Elaborazione dei Segnali Audio

Angelo Ciaramella

... a tutte le nuvole,
ovunque, nell'immensurabile universo.

Indice

Elenco delle Tabelle

Elenco delle Figure

Introduzione

L'audio digitale, oggi, ricopre un ruolo fondamentale in vari settori come ad esempio la multimedialità, la produzione e la fruizione musicale. Ogni giorno percepiamo, attraverso il nostro sistema uditivo, diversi tipi di suoni. Il nostro cervello distingue i suoni e li classifica in base alle nostre caratteristiche e alle nostre esperienze. Ad esempio siamo capaci di distinguere un rumore da un brano musicale e un gruppo musicale da un altro. Da secoli la musica ricopre un ruolo fondamentale nella vita e nella cultura di un essere umano. Grande testimonianza, infatti, è quello dello sviluppo di diversi strumenti musicali in diverse culture. Ma comunque ancora ci chiediamio cos' é la musica, da dove viene e perchè alcune sequenze di suoni ci toccano mentre altre a tanti danno fastidio.

Il suono, a sua volta, può essere interpretato sia come fonte di intrattenimento, sia come fonte di informazione. Nel cinema e nello spettacolo multimediale, ad esempio, l'informazione video contiene il 90% dell'informazione ma il suono crea il 90% del coinvolgimento.

Inoltre bisogna anche sottolineare che l'analisi dei segnali è alla base di molti settori scientifici come ad esempio astrofisica, geologia, genetica, ecc..

Lo scopo di questo libro è quello di introdurre i concetti fondamentali relativi all'analisi dei segnali e con particolare attenzione ai segnali audio. Esso è un supporto al corso di Elaborazione dei Segnali Audio ed è corredato da una libreria in C++ per l'elaborazione pratica dei segnali. Il libro è stato suddiviso in 9 Capitoli ed ogni argomento teorico è affiancato dall' implementazione di una classe C++ corrispondente. Nel primo Capitolo verranno introdotti i concetti fondamentali dell'audio, dell'acustica e le loro relazioni nel campo musicale. In questo Capitolo vengono introdotti alcuni concetti fondamentali rigurdanti

l'audio e la musica. Nel Capitolo 2 verrà affrontata la problematica relativa alla conversione di un suono da analogico a digitale. In particolare sarà focalizzata l'attenzione sulla rappresentazione digitale del suono e quindi del campionamento e della quantizzazione. Inoltre verrà descritto il Teorema di Campionamento che ci fornisce il limite per la scelta di campioni in un segnale audio analogico. Nel Capitolo 3 vengono analizzati i segnali nel dominio del tempo e nel Capitolo 4 nel dominio delle frequenze. In questo ultimo Capitolo vengono presentate le metodologie o trasformate che permettono di analizzare un segnale audio in un dominio diverso da quello temporale. Verranno introdotte, infatti, la Tasformata di Fourier continua, la Trasformata di Fourier a tempo discreto, la Trasforamta z e la trasformata discreta di Fourier.

Come vedremo inoltre nel Capitolo 5 una delle fasi fondamentali nell'analisi e nella modellazione di un segnale audio è il filtraggio. In questo Capitolo verranno introdotte alcune tecniche per la progettazione di un filtro digitale con particolare attenzione al filtraggio FIR e IIR.

Una delle applicazioni dei filtri è quella per la compressione dei segnali audio. Nel Capitolo 6 saranno considerati gli approcci più conosciuti nell'ambito della compressione audio, come PCM, DPCM, ADPCM, ADPCM adattivo, LPC, MPEG 1 - layer I/II/III e Dolby. Nel Capitolo 7 la nostra attenzione sarà rivolta agli algoritmi basati su linea di ritardo per la costruzione di effetti sonori come ad esempio flanger, echo, doubling ecc.. Inoltre verrà analizzato il problema del riverbero del suono in un ambiente reale. Infine, nel Capitolo 8 vengono presentati alcuni algoritmi per la sintesi sonora come ad esempio additiva, sottrattiva, analisi-risintesi, ecc. e nel Capitolo 9 una breve introduzione del linguaggio C++ per semplificare la comprensione delle classi introdotte nei vari Capitoli del libro.

Napoli, 2 Ottobre 2010

Angelo Ciaramella

Il seguente libro è a disposizione degli studenti dell'Università degli Studi di Napoli "Parthenope" che seguono il corso di Elaborazione dei Segnali Audio di cui è docente Angelo Ciaramella.

Si fa presente che è vietata qualsiasi diffusione impropria del libro. Esso deve essere considerato un supporto didattico esclusivo e consultabile solo dagli studenti che seguono il corso di Elaborazione dei Segnali Audio. Si vieta assolutamente qualsiasi suo utilizzo a scopo di lucro.

Capitolo 1

Audio e Acustica

Il suono (dal latino *sonus*) è la sensazione che si percepisce in relazione alla vibrazione di un corpo. La vibrazione del corpo si propaga nell'aria o in un altro mezzo elastico, raggiungendo l'organo uditivo (orecchio). L'orecchio tramite un complesso meccanismo interno è responsabile della creazione di una sensazione "uditiva" correlata alla natura della vibrazione. Un segnale audio, d'altra parte, è l'informazione elettronica che rappresenta il suono. Esso contiene informazioni percepibili dall'uomo come la voce, le vibrazioni acustiche di strumenti musicali e segnali di sistemi elettroacustici. La natura del suono e i meccanismi naturali di produzione ad esso correlati sono analogici. Ad esempio la variazione di tensione elettrica di un microfono è un'analogia elettronica della variazione di pressione dell'aria. Oggi la tecnologia dell'audio digitale ha superato i limiti dell'elaborazione analogica proponendo metodologie di elaborazione numerica dell'informazione. Ad esempio tra le svariate applicazioni della tecnologia audio digitale possiamo ricordare (vedi Figura 1.1):

- Composizione Musicale
- Audio su internet
- Interfaccia uomo-macchina avanzate
- Sistemi multimediali
- Dispositivi di comunicazione
- Robotica

Figura 1.1: Applicazioni dell'audio digitale.

- ecc.

Tra i domini di applicazione dell'audio digitale possiamo ricordare

- L'elaborazione numerica del segnale o Digital Signal Processing (DSP)
- Codifica e diffusione del segnale audio
- Digital Audio Broadcasting (DAB)
- TV digitale
- Voice Over IP (VOIP)
- Internet radio
- ecc.

Inoltre, oggi esistono svariati supporti di memorizzazione per l'audio digitale come ad esempio

- CD
- Mini disk
- Super Audio CD (SACD)
- Digital Versatile Disk (DVD)

- Lettori MP3 e networking

- ecc.

Lo scopo di questo Capitolo è quello di descrivere i concetti fondamentali del suono e di analizzare i parametri fondamentali che entrano in gioco durante la sua generazione, diffusione e percezione. Prima di passare a questa descrizione focalizziamo la nostra attenzione sulla relazione tra musica, suono e mente umana.

1.1 Suono e musica

Nessuna società umana di oggi, o del passato documentato, è mai stata priva di musica. Alcuni dei più antichi manufatti rinvenuti in siti archeologici umani e protoumani sono strumenti musicali: flauti d'osso e tamburi fatti di pelli di animali tirate su ceppi d'albero. Ogni volta che degli esseri umani si riuniscono, per qualsiasi motivo, c'è anche la musica: matrimoni, funerali, eventi sportivi, la preghiera, una cena romantica, una mamma che culla il suo bambino ecc.. Nella nostra cultura, solo in tempi relativamente recenti, è sorta una distinzione the taglia la società in due, dando origine a due classi separate: gli esecutori e gli ascoltatori. Nella maggior parte del mondo e per la maggior parte della storia umana, fare musica era un'attività naturale quanto respirare e camminare, e tutti vi partecipavano.

Molti scienziati si occupano dello studio dell'effetto della musica sul cervello e la mente. Capire perchè ci piace la musica e cosa ci attragga in essa può aprirci una finestra sull'essenza della natura umana e dell'evoluzione.

Un ambito scientifico relativamente nuovo è la *psicologia evolutiva*. Non solo i nostri corpi, ma anche le nostre menti, la nostra predisposizione a risolvere i problemi in certi modi, i nostri sistemi sensoriali, sono tutti prodotto dell'evoluzione. Per i non musicisti, le macchie d'inchiostro definite come "notazione musicale" potrebbero essere benissimo le notazioni di una certa teoria matematica. Parlare di chiavi, cadenze, modulazione e trascrizione può sicuramente confondere. Eppure tutte le persone intimidite da un simile lessico non hanno problemi a dire quale musica amino.

1.1.1 Componenti elementari della musica

La musica ha un effetto diretto sul nostro cervello, sulla nostra mente, sui nostri pensieri e sul nostro umore da una prospettiva neuropsicologica. E' utile a questo punto esaminare le componenti elementari della musica e come sono organizzate per dare origine alla musica. Gli elementi fondamentali di ogni suono sono: **intensità**, **pitch**, **profilo melodico**, **durata** (o **ritmo**), **tempo**, **timbro**, **posizione spaziale** e **riverbero**. I nostri cervelli organizzano questi attributi percettivi fondamentali in concetti di livello superiore (come un pittore organizza le righe in forme) che includono **metro**, **armonia** e **melodia**. Quando ascoltiamo la musica, stiamo in realtà percependo più attributi o "dimensioni". I termini principali sono

- **Pitch**

  Il pitch o altezza del suono è una costruzione puramente psicologica, correlata sia alla frequenza reale di un particolare tono (o nota) che alla sua relativa posizione nella scala musicale. Dà la risposta alla domanda: *che nota è questa*?

- **Ritmo**

  Esso si riferisce alla durata di una serie di note e al modo in cui esse si uniscono in unità.

- **Tempo**

  Il tempo si riferisce alla velocità globale della composizione. Quando si batte il piede, si balla o si marcia seguendo un pezzo, il tempo è la velocità o la lentezza di questi moviementi regolari.

- **Profilo melodico**

  Descrive la direzione generale di una melodia e prende in considerazione solo la sequenza di "su" e "giù" (cioè se la nota scende o sale, ma non di quanto)

- **Timbro**

  Distingue uno strumento da un altro (ad esempio una tromba da un pianoforte) quando entrambi suonano la stessa nota. E' una specie di colore tonale che viene prodotto in parte grazie agli ipertoni delle vibrazioni di uno strumento. Descrive anche il modo in cui un singolo strumento può cambiare suono spostandosi all'interno della propria estensione.

- **Intensità**

  L'intensità o volume sonoro è una costruzione puramente psicologica che ha a che fare con la quantità di energia creata da uno strumento e con quello che un esperto di acustica chiamerebbe l'ampiezza di un tono.

- **Riverbero**

  Si riferisce alla percezione della distanza che ci separa dalla sorgente, unita alla dimensione della stanza o della sala in cui c'è la musica.

Gli psicofisici hanno dimostrato che questi attributi sono separabili. Ciascuno può essere modificato senza alterare gli altri, e questo ci permette di studiarli uno alla volta. La differenza tra la musica e una serie casuale e disordinata di suoni dipende dal modo in cui si combinano questi attributi. Quando questi elementi basilari si combinano a formare delle relazioni in modo sensato, danno origine a concetti di ordine superiore come il metro, la tonalità, la melodia e l'armonia. In dettaglio

- **Metro**

  Viene creato dal nostro cervello, che estrae informazioni dal ritmo e dall'intensità e si riferisce al modo in cui i toni sono raggruppati uno con l'altro nel tempo. Il metro di un valzer, ad esempio, organizza i toni a gruppi di tre, quello di una marcia a gruppi di due o quattro.

- **Tonalità**

  Ha a che fare con la gerarchia d'importanza tra i toni in un pezzo musicale; essa non esiste ma solo nelle nostre menti, quale funzione delle nostre esperienze con uno stile musicale e con i linguaggi musicali e con gli schemi mentali che noi sviluppiamo per capire la musica.

- **Melodia**

  E' il tema principale di un pezzo musicale, la parte che canticchiamo, la successione di toni più rilevanti nella nostra mente. La nozione di melodia varia a seconda dei generi. Nella musica rock, c'è una melodia per la strofa e una per il ritornello, e la strofa si distingue per un cambio di testi e talvolta di strumentazione. Nella musica classica la melodia è un punto di partenza: il compositore crea variazioni su quel tema, che può ritornare per tutto il pezzo sotto forme diverse.

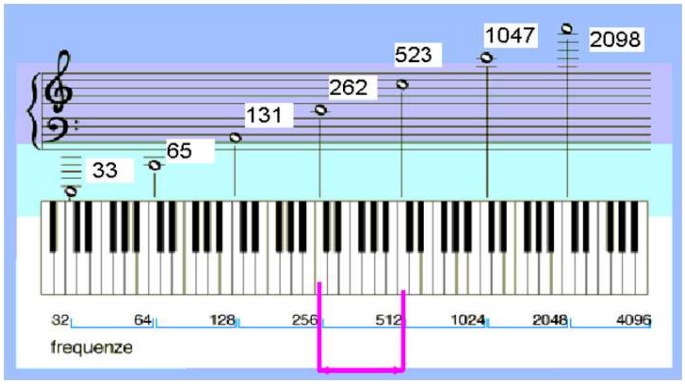
- **Armonia**

  Ha a che fare con le relazioni tra i pitch di toni diversi e con i contesti tonali che questi pitch creano inducendo delle aspettative sul prosieguo del pezzo musicale.

L'idea di elementi primari che si combinano a creare arte, e dell'importanza delle relazioni tra questi elementi esiste nell'arte visiva e nella danza. *Miles Davis* descrisse la sua tecnica di improvvisazione come Picasso fece con il suo uso della tela: *l'aspetto cruciale dell'opera, dissero i due artisti, non erano gli oggetti in se, ma lo spazio tra di essi.*

1.1.2 Pitch e suono

Quasi tutti noi, anche senza aver studiato musica, possiamo dire se un cantante ha preso una stecca. Ciò è dovuto a un'interazione tra la nostra esposizione alla musica e la fisica del suono. Quello che chiamiamo pitch è correlato alla frequenza o velocità di vibrazione di una corda, di una colonna d'aria o di altre sorgenti fisiche. Se una corda vibrando si muove avanti e indietro 60 volte in un secondo, diciamo che ha una frequenza di 60 cicli al secondo. L'unità di misura, cicli per secondo, viene detta Hertz (Hz) in onore di Heinrich Hertz, il fisico teorico tedesco che trasmise per primo le onde radio. Per capire il concetto di pitch diamo un'occhiata alla tastiera del pianoforte della Figura 1.2. I tasti sulla sinistra colpiscono corde più lunghe e spesse che vibrano a una velocità relativamente lenta. I tasti sulla destra colpiscono corde più corte e sottili che vibrano a maggior velocità. La distanza percorsa dalla corda a ogni oscillazione (fino a fermarsi) viene traslata dal nostro cervello in intensità, e la velocità di oscillazione in pitch. Più lontano va la corda, più il suono ci sembra intenso; quando si muove a stento, il suono ci sembra invece debole. La distanza percorsa e la velocità di oscillazione sono indipendenti. La velcotià a cui vibra la corda è dovuta principalmente alla sua dimensione e alla tensione cui è tirata, non alla forza con cui viene colpita. Le onde sonore urtano contro il timpano e il padiglione auricolare, scatenando una catena di eventi (come vedremo in seguito) meccanici e neurochimici il cui prodotto finale è un'immagine mentale che chiamiamo pitch. Infatti ci possiamo chiedere: *se in una foresta cade un albero e nessuno è li a sentire lo schianto, fa rumore*? La risposta è no, il suono è un'immagine mentale creata dal cervello in risposta a molecole che vibrano (stesso meccanismo dei colori come ipotizzato da Newton).



do_4	$do\#_4$	re_4	$re\#_4$	mi_4	fa_4	$fa\#_4$	sol_4	$sol\#_4$	la_4	$la\#_4$	si_4
261,63	277,18	293,66	311,13	329,63	349,23	369,99	392	415,30	440	466,16	493,88

Figura 1.2: Tastiera di un pianoforte e frequenze.

