	

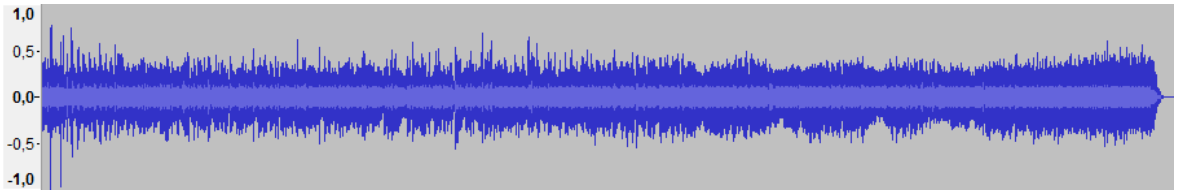
I risultati migliori sono stati ottenuti utilizzando come distanza dell'algoritmo la funzione esponenziale inversa con valore α molto vicino allo 0. In particolare con una calibrazione a 0.05 si è ottenuta una corretta regolazione delle diverse sonorità del brano, andando ad incrementare i suoni deboli senza tuttavia comprimere eccessivamente quelli forti, che in rari casi sono stati addirittura esaltati. Nei grafici che seguono è infatti possibile notare un incremento di ampiezza in presenza dei clap che scandiscono l'inizio brano.

Per concludere si riporta, a fine di confronto, l'output ottenuto con un compressore semplice che regola i valori dell'RMS intorno ad un'ampiezza indicata.

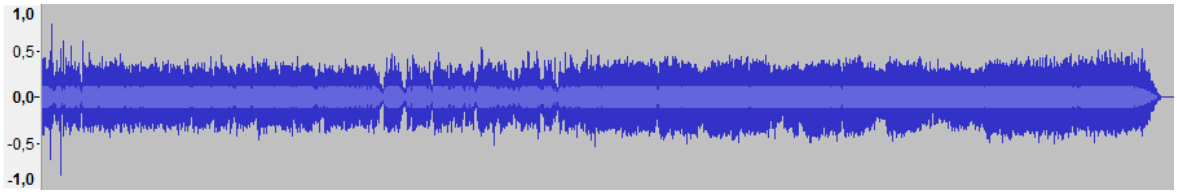
```
% Compressore semplice
amplitude = 0.05;
out = in;
out(in>amplitude) = amplitude;
gainChange = out./in;
```



Forma d'onda originale del brano Blue Birdland.



Forma d'onda risultante dall'applicazione dell'algoritmo ACE con distanza inverse exponential α=0.05 nel range 500.



Forma d'onda del brano in seguito all'applicazione di un compressore semplice che limita l'ampiezza dell'RMS a 0.05.

Come si può notare dal grafico della forma d'onda, applicando il compressore semplice all'input, l'RMS risulta troppo uniforme nel tempo creando un sali-scendi continuo dei suoni a livello percettivo. ACE invece ha esaltato i suoni bassi pur mantenendo la dinamicità del brano.

Capitolo 6

Conclusioni

In conclusione l'applicazione dell'algoritmo ACE, studiato per le immagini digitali, su file acustici agisce in modo differente a seconda della tipologia di applicazione. Sulla forma d'onda si comporta esattamente come un filtro, andando a tagliare una serie di frequenze a seconda della funzione distanza utilizzata. Ciò potrebbe risultare utile se si necessita di un equalizzatore abbastanza automatizzato. In particolare la distanza esponenziale inversa a parametro zero offre la soluzione migliore.

Nelle altre applicazioni invece non si verificano variazioni a livello frequenziale. Sullo spettro, ACE si è rivelato poco efficace producendo solo un incremento del rumore di fondo o un riverbero dovuto all'espansione nel tempo delle alte frequenze.

I risultati migliori si sono ottenuti a livello di dinamica nell'applicazione all'inviluppo dove agisce comprimendo l'audio in modo da rendere udibili anche le sonorità più deboli. Tuttavia ciò comporta un limite ovvero una riduzione drastica dei picchi improvvisi nel volume. Una soluzione migliore viene quindi portata dall'algoritmo sull'RMS. In quest'ultimo caso vengono altresì incrementati i suoni bassi, ma la riduzione dei picchi elevati viene ridotta solo leggermente, lasciando invariato il suono dal punto di vista percettivo. Anche a livello di dinamica si è constatato che la versione dell'algoritmo che agisce in modo più pulito risulta essere quella che utilizza una funzione differenza lineare ed una funzione distanza esponenziale inversa con parametro molto rasente allo zero.

6.1 Sviluppi futuri

I test finora eseguiti e che hanno portato alla stesura del presente elaborato sono serviti ad indagare il comportamento dell'algoritmo ACE sulle diverse forme di rappresentazione dell'audio. Un ulteriore serie di verifiche potrebbe portare a stabilire con precisione in che modo avviene il taglio delle frequenze sulla forma d'onda che, come si è constatato, è leggermente differente a seconda dell'input audio utilizzato. Allo stesso modo si potrebbe stabilire in presenza di quali frequenze avviene l'espansione o la compressione della dinamica durante l'applicazione all'RMS. In tal modo si potrebbe rendere l'algoritmo funzionale per una possibile integrazione come plugin ad un editor audio.

Bibliografia

[1] Paolo Spetrino. *Uso dell'analizzatore di spettro per misure su segnali di tele-comunicazione.* Università degli Studi di Napoli "Federico II", 2003.

[2] Tony J. Rouphael. *RF and Digital Signal Processing for Software-Defined Radio.* Newses, 2008.

[3] Alberto Bertoni, Paola Campadelli, Giuliano Grossi. *Introduzione all'elaborazione dei segnali.* Università degli Studi di Milano, 2010.

[4] Rick Jeffs, Scott Holden, Dennis Bohn. *Dynamics Processors: Technology & Applications.* RaneNote 155, 2005.

[5] Alessandro Rizzi, Carlo Gatta, Daniele Marini. *A new algorithm for unsupervised global and local color correction.* Università degli Studi di Milano, 2002.

[6] Steven J. Perry. *Introduction to Java programming, Part 1: Java language basics.* www.ibm.com/developerworks/java/tutorials/j-introtojava1, 2010.

[7] Scott Wilson. *WAVE PCM soundfile format.* CCRMA Stanford University, 2003.

[8] Fabio Marcobelli. *Il restauro di file audio di pellicole cinematografiche.* Università degli Studi di Milano, 2013.

[9] David Houcque. *Introduction to MATLAB for engineering students.* Northwestern University, 2005.