Un normale pianoforte ha 88 tasti. La nota più bassa vibra con una frequenza di 27.5 Hz e quella più alta intorno a 6000 Hz. Il pitch è uno dei mezzi essenziali con cui la musica comunica emozioni. Stato d'animo, eccitazione, calma, romanticismo e pericolo sono segnalati da una serie di fattori, ma il pitch è tra i più importanti. Una singola nota alta può comunicare eccitazione e una bassa tristezza. Le melodie sono definite da diversi pitch successivi. Una melodia è un oggetto acustico che mantiene la sua identità nonostante le trasformazioni (diverse tonalità).

Come vedremo nei Capitoli successivi, la membrana basilare dell'orecchio interno permette la distinzione dei pitch. La membrana invia un segnale elettrico fino alla corteccia uditiva. Quest'ultima ha una mappa tonotopica, con i toni alti e bassi disposti lungo la superficie corticale. Anche il cervello contiene una mappa dei diversi pitch e aree diverse del cervello lo rappresenta direttamente.

1.1.3 Le note

Una **scala** è un sottoinsieme del numero di pitch e ogni cultura seleziona quelli che affondono nella tradizione storica o si affida a una certa arbitrarietà. I pitch scelti vengono in seguito consacrati come parte di quel sistema musicale. Si tratta dei nomi come Do,

Re, Mi, ecc.. I nomi delle **note** nella musica occidentale vanno da A a G o nel sistema alternativo Do-Re-Mi-Fa-Sol-La-Si-Do. I nomi delle note si ripetono a causa di un fenomeno percettivo che corrisponde a raddoppiare e dimmezzare le frequenze. Quando lo facciamo, otteniamo una nota che assomiglia straordinariamente a quella di partenza. Questa relazione (rapporto di frequenza $1:2$) viene detto **ottava**. Ogni cultura conosciuta ha l'ottava alla base della musica, anche quando ha poco altro in comune con le altre tradizioni. Questo fenomeno porta al concetto di circolarità nella percezione del pitch. Quando uomini e donne parlano all'unisono, normalmente le loro voci sono separate da un'ottava. Vediamo ora quale può essere la correlazione tra note e frequenze. In Figura **??** viene rappresentata una tastiera del pianoforte su cui sono evidenziate le frequenze relative ai Do (e sono $32 - 64 - 128 - 256 - 512 - 1024 - 2048 - 4096$). Viene anche descritta la corrispondente notazione musicale sul pentagramma e la relazione tra pentagramma e quarta ottava. I Do a intervalli di ottava hanno una frequenza doppia della precedente. L'ottava centrale del pianoforte contiene il *La* del diapason (440 Hz) ma potremmo fissare qualsiasi frequenza. All'epoca di Mozart si usavano standard diversi rispetto ad oggi.

Un **intervallo** è la distanza tra due toni. Nella musica occidentale, l'ottava è suddivisa in 12 toni separati da una distanza uguale (logaritmica). La distanza tra Do e Re è detta "intervallo di seconda maggiore" o **tono**. Tagliando a metà un intervallo di seconda maggiore otteniamo un **semitono** (1/12 di ottava). Gli intervalli sono alla base della melodia.

A questo punto ci chiediamo: perchè i nomi sono solo sette, Se in un'ottava ci sono 12 note? La risposta è che le cinque note aggiuntive hanno dei nomi composti come *Mib* (Mi bemolle) e *Fa#* (Fa diesis). Il sistema diventa un pò più chiaro se osserviamo la tastiera di un pianoforte (Figura 1.2). I tasti bianchi sono il Do, Re, Mi, Fa, Sol, La, Si. Le note intermedie e cioè i tasti neri sono quelle con i nomi composti. La nota tra La e Si è detta "La diesis" o "Si bemolle". Diesis significa alto e bemolle basso.

Possiamo fissare i pitch dovunque vogliamo perchè ciò che definisce la musica è una serie di relazioni tra di essi. La distanza tra ogni nota e la successiva ci suona uguale. Ogni tono è il 6% più alto del precedente e quando aumentiamo un tono del 6% per 12 volte, raddoppiamo la frequenza originale (la proporzione reale è $\sqrt[12]{2} = 1.059463\ldots$). Le 12 note del nostro sistema musicale sono la cosiddetta **scala cromatica**. Nella musica occidentale è raro che per comporre si usino tutte le note della scala cromatica ma piuttosto usiamo

un sottoinsieme di sette. Il sottoinsieme più usato si chiama la **scala maggiore** o **modo ionico**. In ogni scala maggiore, l'ordine degli intervalli è: tono, tono, semitono, tono, tono, tono, semitono. In altre parole le note della scala maggiore sono Do-Re-Mi-Fa-Sol-La-Si. Sequenze diverse di toni e semitoni danno origine a scale alternative, la più comune delle quali è la scala minore. Per ragioni principalmente culturali, tendiamo ad associare alle scale maggiori delle emozioni gioiose o trionfali e a quelle minori delle emozioni tristi o di sconfitta. La musica *blues* di solito usa una scala a 5 note (*pentatonica*) che è un sottoinsieme della scala minore.

Un **accorodo** è un gruppo di 3 o più note suonate contemporaneamente. Un tipico accordo si ha suonando insieme la prima, la terza e la quinta nota di una scala. Se costruiamo un accordo a partire da un Do e impieghiamo i toni della scala di Do maggiore, usiamo Do, Mi e Sol. Se invece adottiamo la scala di Do minore, la prima, terza e quinta nota sono Do, Mi bemolle e Sol.

Quando si genera un suono con un pianoforte, un flauto o qualsiasi altro strumento, inclusi strumenti a percusione, questo produce diversi tipi di vibrazione contemporaneamente. Quando ascoltiamo una singola nota emessa da uno strumento, in realtà stiamo ascoltando molti pitch nello stesso momento e non uno solo. La frequenza di vibrazione più lenta, il pitch più basso, è detta **frequenza fondamentale** mentre le altre vengono chiamate **ipertoni** o **armoniche**. Gli oggetti, quindi, vibrano a diverse frequenze per volta. Queste altre frequenze sono una il multiplo intero dell'altra. Se pizzichiamo una corda e la sua frequenza di vibrazione più lenta è di 100 volte al secondo, le altre frequenze di vibrazione sranno di 2x100 = 200 Hz, 3x100 = 300 Hz, ecc..

Il cervello è talmente sintonizzato sulla serie degli armonici che se incontriamo un suono che possiede tutte le componenti tranne la fondamentale, il cervello la aggiunge per noi in un fenomeno detto "ripristino della fondamentale mancante".

Non tutti gli strumenti vibrano in modo definito. Talvolta come nel caso di strumenti a percussione gli ipertoni possono essere "quasi" multipli della frequenza fondamentale e ciò contribuisce al loro caratteristico suono.

Il **timbro** è la caratteristica principale che distingue due suoni ed è una conseguenza degli ipertoni. Quando sentiamo un sassofono suonare un tono con una frequenza fondamentale di 220 Hz, in realtà stiamo sentendo molti toni, non uno solo. Gli altri toni che sentiamo sono multipli interi della fondamentale: 440, 660, 880, 1100, 1320, 1540, ecc. Questi

ipertoni hanno una diversa forza e quindi li sentiamo come se avessero una differente intensità. Il particolare pattern di intensità è tipico del sassofono e si deve ad essi il suo colore tonale unico e il suo suono inconfondibile: il suo timbro.

Ciascuno strumento ha il suo profilo di ipertoni, che è come un'impronta digitale. I clarinetti, ad esempio, si contraddistinguono per avere quantità relativemante alte di energia negli armonici dispari (è una conseguenza dela fatto che si tratta di tubi chiusi a un'estremità a aperti all'altra). Le trombe invece si contradddistinguono per avere quantità relativamente uniformi di energia negli armonici pari e nei dispari. Un violino suonato al centro con l'archetto produrrà soprattutto armonici dispari e quindi risulterà simile ad un clarinetto. Ma se l'archetto tocca le corde un terzo più in basso, lo strumento enfatizza il terzo armonico e i suoi multipli: sesto, nono, dodicesimo, ecc.

Gli strumenti naturali tendono a produrre energia a più frequenze per volta, a causa del modo in cui vibra la struttura interna delle loro molecole.

La fase di attacco e cioè l'introduzione di energia in uno strumento di solito produce energia a molte frequenze diverse che non sono correlate tra loro da multipli interi. In altre parole, dopo che abbiamo colpito, soffiato, pizzicato o comunque fatto suonare uno strumento, per una frazione di secondo l'impatto ha un che di rumoroso e non musicale. Dopo l'attacco c'è una fase più stabile (come vedremo in seguito). Negli anni Cinquanta, il compositore d'avanguardia Pierre Schaeffer svolse alcuni esperimenti che hanno dimostrato un importante attributo del timbro. Dopo che Schaeffer ebbe tagliato l'attacco, riascoltò il nastro scoprendo che alla maggior parte della gente risultava quasi impossibile identificare lo strumento che stava suonando. Senza l'attacco, il suono di pianoforti e campane era incredibilmente simile.

Comunque, anche se il timbro sia il centro della valutazione della musica, il ritmo ha dominato gli ascoltatori per molto più tempo.

1.1.4 Ritmo, intensità e armonia

Per tutte le culture e civiltà il movimento è parte integrante della musica, suonata e ascoltata. E' in base al ritmo che balliamo, che ondeggiamo il nostro corpo e che battiamo il piede. Quando organizziamo i ritmi in serie di note, con varie lunghezze ed enfasi, sviluppiamo il metro e stabiliamo il tempo.

Il **tempo** si riferisce alla velocità di un pezzo musicale cioè quanto rapidamente o lentamente si svolge. Se battiamo il piede o schiocchiamo le dita seguendo un brano, il tempo del pezzo è in diretta relazione con la velocità o lentezza con cui battiamo. Il **battito** indica l'unità elementare di misura di un brano musicale. Ad esempio un battito di 96 significa che ci sono 96 battiti per minuto (usiamo il metronomo per controllarlo). Il tempo è un fattore molto importante nell'espressione delle emozioni. Le canzoni con un tempo veloce tendono a essere considerate allegre e quelle con tempo lento malinconiche. La base neurale di questa accuratezza è il cervelletto che contiene una specie di metronomo per la nostra vita quotidiana e per sincronizzarsi sulla musica che stiamo sentendo.

Il **metro** si riferisce al modo in cui le pulsazioni o battiti sono raggruppati insieme. In generale, quando battiamo un piede o le mani a ritmo di musica, sentiamo alcuni battiti più di altri. Lo schema più comune della musica occidentale prevede un battito forte ogni quattro: FORTE-debole-debole-debole. Più raramente il battito forte è uno ogni tre e cioè il battito valzer.

I tre metri più comuni della musica occidentale sono: 4/4, 2/4 e 3/4. Esistono anche altri raggruppamenti ritmici come 5/4, 7/4, 9/4 e 6/8.

Infine, l'**intensità** è la percezione che il cervevello associa all'altezza del suono. Le intensità non sono additive come le ampiezze (come vedremo in seguito la scala è quella logaritmica) ed è misurata in decibel (dB).

La maggior parte della gente non può isolare il suono di un singolo violino all'interno di un'orchestra: i violini formano un gruppo. L'intera orchestra può formare un singolo gruppo percettivo. Quando raggruppa i suoni il nostro cervello sfrutta la serie degli armonici. Se sentiamo una tromba invece dei singoli ipertoni che colpiscono le nostre orecchie; si raggruppano insieme come fili d'erba che ci danno l'impressione di un prato. Per distinguere una tromba da un oboe che suonano la stessa nota il sistema uditivo fa affidamento al principio di attacchi simultanei.

1.1.5 Musica e mente

Perchè a volte le canzoni ci entrano in testa e non riusciamo più a liberarcene? L'attività musicale coinvolge quasi ogni regione del cervello e quasi ogni sottosistema neurale. Aspetti differenti della musica vengono gestiti da regioni neurali differenti in modo da eccitare parti del cervello adibite alla cognizione di differenti emozioni. I neuroni, ad esempio, che si

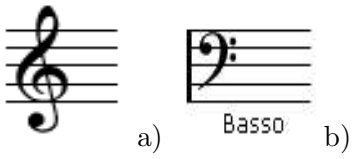


Figura 1.3: Chiavi musicali: a) violino (o sol): b) basso.

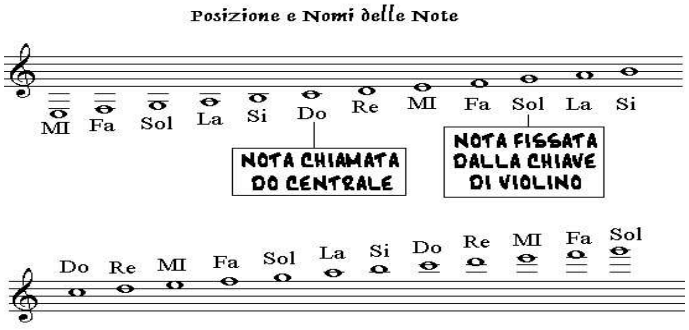


Figura 1.4: Posizione note sul pentagramma secondo la chiave di Violino.

attivano quando mi sento tranquillo e al sicuro nel mio ambiente possono essere stimolati dalli parti calme di un concerto classico o una canzone.

1.1.6 Note e pentagramma

La durata standard di una nota è detta *semibreve* e dura quattro battiti, indipendentemente dal tempo della musica. Essa viene rappresentata con un cerchio vuoto.
La **chiave musicale** è un simbolo che viene posto sul pentagramma e serve a fissare la posizione delle note e la relativa altezza dei suoni. Due esempi di chiave sono quella di Sol o di Violino o quella di basso come è illustrato in Figura 1.3. Può essere posta all'inizio del pentagramma (maggioranza dei casi) oppure in un punto qualsiasi (ad es. a metà di una battuta o misura). I segni delle chiavi provengono da una progressiva alterazione grafica delle lettere dell'alfabeto gotico. Le note vengono rappresentate sul **pentagramma**. Il pentagramma è formato da 5 linee e 4 spazi tra i quali e sopra e sotto vengono scritte le note. In Figura 1.4 vengono presentate le posizioni delle note sul pentagramma. Generalmente il pentagramma viene diviso in diverse **battute** come in Figura 1.5 per formare uno **spartito musicale**. All'inizio di un **partitura** (sequenza

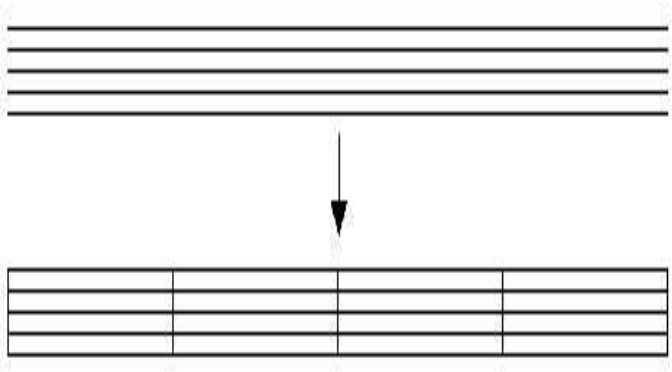


Figura 1.5: Suddivisione del pentagramma in battute.

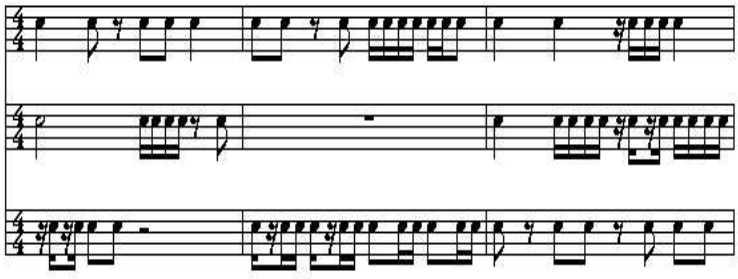


Figura 1.6: Esempio di spartito per batteria.

di note sul pentagramma) viene stabilito il tempo di esecuzione. Ad esempio possiamo fissare il nostro tempo ad un valore di 4/4, 3/4, 6/8, ecc. In Figura 1.6 viene rappresentata una tipica partitura musicale relativa a uno strumento a percussione (ad esempio rullante bandistico o di una batteria). Per capire il significato di questa partitura deve essere definito il valore delle note. In Figura 1.7 viene proposto un confronto completo tra le diverse note e il loro significato temporale all'interno di una battuta. Le note che introduciamo sono la nota *semibreve* che ha un valore di 4/4 (riferirsi all Figura 1.7 dove ci sono le note e le corrispondenti **pause**). Inoltre vengono rappresentate le note *minima* con valore di 2/4, la nota *semiminima* con un valore di 1/4, la nota *croma* che ha un valore di 1/8, la *semicroma* invece ha un valore di 1/16 e infine la *biscroma* ha un valore di 1/32.

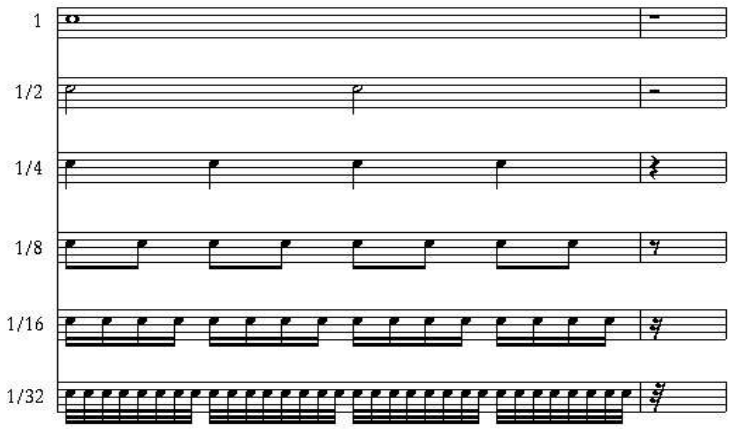


Figura 1.7: Valore delle note e rispettive pause.

1.2 Elaborazione del segnale

La riproduzione di un segnale può avvenire in svariati modi. Ad esempio un segnale acustico può essere ottenuto tramite uno strumento musicale tradizionale o uno strumento elettronico. Dobbiamo precisare che nelle varie fasi di riproduzione e ascolto di un segnale acustico un ruolo fondamentale è ricoperto dai sistemi di *elaborazione del segnale* o *Digital Signal Processing* (DSP). Sostanzialmente il DSP comprende tutte le tecniche di analisi dei segnali analogici o digitali utili per elaborare un suono. Consideriamo come esempio la fase di incisione di un brano musicale in una sala di registrazione discografica. Come possiamo immaginare abbiamo a disposizione diversi strumenti musicali sia acustici che elettronici (sintetizzatori) o elettroacustici (chitarre elettriche, ecc.) e inoltre possiamo avere la necessità di usare equalizzatori di voce. Per avere un'elevata qualità di ascolto dobbiamo sicuramente usare un sistema per la ripresa e la miscelazione dei segnali audio. Uno dei componenti fondamentali risulta, infatti, il *mixer* (vedi Figura 1.8). Esso è usato essenzialmente per la ripresa, la post-produzione e l'elaborazione del segnale audio. Un mixer può essere visto come una matrice con N canali di ingresso e M canali in uscita. Ogni canale di ingresso è dedicato ad un solo dispositivo e quindi permette di effettuare le regolazioni sul segnale corrispondente. Il segnale, a sua volta, può essere inviato contemporaneamente ad una o più uscite e per ognuna delle uscite è possibile prevedere un trattamento diversificato. La parte principale di un mixer è la *consolle di missaggio* che tratta singolarmente, o in sottogruppi, i vari canali audio di ingresso, per miscelarli e dis-

Figura 1.8: Mixer.

tribuirli opportunamente. Il percorso del segnale (*data path*) ha vari livelli di elaborazione e controllo:

- *Guadagni* (livelli dei segnali)
- *Filtri* (ad esempio passa basso e passa alto)
- *Equalizzatori* (variazione della risposta in frequenza)
- *Delay* (inserimento di ritardi)
- *Effetti* (flanger, chorus, wha-wha, ec..)
- *Controllo dinamico* (compressori ed espansori)

Come possiamo notare, quindi, il segnale acustico può essere elaborato in diversi modi durante il suo percorso. Nel seguito di questo libro estenderemo e cercheremo di approfondire sia da un punto di vista teorico che pratico questi concetti.

1.3 Fondamenti di acustica

Come abbiamo accennato precedentemente, il suono è un fenomeno meccanico derivante dalla perturbazione di un mezzo di trasmissione e percepito dall'orecchio umano. Le vibrazioni sono generate da differenti oggetti come ad esempio gli strumenti musicali (pianoforte, chitarra, violino, percussioni, batteria, ecc.). Il suono percepito può essere definito coma una variazione ciclica, rispetto ad un valore costante, della pressione dell'aria. Questo ci permette di formalizzare un concetto fondamentale nella teoria dei segnali: *la*

natura fisica del suono è di tipo ondulatorio.
I suoni in natura, comunque, sono complessi da analizzare e possono essere definiti come combinazioni intenzionali (e disciplinate) di altri segnali indipendenti. Ad un suono può essere assegnata un'informazione come nel caso della musica o del linguaggio.
Il termine rumore viene usato generalmente per indicare tutti gli altri suoni che sono non organizzati, non piacevoli, o non voluti. Prima di analizzare e caratterizzare i suoni e quindi i segnali vediamo in dettaglio come essi possono essere generati e come si possono propagare.

1.3.1 Generazione del suono

Abbiamo già sottolineato che i suoni che percepiamo derivano da vibrazioni di sorgenti sonore. Tutte le sorgenti sonore oscillano ed ogni vibrazione completa è detta ciclo. Ogni segnale sonoro comprende molti cicli. I moti ciclici possono essere il pendolo, peso attaccato ad una molla (come vedremo successivamente), i moti di un atomo in uno stato solido, il moto di una corda, ecc. Per capire il moto oscillatorio consideriamo un *diapason*. Esso è uno strumento che ci permette di generare una nota standard (La_4 per un musicista).
Come rappresentato in Figura 1.9, il diapason è costituito da una forcella di acciaio con un manico, anch'esso di acciaio, saldato alla base. Quest'ultimo consente di tenere lo strumento senza ostacolare l'oscillazione della forcella, e di trasmettere le vibrazioni, se colpito da un martelletto. Nella musica esso è generalmente utilizzato per l'accordatura degli strumenti musicali. Supponiamo di colpire la sua punta con un martelletto e di osservare il suo andamento nel tempo. Mentre il pennino alla punta del diapason vibra, disegna una curva sinusoidale. Infatti, se sull'asse rappresentiamo il tempo la curva che rappresenta la posizione della particella è sinusoidale come si può vedere nella Figura 1.10.

1.3.2 Propagazione del suono

Le onde possono essere suddivise in *longitudinali* per le quali l'asse lungo il quale avviene la vibrazione è lo stesso della direzione di propagazione dell'onda e *trasversali* in cui l'asse della vibrazione è perpendicolare alla direzione di propagazione dell'onda. Il segnale sonoro si propaga come un'onda longitudinale. Per capire come si propaga un suono, come sor-

Figura 1.9: Diapason.

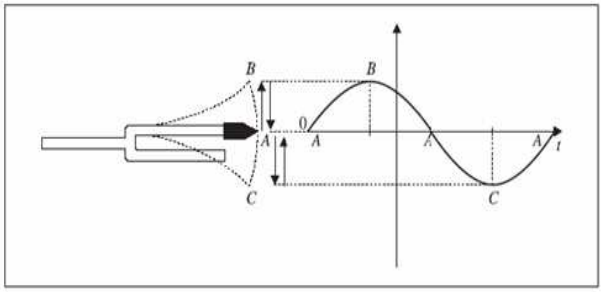


Figura 1.10: Diapason e curva sinusoidale.

gente sonora, consideriamo un altoparlante. In un diffusore di questo tipo (vedi Capitolo 2 per maggiore dettagli) il magnete si muove avanti e indietro seguendo l'ampiezza del segnale elettrico che viene applicato all'induttore su cui si appoggia. In questo modo una membrana sposta le particelle d'aria, nel suo intorno, comprimendole prima e dilatandole poi.

Il suono generato dall'altoparlante si propaga nell' aria mediante collisioni multiple tra particelle. Con riferimento alla Figura 1.11 identifichiamo chiaramente una fase di *compressione* (C) e una fase di *rarefazione* (R) delle particelle dell'aria. Sostanzialmente abbiamo che l'altoparlante si muove e spinge le particelle d'aria ottenendo in questo modo una compressione (fase C) delle particelle. Queste, a loro volta vanno a spingere le particelle che sono a loro vicine e trasferiscono loro l'energia che hanno ricevuto dall'altoparlante. In seguito l'altoparlante torna indietro ed esegue una compressione nel verso opposto ovvero una dilatazione verso sinistra, fase di rarefazione (fase R) e nel fare ciò crea una depressione davanti a se che viene colmata dalle particelle d'aria che si trovano nelle immediate vicinanze. Queste particelle che si muovono creano a loro volta una depressione alla loro

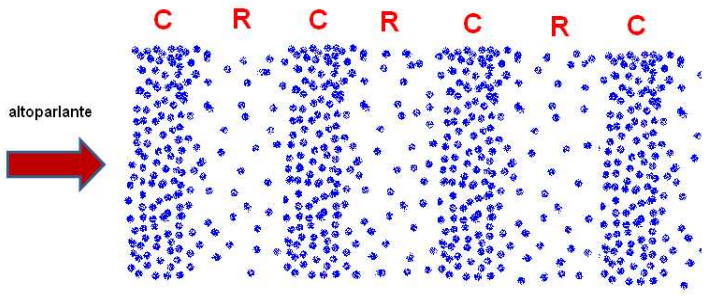


Figura 1.11: Fase di compressione e rarefazione.

destra e cosi via.

1.4 Il tono puro

In questa sessione descriviamo e caratterizziamo il suono che è alla base dei sistemi di elaborazione dei segnali e cioè il *tono puro*. Il tono puro non è altro che un segnale sinusoidale semplice. Esso è fondamentale in quanto in natura le forme d'onda sono molto complesse ma come vedremo successivamente tutte possono essere caratterizzate da un insieme di segnali sinusoidali (o toni puri). Il tono puro è caratterizzato da un'unica frequenza e non esiste come tale in natura. L'unico strumento che riproduce un tono quasi puro è il diapason. Esso produce un suono con una frequenza precisa che per un musicista è la nota *La* e corrisponde ad una fissata frequenza (440 Hz). La forma d'onda del tono puro coincide con la funzione trigonometrica sinusoidale (come nel caso dell'esperimento con il diapason)

$$y(t) = A\sin(t) \tag{1.1}$$

Apparentemente l'equazione del tono puro sembra semplice ma come vedremo essa possiede diverse proprietà scaturite ai seguenti parametri

- *Frequenza* (f)
- *Periodo* (T)

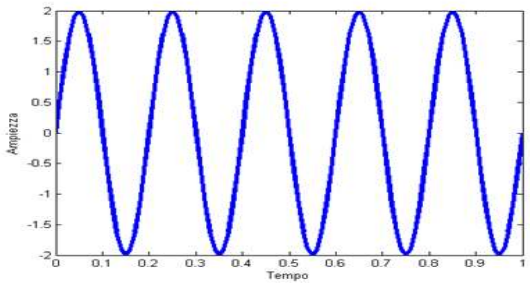


Figura 1.12: Tono puro con frequenza di 5 Hz.

- *Lunghezza d'onda* (λ)
- *Ampiezza* (A)
- *Fase* (ϕ)
- *Fase iniziale* (ϕ_0)
- *Pulsazione* (ω)
- *Velocità* (v)

Successivamente analizzeremo, caso per caso, le propietà di questi parametri.

1.4.1 La frequenza

La *frequenza* è definita come il numero di cicli compiuti dall'onda in un secondo. Il ciclo consiste di una semi-onda positiva e una semi-onda negativa. La frequenza è misurata in Hz ($1/sec$). Quindi, ad esempio, la frequenza pari a 1 Hz corrisponde ad un ciclo ogni secondo. In Figura 1.12 viene rappresentato un tono puro con una frequenza di 5 Hz.

1.4.2 Il periodo

Il *periodo* è il tempo impiegato per compiere un ciclo completo ed è inversamente proporzionale ala frequenza. Esso corrisponde, infatti, a

$$T = \frac{1}{f} \tag{1.2}$$

In Figura 1.13 viene presentato un esempio di tono puro e il corrispondente periodo.

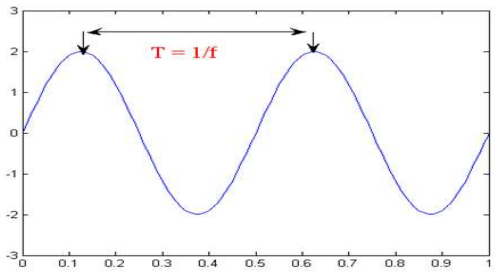


Figura 1.13: Periodo nel tono puro.

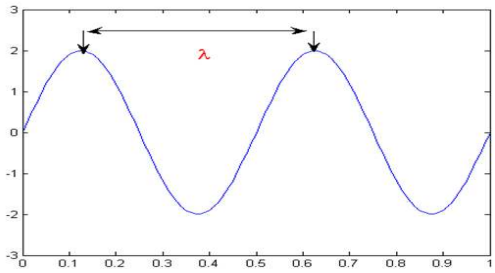


Figura 1.14: Diapason e curva sinusoidale.

1.4.3 La lunghezza d'onda

La *lunghezza d'onda* corrisponde alla distanza tra due punti corrispondenti lungo la forma d'onda. Il suo valore è espresso dalla seguente espressione

$$\lambda = \frac{c}{f} \tag{1.3}$$

dove c è la velocità del suono nel mezzo che si sta considerando (ad esempio nell'aria è 344 m/sec). In Figura 1.14 viene presentato un esempio di tono puro e lunghezza d'onda. Ad esempio un'onda di frequenza 1 Hz che viaggia nell'aria ha la seguente lunghezza d'onda

$$\lambda = \frac{c}{f} = \frac{344 \text{ m/s}}{1 \text{ 1/s}} = 44\text{m} \tag{1.4}$$

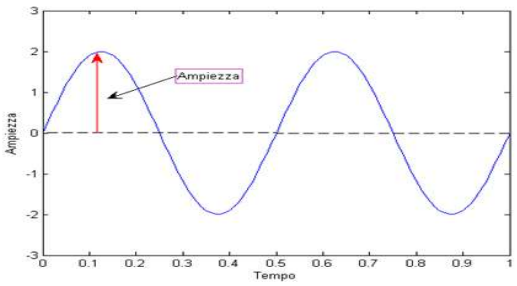


Figura 1.15: Diapason e curva sinusoidale.

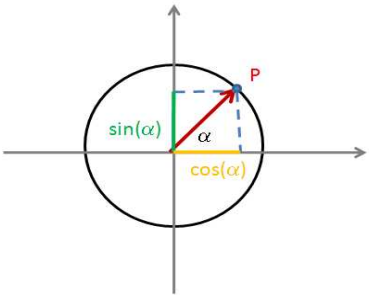


Figura 1.16: Punto P che si muove su una circonferenza.

1.4.4 Ampiezza

L'*ampiezza* è la misura dello scostamento massimo dalla posizione del tono puro dall'equilibrio. Naturalmente ampiezze maggiori corrispondono a volumi più alti. Il tono puro in Figura 1.15 viene rappresentato con ampiezza 2.

1.4.5 Fase e Fase iniziale

Per introdurre il concetto di *fase* faremo riferimento alla Figura 1.16. Immaginiamo che il punto P si muova lungo la circonferenza in senso antiorario a partire dal punto a 0 gradi. Se α è l'angolo che forma il segmento che congiunge l'origine a P, avremo che i segmenti di proiezione del punto P sugli assi y e x saranno rispettivamente rappresentati dal seno ($\sin(\alpha)$) e dal coseno ($\cos(\alpha)$) dell'angolo. Immaginiamo ora di ruotare il punto P in senso antiorario, la sua proiezione sulle y sarà all'inizio positiva e avrà un andamento sinusoidale. Se rappresentiamo il segnale nel tempo, come abbiamo fatto nel caso dell'esempio con il diapason, questo rappresenta la funzione trigonometrica seno. In Figura 1.17 viene

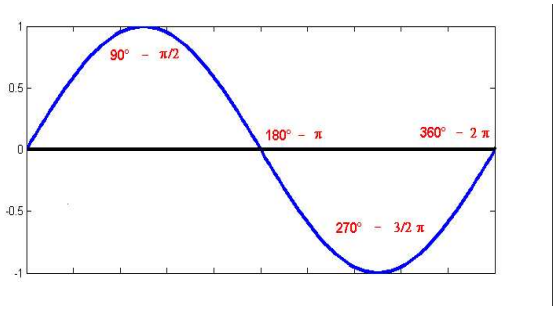


Figura 1.17: Funzione trigonometrica seno.

rappresentato il segnale seno nel tempo.
Vediamo ora come mettere in correlazione la frequenza del tono puro, il tempo e la funzione trigonometrica che abbiamo appena introdotto. Essendo la funzione seno periodica di periodo 2π, possiamo dare un'interpretazione alternativa della frequenza e cioè può essere vista come il numero di volte che il punto P compie un giro completo in un secondo. Ad esempio se la frequenza è 1 Hz abbiamo un giro al secondo. Quindi l'equazione che lega la fase ϕ (angolo α precedentemente) al tempo risulta

$$\phi = 2\pi \triangle t \tag{1.5}$$

dove $\triangle t = t - t_0$ e se l'istante iniziale $t_0 = 0$ $\triangle t$ coincide con t. La *fase iniziale* ϕ_0 è lo scostamento iniziale da dove si inizia ad osservare il tono puro. Ad esempio potrebbe essere un angolo come $\pi/2$.

1.4.6 Pulsazione

La *pulsazione pura* (o risonanza) è definita come

$$\omega = 2\pi f \tag{1.6}$$

dove f è la frequenza del tono puro. L'interpretazione della pulsazione è quella della frequenza (velocità angolare) espressa in radianti.

1.4.7 Tono puro completo

Considerando i paramenti introdotti precedentemente, possiamo caratterizzare la forma d'onda completa del tono puro. Essa risulta complessivamente

$$y(t) = A\sin(\phi + \phi_0) = A\sin(\omega\triangle t + \phi_0) = A\sin(2\pi f\triangle t + \phi_0). \tag{1.7}$$

Alla fine di questo Capitolo verrà proposta un'implementazione della classe C++ per il calcolo di questa forma d'onda.

1.4.8 Sistemi vibranti

Il moto ciclico generato da un sistema vibrante permette di percepire un suono. I sistemi vibranti possono essere ad esempio il pendolo, peso attaccato ad una molla, i moti di un atomo in uno stato solido, il moto di una corda di uno strumento musicale, una percussione ecc.. Per capire come da un sistema si possa generare un'onda sinusoidale consideriamo un sistema vibrante molla-massa. Facendo riferimento all Figura 1.18, supponiamo che un corpo di massa m si muove lungo l'asse delle x sotto l'azione della molla ideale di costante elastica k. In assenza di forze dissipative, la massa m si muoverà avanti ed indietro lungo x compiendo un moto oscillatorio periodico. Dalla seconda legge di Newton abbiamo che

$$-kx = ma. \tag{1.8}$$

Ricordiamo ora che l'accelerazione a è legata ad x (derivata seconda) ed esplicitamente otteniamo

$$\frac{d^2x}{dt^2} + \frac{k}{m}x = 0. \tag{1.9}$$

Questa risulta l'equazione rappresentativa del *moto armonico semplice* con coefficiente unitario per la derivata seconda ed omogenea. La soluzione dell'equazione differenziale risulta essere

$$x(t) = A\cos(\omega t + \phi_0) \tag{1.10}$$

dove A è l'ampiezza massima del moto oscillatorio, ϕ_0 è la fase iniziale e ω è la pulsazione pura

$$\omega = \sqrt{\frac{k}{m}} = 2\pi f \tag{1.11}$$

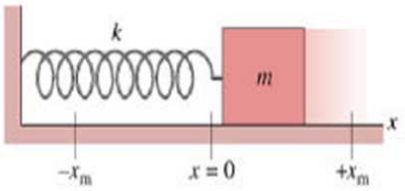


Figura 1.18: Sistema molla-massa.

La funzione trigonometrica coseno ha periodo 2π. Il moto si ripete dopo un tempo T (periodo)

$$T == \frac{2\pi}{\omega} = \frac{1}{f} = 2\pi\sqrt{\frac{m}{k}} \tag{1.12}$$

1.5 Caratteristiche del suono

Abbiamo osservato che il suono può essere definito come la percezione della compressione e della rarefazione dell'aria in prossimità dell'organo uditivo. I suoni possono essere sia prodotti naturalmente che artificialmente. Ci sono diverse proprietà relative ai suoni che derivano da effetti di propagazione e condizioni ambientali. Successivamente ci concentreremo su alcune di queste propietà.

1.5.1 Velocità di propagazione del suono

Il suono si propaga in un gas (ad esempio nell'aria) a una definita velocità

$$c = \sqrt{\frac{\gamma\rho_0}{\varrho}} \tag{1.13}$$

dove γ è un fattore di temperatura, ρ_0 è la densità del gas e ϱ è la pressione statica del gas. Ad esempio il suono si propaga nell'aria alla velocità di circa 344 m/s ed è soggetto al fenomeno della rifrazione. La rifrazione all'interno di un mezzo dipende dalle caratteristiche del mezzo stesso. Ogni mezzo ha una sua tipica velocità del suono a temperatura costante di 23.24^o C. In un mezzo riscaldato siccome alle sue particelle viene trasferita energia cinetica abbiamo una maggiore velocità del suono nel mezzo. Mediamente aumento (diminuzione) di velocità di 0.6 m/s per ogni incremento (decremento) di un grado C della temperatura del mezzo. La velocità del suono dipende, dalla pressione, dalla temperatura

e dalla densità del gas entro cui si propaga. Notiamo, infatti, che la compressione è legata al riscaldamento e la rarefazione al raffreddamento dell'aria. La velocità di propagazione implica intensità di distorsione della forma d'onda. Per le alte frequenze è più probabile una distorsione della forma d'onda.

1.5.2 Lunghezza d'onda

La *lunghezza d'onda* è la distanza che il suono percorre per completare un ciclo completo di compressione e rarefazione. Come abbiamo visto precedentemente essa è correlata con la frequenza e con la velocità di propagazione nel seguente modo

$$c = \lambda f \tag{1.14}$$

dove c è la velocità di propragazione, λ la lunghezza d'onda e f la frequenza.

1.5.3 Ampiezza sonora

L'*ampiezza sonora* è l'entità dello spostamento di una particella d'aria in un punto. Ci sono due tipi misure

- *Pressione sonora* (Sound Pressure level) - compressione e rarefazione delle particelle
- *Intensità sonora* (Sound Intensity Level) - energia trasportata dall'onda sonora

Per comprendere qual è l'ampiezza di pressione per i suoni si considerano il suono più debole (soglia di udibilità) e il suono più forte (soglia del dolore). Generalmente per "schiacciare" la scala di riferimento si usa la scala logaritmica del rapporto tra due suoni. La misura risultante è in decibel (db). In questo modo abbiamo che la potenza sonora e l'intensità sonora sono rispettivamente

$$P_{db} = 20 \log_{10} \frac{P_1}{P_0} \tag{1.15}$$

$$I_{db} = 10 \log_{10} \frac{I_1}{I_0} \tag{1.16}$$

$$\tag{1.17}$$

dove P_0 e I_0 sono la potenza e l'intensità di riferimento. Da quanto detto abbiamo che il raddoppio di una misura comporta +3 dB di potenza e +6 dB di intensità. Inoltre, 0 dB è l'intensità pari al riferimento. Si caratterizzano le seguenti scale

Suono	P_{db}	Reazione
Massimo rumore prodotto in laboratorio	210	suono insopportabile
Rottura del timpano	160	
Suono al limite del dolore	120	dolore fisico
Complesso rock al chiuso	110	
Urlo (a 3 m)	100	
Traffico cittadino	70-80	
Conversazione	50	
Zanzara (vicino all'orecchio)	10	
Soglia del'udito (a 1000 Hz)	0	Non udibile

Tabella 1.1: Potenze in *db* di alcuni suoni più comuni.

- *Amplificazione* - scala di ampiezza in dB prevalentemente positiva 0 dB ha il significato di nessuna amplificazione
- *Attenuazione* - è una scala di ampiezza in dB prevalentemente negativa 0 dB ha il significato di nessuna attenuazione
- *Equalizzazione* - è una scala in dB sia positiva che negativa 0 dB ha il significato di segnale non equalizzato

Nella tabella 1.5.3 vengono riportate le potenze in *db* di alcuni suoni comuni.

1.6 Effetti di propagazione

La frequenza è una caratteristica fondamentale di un suono. Essa è responsabile dell'altezza di un suono, ma non tutti i suoni hanno un'altezza definita. L'altezza, quindi, scaturisce dalla periodicità di un segnale. Le caratteristiche frequenziali ci fanno distinguere un suono puro da un suono complesso. Un suono puro (tono) è costituito da una sola frequenza ed è quindi descritto da un'onda sinusoidale semplice. Un suono complesso consiste invece dalla sovrapposizione di più frequenze che rendono l'andamento del suono più articolato. Nella Figura 1.19 viene rappresentato un tono puro con frequenza di 100 Hz e in Figura

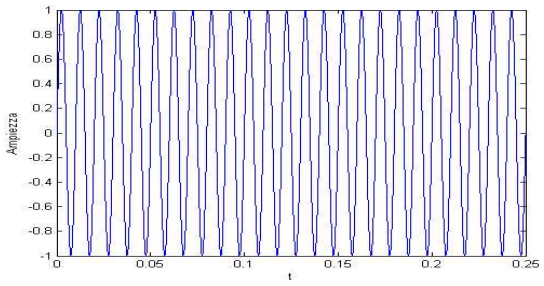


Figura 1.19: Un tono puro con frequenza di 100 Hz.

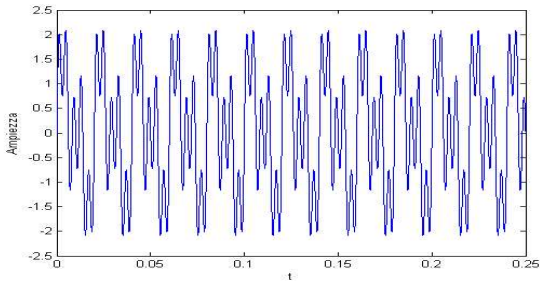


Figura 1.20: Un suono complesso con frequenze di 50, 100 e 250 Hz.

1.20 un suono complesso composto dalla sovrapposizione di tre frequenze che sono 50, 100 e 250 Hz. Il range di frequenze dei suoni udibili va da circa 16 Hz a 20 KHz. Gli *infrasuoni* hanno una frequenza inferiore ai 20 Hz. Gli *ultrasuoni* hanno una frequenza superiore ai 20 KHz.

Nella Tabella 1.6 sono riportati alcuni esempi di suoni e le relative frequenze. Ad esempio possiamo considerare la nota più bassa di un pianoforte che ha frequenza $27,5$ Hz oppure il limite dell'udito che può essere identificato dalle frequenze comprese tra 16000 e 20000 Hz.

1.7 La forma d'onda

Una delle caratteristiche che ci permettono di distinguere suoni a parità di frequenza e ampiezza è la *forma d'onda* (o *waveform*). La forma d'onda identifica l'origine del suono e descrive l'andamento delle compressioni e rarefazione. Due elementi contribuiscono alla

Suono	frequenza (Hz)
Nota più bassa del pianoforte	27.5
Do centrale del pianoforte	26.6
il La dopo il DO centrale	440
nota più alta di un pianoforte	4180
armonica superiore degli strumenti musicali	10000
limite dell'udito delle persone anziane	12000
limite dell'udito	16000-20000

Tabella 1.2: Esempi di suoni con frequenze relative.

ricchezza delle forme d'onda complesse (e quindi timbro)

- Le *componenti spettrali* nel dominio della frequenza
- I *transitori* nel dominio del tempo (tempo di inizio e estensione di una vibrazione)

I segnali audio trasportano informazioni che possono essere caratterizzate sia nel tempo che nel dominio della frequenza. La forma d'onda è la variazione di ampiezza nel tempo ed è la caratteristica del fenomeno acustico. Tale informazione varia istante per istante ed è presentata matematicamente come funzione continua del tempo $A(t)$ (generalmente indicata con $x(t)$). Vi sono varie misure sintetiche dell'ampiezza di un segnale audio intese a fornire una informazione finalizzata e sintetica come mostrate in Figura 1.21

- *Ampiezza Massima*
  ampiezza massima (positiva o negativa) raggiunta dal segnale audio durante il periodo di misura
- *Ampiezza picco-picco*
  escursione massima di ampiezza raggiunta dal segnale audio (massima dinamica) durante il periodo di misura;
- *Ampiezza Media*
  media temporale delle ampiezze istantanee del segnale (offset) durante il periodo di misura;

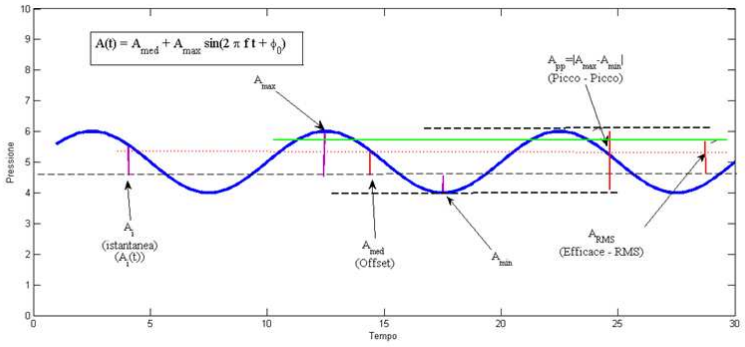


Figura 1.21: Ampiezza istantanea di un tono puro.

- *Ampiezza Efficace* (RMS)
  ampiezza effettiva, indice della potenza efficace del segnale durante il periodo di misura.

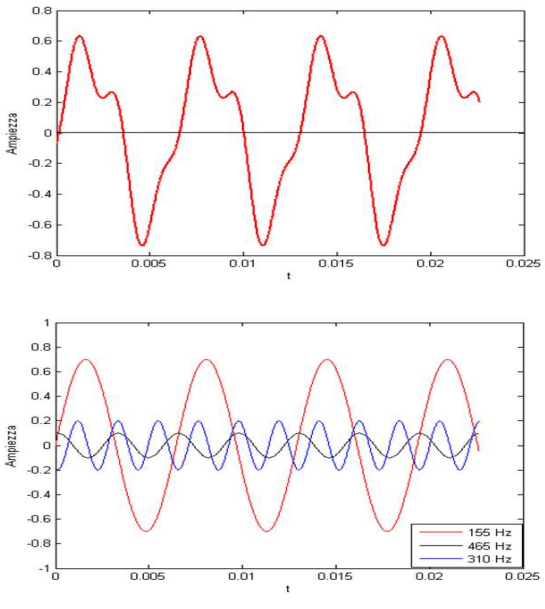
1.8 Dominio delle frequenze

Sebbene molto complessa, la forma d'onda può essere espressa come una sovrapposizione lineare di segnali elementari (toni puri sinusoidali). La distribuzione nello spazio delle frequenze delle componenti sinusoidali viene chiamato *spettro* e l'intervallo di esistenza è la *banda di frequenza*. La differenza tra la massima frequenza e la minima frequenza è la *larghezza di banda*. La larghezza di banda massima di percezione uditiva umana va da 16 a 20000 Hz. La natura fisica del segnale può ad esempio essere

- Segnali audio vocali (parlato) - larghezza di banda stretta (300 - 3000 Hz)
- Segnali audio musicali - larghezza di banda larga (16 - 20000 Hz)

1.8.1 Teorema di Fourier

Uno dei risultati fondamentali nel campo dell'elaborazione dei segnali audio è sicuramente il Teorema di Fourier. Esso fu formulato dal matematico francese Joseph Fourier agli inizi del 1800. Afferma che un segnale periodico qualsiasi è costituito dalla sovrapposizione di onde sinusoidali semplici, ciascuna con la sua ampiezza e fase, e le cui frequenze sono



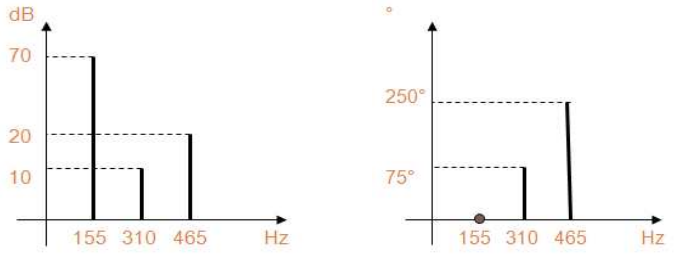
a)

b)

Figura 1.22: Segnale complesso e componenti armoniche.

armoniche della frequenza fondamentale del segnale. L'analisi di Fourier permette di studiare un suono nel dominio delle frequenze evidenziando aspetti del suono non riconducibili direttamente alla rappresentazione nel dominio del tempo.

In altre parole l'idea è quella che si può pensare ad un segnale periodico complesso come costituito da più segnali semplici. La banda di frequenze del segnale è costituita dalle frequenze dei segnali semplici. Il processo di individuazione delle componenti di frequenza di un segnale è denominato *analisi spettrale* o *armonica*. Le armoniche della frequenza fondamentale sono multipli interi della frequenza fondamentale. Consideriamo il seguente segnale complesso con le seguenti frequenze (Figura 1.22): fondamentale 155 Hz, la seconda armonica 310 Hz e terza armonica 465 Hz. Inoltre la seconda armonica ha un'ampiezza 7 volte inferiore alla fondamentale e una fase di 75 gradi. La terza ha un'ampiezza doppia della seconda armonica e una fase di 250 gradi. L'*analisi* di Fourier, quindi, dato un segnale complesso corrisponde all'individuazione dei segnali semplici che lo compongono (Figura 1.22b)). La *sintesi* di Fourier, invece, permette di ricomporre un suono complesso a partire da sinusoidi semplici. Lo spettro di Fourier è l'insieme delle componenti di un segnale, con la propria ampiezza e fase. Le armoniche vengono dette anche *componenti* di Fourier. Ciascuna componente ha una sua ampiezza e fase. Applicando l'analisi di



a) b)

Figura 1.23: Analisi di Fourier: a) diagramma delle ampiezze; b) diagramma delle fasi.

Fourier possiamo avere due diagrammi, uno per le ampiezze e uno per la fase. Come mostrato in Figura 1.23 i due diagrammi mostrano le caratteristiche del segnale complesso precedentemente considerato.

1.8.2 Trasformata di Fourier

La tecnica per ottenere le informazioni di un segnale audio nel dominio delle frequenze si chiama *Trasformata diretta di Fourier*. La *Trasformata inversa di Fourier*, invece, permette di effettuare il percorso opposto, dal dominio della frequenza al dominio del tempo. Dobbiamo comunque notare che i segnali in natura non sono strettamente periodici. Il teorema di Fourier si può estendere a segnali non periodici considerando una sovrapposizione di sinusoidi che non sono in un rapporto armonico. Lo spettro in questo caso si dice inarmonico. Il caso limite di inarmonicità è il *rumore bianco* (vedi Figura 1.24) chiamato in questo modo per il paragone con il fenomeno luminoso. Lo spettro di un rumore bianco contiene tutte le frequenze con la stessa ampiezza. Esistono anche altre tipi di rumore come ad esempio il *rumore rosa* (vedi Figura 1.25) che privilegia le basse frequenze.

1.9 I transitori

Sebbene i suoni musicali non presentano periodicità essi si distinguono dal rumore poichè la loro non periodicità può essere caratterizzata mediante delle funzioni. La funzione che descrive il comportamento del suono a un livello macroscopico è tipicamente quella dell'*inviluppo*.

Dato un segnale, il suo inviluppo è la curva che si ottiene congiungendo tutti i picchi della

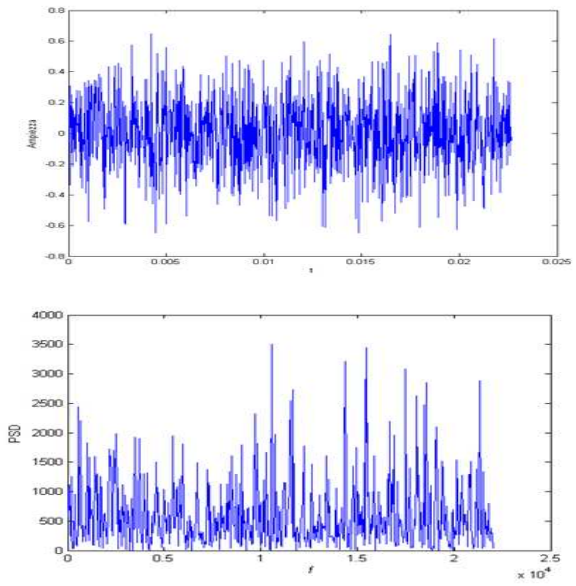


Figura 1.24: Rumore bianco e suo spettro.

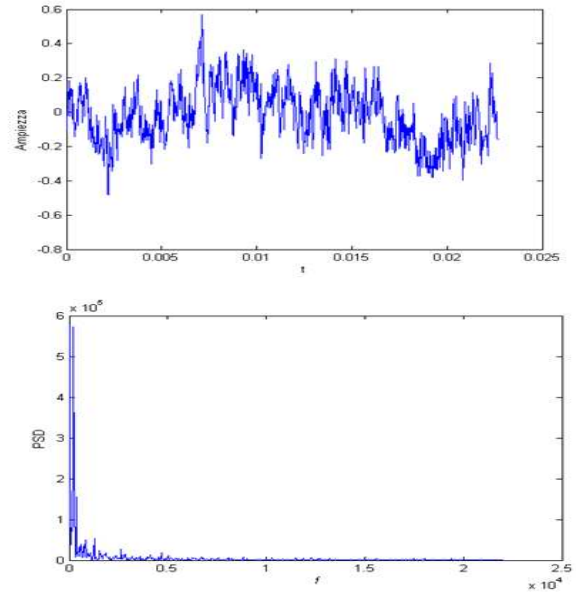


Figura 1.25: Rumore rosa e suo spettro.

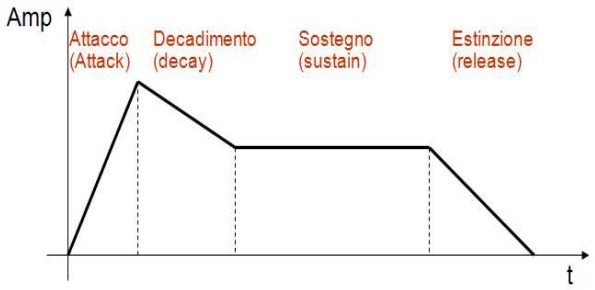


Figura 1.26: Fasi fondamentali in un inviluppo.

parte positiva. Per quanto riguarda l'inviluppo si possono individuare quattro transitori (vedi Figura 1.26)

- *Attacco* (attack) in cui l'ampiezza varia da zero alla massima ampiezza
- *Decadimento* (decay) in cui l'ampiezza diminuisce fino ad un certo livello
- *Sostegno* (sustain) in cui l'ampiezza rimane pressappoco costante
- *Estinzione* (release) in cui l'ampiezza diminuisce fino a zero.

In Figura 1.27 vengono proposti gli inviluppi caratteristici di alcuni strumenti musicali: flauto, tromba, pianoforte, violino, blocchi di legno, contrabasso. Dobbiamo notare che i suoni percussivi come il pianoforte o i blocchi di legno hanno degli inviluppi con dei transitori molto repentini con l'assenza di una fase di sostegno. I suoni derivanti da vibrazioni di colonne d'aria (flauto e tromba) o corde (violino e contrabasso) hanno dei transitori meno repentini. L'attacco della tromba, invece, presenta due massimi caratteristici. L'inviluppo della chitarra segue in modo più morbido l'inviluppo del pianoforte.

1.10 Periodogramma e Spettrogramma

Una diretta conseguenza della trasformata di Fourier è la *stima spettrale*. E' il meccanismo che permette di studiare il conenuto frequenziale di un segnale audio discreto. Come vedremo nei prossimi Capitoli molte informazioni di un segnale audio sono descritte in maniera univoca nel dominio delle frequenze.
Come esempio consideriamo un segnale audio corrrispondente ad un pezzo di un brano musicale (Paolo Fresu Quintet). In Figura 1.28 è stato rappresentato il segnale audio nel

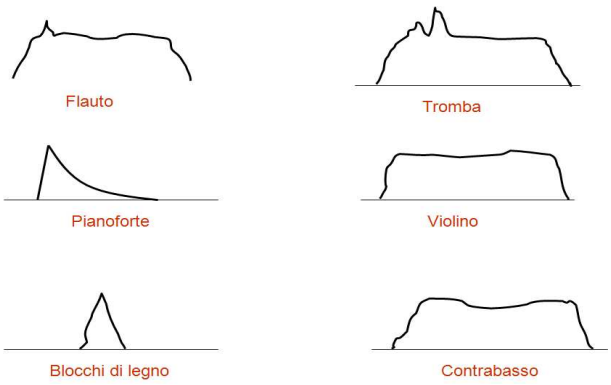


Figura 1.27: Inviluppo di strumenti musicali.

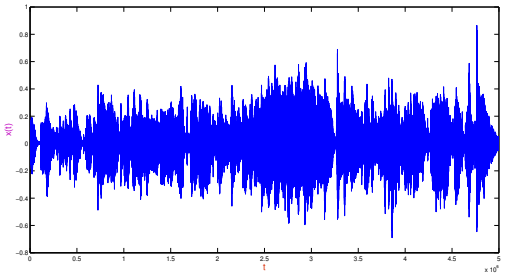


Figura 1.28: Brano musicale (Paolo Fresu Quintet).

dominio del tempo. In Figura 1.29 viene rappresentata l'analisi spettrale di questo segnale audio. Il segnale è caratterizzato da una propria **banda di frequenza** che rappresenta tutte le frequenze contenute dal segnale nel range identificato dalla frequenza di campionamento. Il concetto di banda di frequenza risulterà di fondamentale rilievo nel seguito del libro.

Comunque, quando si vuole realizzare un'analisi di Fourier di un suono musicale, occorre considerare l'evoluzione dello spettro nel tempo. In questo caso parliamo della tecnica dello *spettrogramma*. Questa tecnica permette di avere un grafico tridimensionale che rappresenta l'evoluzione nel tempo, nella frequenza e nell'ampiezza del segnale. Esso viene realizzato mediante una variante della trasformata di Fourier chiamata *Short Time Fourier Transform* (vedi Figura 1.30).

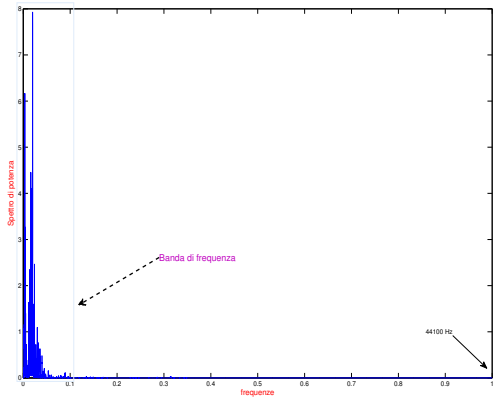


Figura 1.29: Stima spettrale del brano musicale di Figura 1.28.

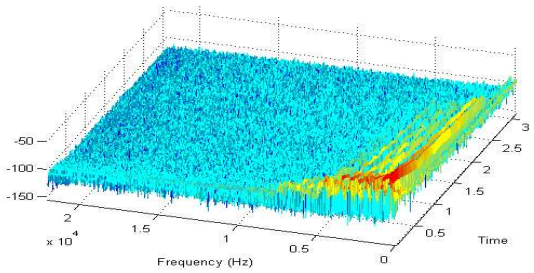


Figura 1.30: Spettrogramma.

	sequenze			
#	t	:	vector < double >	informazione temporale
#	x	:	vector < Type >	ampiezze(x(t))
#	size	:	int	lunghezza sequenza
+	sequenze (int)			costruttore
+	sequenze (double,double,double)			costruttore
+	sequenze (vector <Type>)			costruttore
+	∼sequenze (double,double,double)			distruttore
+	gett()	:	vector <double>	informazione temporale
+	gett(int)	:	double	informazione temporale
+	getx()	:	vector <Type>	informazione ampiezza
+	getx(int)	:	Type	informazione ampiezza
+	energia()	:	double	energia
+	clone(sequenze ∗)	:	sequenze&	metodo clone
+	somma(sequenze ∗)	:	sequenze &	somma di sequenze
+	sottrazione(sequenze ∗)	:	sequenze&	sottrazione di sequenze
+	prodotto(sequenze ∗)	:	sequenze&	prodotto di sequenze
+	sprodotto(sequenze ∗)	:	double	prodotto scalare
+	ritardo(int)	:	sequenze&	delay di sequenze
+	delta(double)	:	int	delta di Dirac
+	rappimpulsi()	:	sequenze&	rappresentazione ad impulsi
+	conv(vector <double>)	:	sequenze&	convoluzione
+	eqdiff(vector<double> , int , vector<double>, int)	:	sequenze&	equzioni alle differenze
+	length()	:	int	lunghezza della sequenza

Tabella 1.3: Classe **sequenze**

1.11 Tono puro e C++

Per l'applicazione, a casi reali, dei concetti teorici trattati, con il libro viene distribuita una Libreria in C++ chiamata ESA-DSP. Nel corso del libro faremo riferimento a questa libreria per realizzare i concetti che vengono via via introdotti. Si consiglia di consultare il codice della libreria per un ulteriore approfondimento.

La classe base da cui verranno ereditate le successive classi è chiamata **sequenze**. Il suo contenuto è specificato tramite la classe della Tabella 9.16.2 secondo lo schema **Class Diagram** del linguaggio UML (Unified Modeling Language).

Il codice di riferimento nell libreria ESA-DSP è il seguente

```
#ifndef SEQUENZE_H_
#define SEQUENZE_H_
// Classe base: sequenze
template <class Type> class sequenze {
public:
    // Costruttori e distruttori
```

```
    sequenze(int);
    sequenze(double, double, double);
    sequenze(vector<Type>);
    ~sequenze();
    // Metodi per il recupero dei dati
    vector<double> get_t();
    vector<Type> get_x();
    double get_t(int);
    Type get_x(int);
    // Metodo per calcolare l'energia di una sequenza
    Type energia();
    // Metodo Clone per copiare due oggetti sequenza
    sequenze& clone(sequenze *s);
    // Metodi per per operazioni sulle sequenze
    sequenze& somma(sequenze *s);
    sequenze& sottrazione(sequenze *s);
    sequenze& prodotto(sequenze *s);
    sequenze& s_prodotto(double);
    sequenze& ritardo(int);
    // Metodo per la rappresentazione tramite impulsi
    double delta(int);
    sequenze& rapp_impulsi();
    // Metodo per la convoluzione
    sequenze& conv(vector<double> );
    // Metodo per il calcolo delle equazioni alle differenze
    sequenze&  eq_diff(vector<double> , int , vector<double>, int);
    // Metodo per ottenere la lunghezza della sequenza
    int length();
protected:
  // Campi
  // array temporale, array delle ampiezze e lunghezza sequenza
  vector<double> t;
  vector<Type> x;
  int size;
};
#endif //SEQUENZE_H
```

La classe usata per creare una sequenza discreta sinusoidale (tono puro) invece risulta un'estensione della classe precedente. Il suono nome è *sinusoidale*. La Tabella 5.5.4

sinusoidale				
#	name	:	string	nome della sequenza
+	sinusoidale (int, double, double, double)			costruttore
+	sinusoidale (double,double,double)			costruttore
+	∼sinusoidale ()			distruttore
+	sinwt(double, double, double, double)	:	double	sinusoide
+	getname()	:	string	nome

Tabella 1.4: Classe **sinusoidale**.

descrive questa classe e il codice C++ è il seguente

```
#ifndef SINUSOIDALE_H_ #define SINUSOIDALE_H_ template <class Type>
class sinusoidale : public  sequenze<Type>{
public:
    // Costruttori e distruttori
    sinusoidale(int n, double A, double f, double phase);
    sinusoidale(double inf, double sup, double step,
    double A, double f, double phase);
    ~sinusoidale() { }
    // Funzione sinusoidale
    double sin_wt(double A, double f, double phase, double t_n) {
    return A * sin(2*PI*f*t_n + phase);
    }
    // Metodo per ottenere il nome dell'oggetto
    string get_name() {
    return name;
    }
private:
    string name;
};
#endif // SINUSOIDALE_H_
```

Capitolo 2

Rappresentazione digitale del suono

La maggior parte dei moderni sistemi di elaborazione, memorizzazione e trasmissione dei segnali audio sono di tipo digitale. In altre parole i dati in ingresso a questi dispositivi sono di tipo numerico. In altri casi, comunque, la rappresentazione numerica dei segnali originari non è direttamente digitale e quindi richiede un'ulteriore elaborazione. Infatti, molti segnali con cui abbiamo a che fare nella realtà quotidiana sono continui sia nel tempo che nelle ampiezze. Il suono analogico, infatti, pervade ancora la maggior parte dell'audio che ascoltiamo: radio, televisione, ecc.. La rappresentazione del segnale audio sotto forma numerica è utile poichè in questo modo un segnale è adatto ad essere memorizzato, adatto ad essere elaborato e trasmesso tramite sistemi di natura digitale. In molti sistemi di elaborazione per poter trasformare un segnale da analogico a digitale, come ad esempio per un microfono connesso ad un computer, abbiamo bisogno di un dispositivo per convertitore il segnale da Analogioco-Digitale (A/D). Inoltre per il passaggio inverso, come ad esempio per le casse connesse ad un computer, abbiamo bisogno di un convertitore Digitale-Analogico (D/A). Queste due fasi sono cruciali e non possono essere fatte in modo casuale. Bisogna infatti rispettare il Teorema di Campionamento per non avere problemi nella ricostruzione del segnale, come vedremo successivamente. Inoltre per memorizzare la sequenza numerica, e cioè il segnale digitale su un computer, abbiamo bisogno di definire un numero di bit per la rappresentazione. Questa fase comprende la quantizzazione e la codifica. Prima di introdurre le fasi di campionamento e di quantizzazione vediamo come funzionano due tipici trasduttori e cioè il microfono e l'altoparlante.

2.1 Trasduzione

La trasduzione è un meccanismo per trasformare un segnale fisico in un segnale elettrico e viceversa. La trasduzione da natura fisica qualsiasi a natura elettrica si ottiene con i cosiddetti *sensori*. Per trasformare un segnale audio (natura meccanica) in segnale elettrico, si usa il *microfono*. La trasduzione da natura elettrica a natura fisica qualsiasi si ottiene con gli *attuatori*. Per trasformare il segnale elettrico in segnale acustico si usa l'altoparlante (attuatore elettro-meccanico).

2.1.1 Il microfono

Il *microfono* è il principale trasduttore per la trasformazione del suono dalla sua natura meccanica a quella elettrica. Nel principio di trasduzione microfonica la variazione di pressione dell'aria prodotta dal suono (compressione e rarefazione) viene utilizzata come mezzo per agire su un dispositivo o componente dotato di proprietà meccano-elettra (vedi Figura 2.1). Con il principio di induzione elettromagnetica la pressione acustica viene utilizzata per far muovere un magnete all'interno di un campo magnetico. La variazione di campo magnetico segue quella dell'onda acustica del suono. Tale variazione induce su un filo di rame una corrispondente variazione di corrente. Tale corrente variabile è una copia trasdotta della pressione variabile del suono e cioè la copia elettronica del suono. La trasduzione crea un'analogia tra la rappresentazione acustica del suono e la rappresentazione elettrica. Per questo motivo il suono, nella sua natura fisica, viene classificato come *analogico*. Il termine analogico ha un sinonimo che è il termine *continuo*.

2.1.2 L'altoparlante

Il suono in forma elettrica è utile per essere trattato dalle apparecchiature elettroniche ma non è percepibile dall'orecchio. La trasduzione di natura elettro-meccanica consente di trasformare le variazioni di tensione elettrica in variazioni di pressione acustica (vedi Figura 2.2). L'altoparlante ha un principio di funzionamento simile a quello del microfono. Una corrente variabile applicata ad un filo elettrico induce un campo magnetico identicamente variabile nell'intorno del filo stesso. Il campo magnetico è in grado di opporsi con forza ad un altro campo magnetico opposto. Al magnete sottoposto al campo variabile si applica

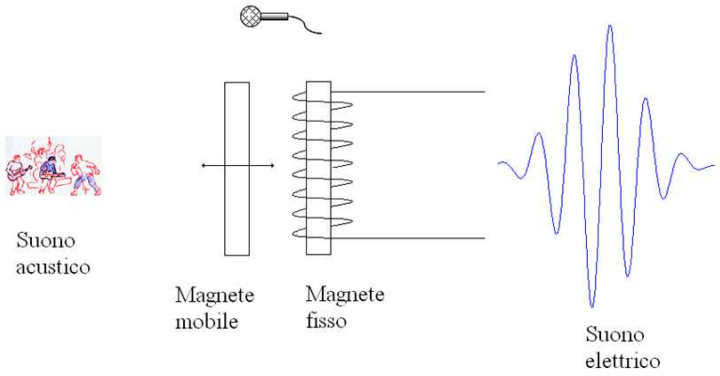


Figura 2.1: Principio di funzionamento di un microfono.

una membrana (cono) che trasforma la variazione di campo magnetico in variazione di pressione acustica equivalente.

2.2 Audio analogico e digitale

Per capire la differenza tra i sistemi analogici e quelli digitali vediamo diversi componenti che possono essere impiegati nella produzione e nella fruizione di audio digitale.
Prima di tutto un microfono cattura le variazioni di pressione nell'aria e le trasduce in un segnale elettrico. Il segnale elettrico in uscita passa attraverso un pre-amplificatore e un amplificatore prima di essere registrato in generale su un nastro magnetico. A questo punto il segnale può essere trasferito su uno qualsiasi dei supporti del suono analogico: nastro magnetico, vinile, pellicola cinematografica. Da questi supporti il segnale può essere prelevato (o meglio trasdotto) mediante un lettore adeguato (giradischi). Il segnale viene amplificato e mandato agli altoparlanti che trasducono il segnale elettrico in un segnale sonoro.
Per stimare l'ammontare di rumore introdotto da un sistema analogico si usa il rapporto segnale rumore (Signal to Noise Ratio, SNR). Il rapporto SNR si definisce come il rapporto tra la massima ampiezza utile del segnale e l'ampiezza del rumore presente

$$\text{SNR} = \frac{\text{max ampiezza segnale}}{\text{ampiezza rumore}} \tag{2.1}$$

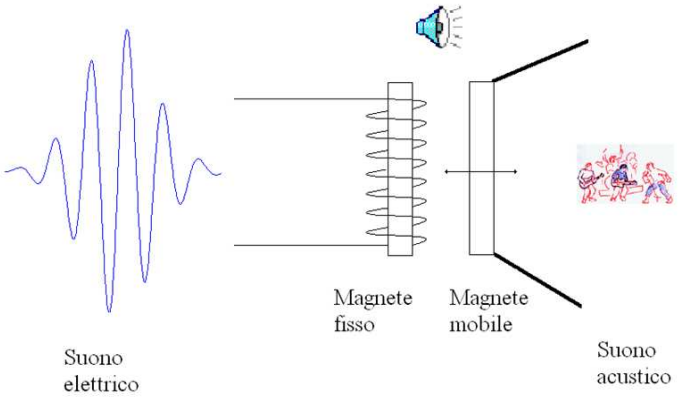


Figura 2.2: Principio di funzionamento di un altoparlante.

Maggiore è il rapporto migliore sarà la qualità del segnale. Generalmente viene calcolato in db

$$\mathrm{SNR}_{\mathrm{db}} = 20 \log(\mathrm{SNR}) \tag{2.2}$$

Il *Dynamic Range* misura la bontà di un dispositivo analogico ed è la differenza in db tra le ampiezze massima e minima del segnale utile.
Nel caso di sistemi digitali, invece, bisogna convertire il segnale analogico del microfono in segnale digitale (convertitore A/D). Prima di digitalizzare il segnale occorre filtrarlo in modo da eliminare una parte del suo spettro che potrebbe causare il problema dell'aliasing (secondo il teorema di Nyquist). Dall'altra parte il segnale viene convertito da digitale ad analogico (D/A). Il D/A genera le tensioni elettriche che viene interpolato con un filtro passa-basso. Infine il segnale viene amplificato e mandato ai diffusori per la trasduzione in segnale acustico.
Possiamo quindi affermare che per la digitalizzazione sono fondamentali due operazioni di discretizzazione

- discretizzazione del tempo - *campionamento*

- discretizzazione dell'ampiezza - *quantizzazione*

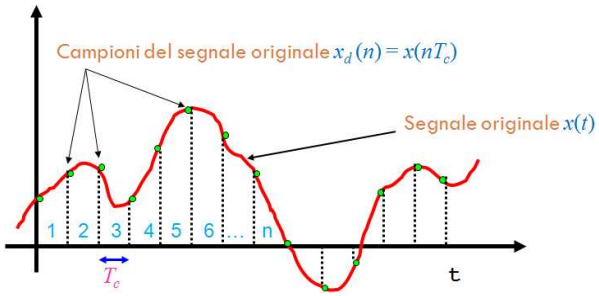


Figura 2.3: Segnale continuo $x(t)$.

2.3 Il campionamento

Focalizziamo ora la nostra attenzione sulla prima fase di discretizzazione e cioè il campionamento. Sostanzialmente occorre discretizzare il tempo e cioè scegliere una frequenza di campionamento f_c. Purtroppo non possiamo scegliere una frequenza di campionamento qualsiasi ma dobbiamo soddisfare alcune condizioni. Queste condizioni sono dettate dal Teorema di Campionamento.

In Figura 2.3 viene rappresentato un segnale continuo variabile nel tempo che denominiamo $x(t)$. Sull'asse delle ascisse è indicato il tempo continuo t e sulle ordinate il valore dell'ampiezza del segnale e cioè $x(t)$. A questo punto fissiamo un periodo di campionamento T_c. In base a questo periodo di campionamento i valori del segnale vengono letti a intervalli regolari. L'n-esimo campione della sequenza discreta $x_d(.)$ è ottenuto, quindi, secondo la relazione

$$x_d(n) = x(nT_c) \tag{2.3}$$

e cioè $x_d(n)$ corrisponde al campione preso al tempo nT_c. Ad esempio $x_d(0)$ corrisponde al campione a $0 \cdot T_c = 0$, $x_d(1)$ è il campione preso a $1 \cdot T_c = T_c$ e cosi via. La frequenza di campionamento risulta il reciproco del periodo di campionamento

$$f_c = \frac{1}{T_c} \tag{2.4}$$

Per ottenere una sequenza discreta viene usato un convertitore da tempo continuo a tempo discreto (Convertitore C/D). Lo schema di un convertitore è raffigurato in Figura 2.4. Il convertitore preso un segnale continuo in input e un periodo di campionamento T_c restituisce la sequenza discreta $x_d(n)$. In realtà l'operazione di campionamento è spesso implementata con un convertitore Analogico-Digitale (A/D). Esso è un'approssimazione

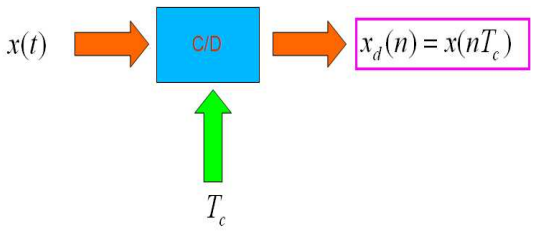


Figura 2.4: Convertitore C/D.

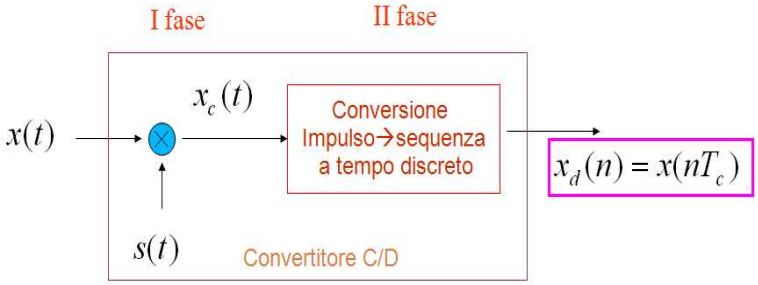


Figura 2.5: Fasi del convertitore C/D.

del C/D ideale in quanto viene considerata la quantizzazione, linearità, circuiti di memorizzazione e limitazioni sulla frequenza di campionamento.

In generale è conveniente rappresentare il processo di campionamento come suddiviso in due fasi (vedi Figura 2.5). La prima fase è inerente alla costruzione di una risposta impulsiva $x_c(t)$ e la seconda fase è relativa alla conversione da impulso a sequenza a tempo discreto $x_d(n)$. Nella prima fase viene usata una tecnica di modulazione ad impulsi e nella seconda una semplice conversione impulso-sequenza. Per chiarire il ruolo del periodo di campionamento e l'associazione tra sequenza discreta e segnale continuo proponiamo il seguente esperimento. Consideriamo un segnale $x_c(t)$ campionato ad una certa frequenza di campionamento f_c. Dal campionamento otteniamo il segnale come in Figura 2.6a. In Figura 2.6b, invece, consideriamo un periodo di campionamento doppio. Nelle Figure 2.6c e 2.6d vengono visualizzate le sequenze ottenute dopo la conversione dei segnali campionati rappresentati nelle Figure 2.6a e b, rispettivamente. Sostanzialmente tra le due sequenze c'è una differenza di normalizzazione rispetto al periodo T_c.

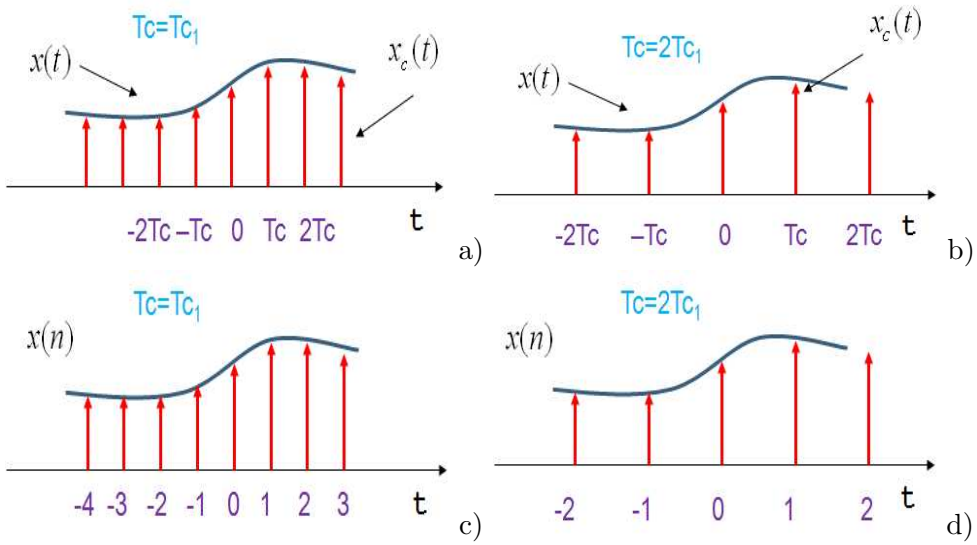


Figura 2.6: Esempio di campionamento.

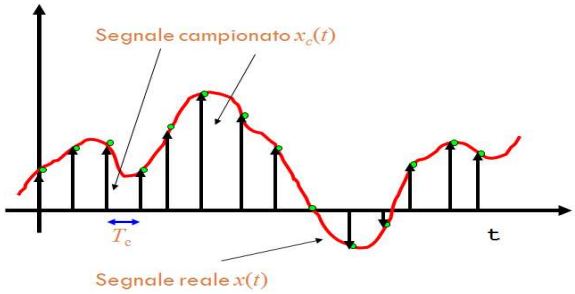


Figura 2.7: Segnale campionato $x_c(t)$.

2.3.1 Pulse Code Modulation

Per ottenere una sequenza discreta $x_c(t)$ da un segnale continuo $x(t)$ si usa la **modulazione ad impulsi** (o **Pulse Code Modulation**) (vedi Figura 2.7). Dal punto di vista matematico il campionamento è il prodotto tra il segnale continuo $x(t)$ da campionare e una funzione di campionamento $s(t)$ rappresentata da una sequenza periodica di impulsi $\delta(t)$ (vedi Figura 2.8).

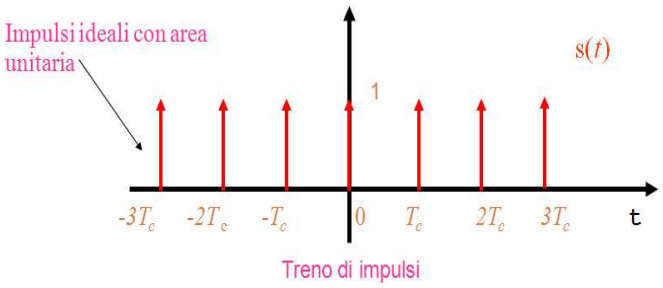


Figura 2.8: Treno di impulsi $s(t)$.

2.3.2 Teorema di campionamento

Il risultato che vogliamo raggiungere è quello di stabilire la frequenza minima di campionamento di un segnale, necessaria per evitare distorsioni dello stesso nella fase di ricostruzione. Questo risultato prende il nome di Teorema di Campionamento (o di Nyquist). Fu introdotto da Harold Nyquist e comparso nel 1949 in un articolo di C.E. Shannon. Esso afferma che un segnale a tempo continuo $x(t)$ con una banda spettrale B strettamente limitata, può essere univocamente ricostruito dalla sua versione campionata $x(n) = x(nT_c)$ $(n = 0, \pm 1, \pm 2, \pm 3, \ldots)$ se la frequenza di campionamento $f_c = 1/T_c$ soddisfa la seguente relazione

$$f_c = \frac{1}{T_c} \geq 2B \tag{2.5}$$

La frequenza di Nyquist è definita come

$$f_N = \frac{f_c}{2} \tag{2.6}$$

Nel Capitolo 4 "dimostreremo" graficamente il Teorema di Campionamento facendo uso della Trasformata di Fourier. Successivamente ci concentreremo sulle problematiche che possono scaturire da una errata scelta della frequenza di campionamento.

2.3.3 Aliasing

L'*aliasing* consiste in un effetto di rallentamento delle frequenze presenti oltre la frequenza massima stabilita dal teorema di campionamento. E' una conseguenza del sottocampionamento delle componenti armoniche del segnale che superano la metà della frequenza di campionamento. La frequenza alias è pari alla differenza tra il valore della frequenza

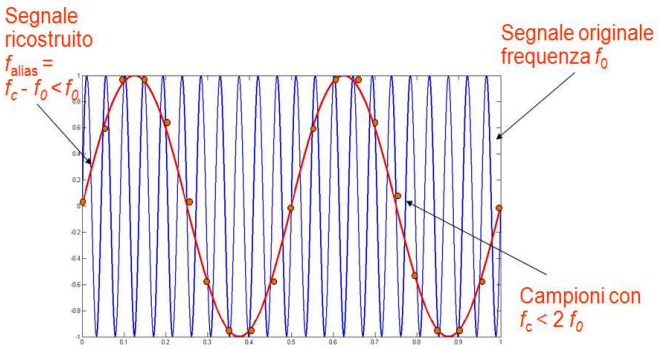


Figura 2.9: Effetto di aliasing su onde sinusoidali.

di campionamento e il valore della frequenza reale. Come esempio (vedi Figura 2.9), consideriamo una sinusoide con frequenza $f_0 = 22$ e frequenza di campionamento $f_c = 20$ possiamo notare come il segnale ricostruito non riproduce la frequenza originale ma quella di aliasing.

Un esempio di aliasing sono i *battimenti*. In questo caso la frequenza spuria è molto vicina a una frequenza già presente nel segnale. Un ulteriore esempio è il *glissando*, dove se la frequenza al di sotto della metà del tasso di campionamento, il segnale digitale rappresenta correttamente il segnale analogico, mentre se la frequenza del segnale supera la metà del tasso di campionamento, il segnale ricostruito diminuisce la sua frequenza. I segnali reali sono a banda infinita. La voce è un segnale audio banda fino a 20000 Hz. L'informazione necessaria è quealla fino a 3000 Hz e quindi campionare a 8000 Hz è sufficiente.

2.4 Quantizzazione

La seconda fase della digitalizzazione è la quantizzazione. La quantizzazione permette di trasformare valori numerici in sequenze di bit memorizzabili e utilizzabili da sistemi digitali. Dobbiamo comunque notare che utilizzando un numero prefissato di bit inevitabilmente si commette un errore (errore di quantizzazione). Successivamente introdurremo e confronteremo due tipi di quantizzazione, quella uniforme e quella non uniforme.

Figura 2.10: Dispositivo quantizzatore.

2.4.1 Quantizzazione uniforme

Nella fase di quantizzazione lo scopo principale è quello di trasformare una sequenza campionata e discreta $x_d(n)$ in una sequenza di bit memorizzabili su apparecchi digitali. Per introdurre il concetto di quantizzazione facciamo riferimento ad un dispositivo che chiameremo *quantizzatore* (vedi Figura 2.10). Esso trasforma il campione reale $x_d(n) = x(nT_c)$ nel campione quantizzato $x_q(n)$ mediante l'utilizzo di un numero prefissato di livelli. Vediamo ora come sono definiti i livelli. Abbiamo che ogni campione $x_d(n)$ è un numero reale che può assumere con continuità qualsiasi valore compresso in un certo intervallo di ampiezze che chiamiamo $[V_{\min}, V_{\max}]$. Per rappresentare il segnale discreto in forma numerica si stabiliscono K livelli uniformemente distribuiti nell'intervallo $[V_{\min}, V_{\max}]$ (vedi Figura 2.12). Essendo la quantizzazione uniforme, dato l'intervallo delle ampiezze e il numero di livelli possiamo, definire l'ampiezza Δ tra due intervalli consecutivi

$$\Delta = \frac{|V_{\min}| + |V_{\max}|}{K}. \tag{2.7}$$

Come è mostrato nella Figura 2.12, l'obiettivo principale della quantizzazione è quello di approssimare ogni singolo campione $x_d(n)$ (cerchio verde) con il campione $x_q(n)$ (stella rossa) a lui più vicino.

Nel processo di quantizzazione si commette un errore tanto più piccolo quanto più elevato è il numero K di livelli. L'errore di quantizzazione viene definito come la differenza tra il segnale quantizzato e quello campionato. In dettaglio abbiamo che l'errore $e(n)$ di

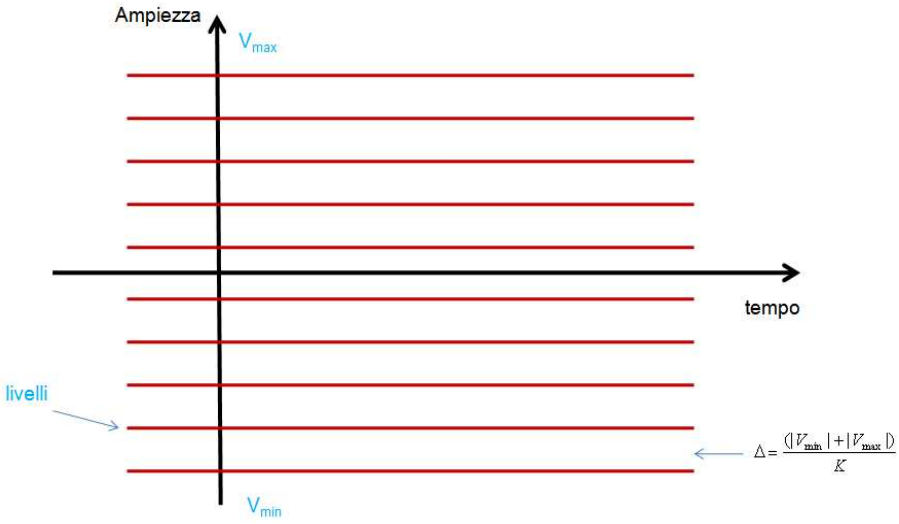


Figura 2.11: Livelli di quantizzazione.

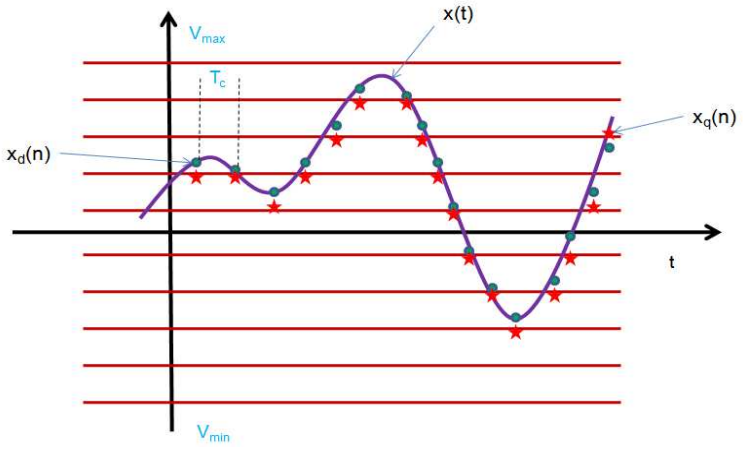


Figura 2.12: Quantizzazione di un segnale discreto.

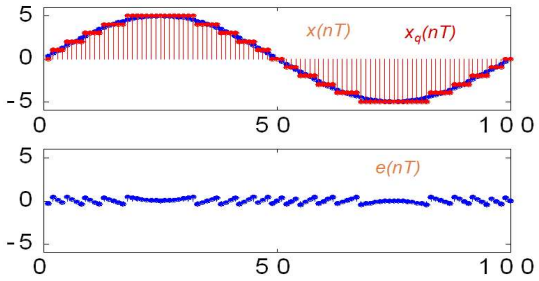


Figura 2.13: Quantizzazione ed errore di quantizzazione.

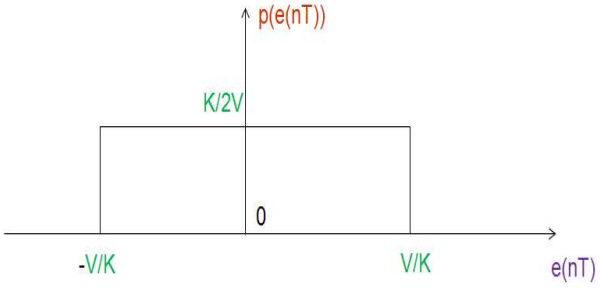


Figura 2.14: Distribuzione dell'errore di quantizzazione.

quantizzazione risulta

$$e(n) = x_q(n) - x(n). \tag{2.8}$$

In Figura 2.13 viene mostrata la fase di quantizzazione di una una funzione sinusoidale. Il segnale campionato è visualizzato in blue e il segnale quantizzato in rosso. Inoltre viene mostrato l'errore di quantizzazione.

L'errore di quantizzazione ha una distribuzione uniforme come visualizzato in Figura 2.14. Le caratteristiche sono il valore medio della distribuzione è nullo, la densità di probabilità è uniforme e quindi risulta $p(e) = K/2V$, valore quadratico medio (valore efficace) pari a $s = V/K3^{-1/2}$ e i campioni sono non correlati (se K sufficientemente grande).

2.4.2 Codifica

La fase della codifica corrisponde alla definizione del numero di bit che servono per la rappresentazione dei K livelli definiti nella fase di quantizzazione. Abbiamo quindi che con N cifre binarie (bit) si ottengono $K = 2^N$ livelli di quantizzazione. Ad ogni livello si

Livello	bit
L_1	000
L_2	001
L_3	010
L_4	011
L_5	100
L_6	101
L_7	110
L_8	111

Tabella 2.1: Codifica di 8 livelli usando 3 bit.

può dunque associare un codice di N bit (vedi la Tabella 2.4.2 per un esempio).
Inoltre abbiamo che se il numero di bit è almeno 6 o 7 il Rapporto-Segnale-Rumore SNR corrisponde a $SNR = 2^N$. In decibel ottentiamo $SNR = 20 \log_{10} 2^N$ e quindi $SNR = N \cdot 20 \log_{10} 2 = 6.02 \cdot N$ dB. Cioè abbiamo che ogni bit contribuisce con circa 6 decibel di rumore. Ad esempio per $N = 16$ bit $SNR = 96$ dB.

2.4.3 Bit rate

Uno dei concetti più usati nel caso di trasmissione o compressione di un file sonoro è quello di *bit rate*. Il bit rate è espresso come la cadenza di bit al secondo di un segnale numerico. Esso corrisponde al prodotto tra la frequenza di campionamento e il numero di bit di quantizzazione

$$\text{bit rate} = f_c \cdot N. \tag{2.9}$$

Ad esempio per un segnale telefonico con frequenza massima di 3.6KHz il teorema del campionamento ne impone una frequenza di campionamento f_c maggiore di 7.2KHz. Se utilizziamo quindi $f_c = 8$ KHz abbiamo 8000 campioni al secondo. Se quantizziamo il segnale con $K = 256$ livelli servono $N = 8$ bit. Il segnale telefonico numerico avrà, dunque, una bit rate di 64 Kbit per secondo.

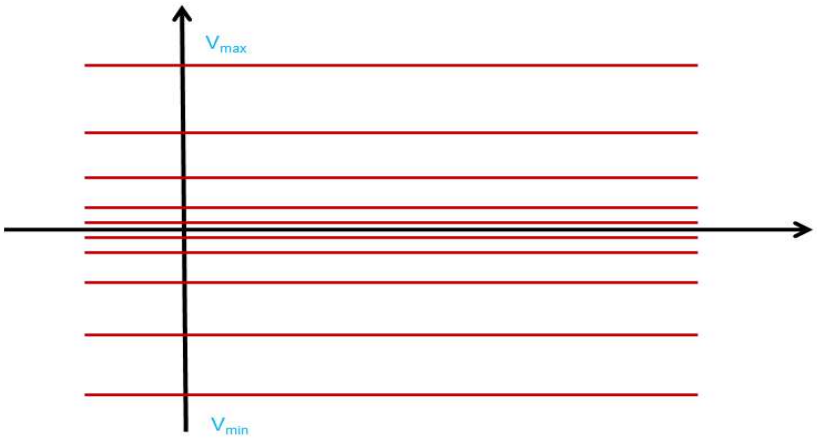


Figura 2.15: Quantizzazione non uniforme.

2.4.4 Quantizzazione non uniforme

La quantizzazione uniforme è molto semplice da implementare ma risulta poco efficace in alcuni casi reali. Uno svantaggio è quello di richiedere un numero elevato di livelli per ottenere un buon livello di quantizzazione. Per questo motivo spesso si preferisce la quantizzazione non uniforme (come ad esempio negli algoritmi di compressione). L'idea principale è quella di ridurre il rumore di quantizzazione alle ampiezze deboli ammettendo che aumenti di errore alle ampiezze forti. Il risultato viene raggiunto con una spaziatura non uniforme delle regioni di quantizzazione (vedi Figura 2.15).
Una delle quantizzazioni non uniformi maggiormente usate, nei comuni formati audio, è quella logaritmica. In questo caso la dimensione delle regioni di quantizzazione cresce con l'ampiezza del segnale secondo una curva logaritmica. Possiamo notare che assegnare le regioni di quantizzazione in modo uniforme rispetto ad una scala logaritmica porta alla non uniformità sulla scala lineare (vedi Figura 2.16). La quantizzazione non essendo uniforme ci consente di ottenere un errore di quantizzazione distribuito diversamente nelle varie regioni. Ad esempio le regioni intorno all'origine avranno un errore di quantizzazione minore rispetto alle altre.
Facciamo ora un confronto tra quantizzazione lineare e logaritmica (Figure 2.17 e 2.18 rispettivamente). Dai grafici possiamo notare l'effetto della quantizzazione tra le due diverse tecniche (linea arancione) e quindi del corrispondente errore di quantizzazione. Sebbene la quantizzazione uniforme usa 4 bit per la codifica e quella logaritmica usa solo

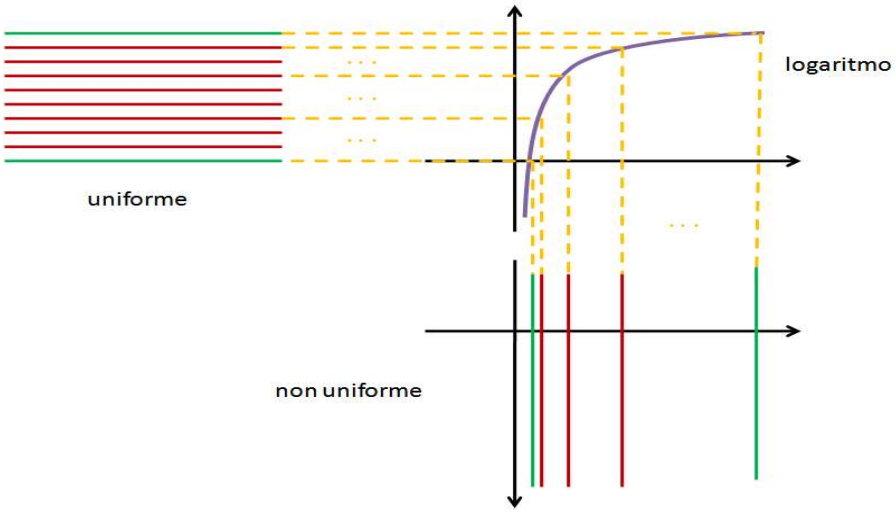


Figura 2.16: Trasformazione logaritmica.

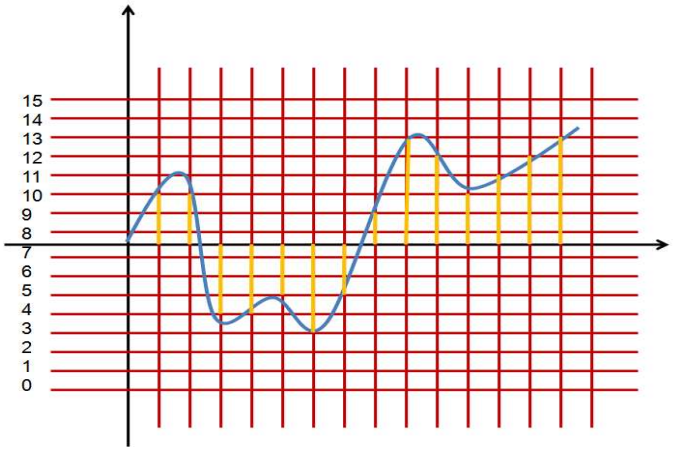


Figura 2.17: Quantizzazione uniforme.

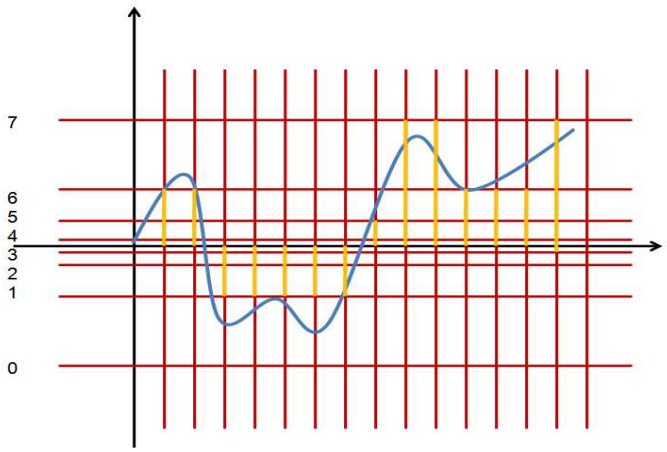


Figura 2.18: Quantizzazione logaritmica.

3 bit, l'errore non è molto differente.
In sostanza abbiamo che nei casi reali si preferisce la quantizzazione logaritmica siccome si nota che la qualità di quantizzazione logaritmica con 8 bit ha la stessa gamma dinamica della un quantizzatore lineare a $13 - 14$ bit. Per quanto riguarda il rapporto segnale rumore un convertitore 8-bit logaritmico va meglio di un convertitore 8-bit lineare alle ampiezze basse, ma peggio alle ampiezze elevate. In generale si ha che la quantizzazione logaritmica ha una migliore qualità audio a parità di frequenza di campionamento ma risulta più complessa nell'applicazione alle tecniche di elaborazione del segnale.

2.5 Note e accordi in C++

Nel Capitolo precedente, partendo dalla classe base **sequenze**, abbiamo visto la classe **sinusoidale** che ci permetteva di generare un tono puro. A questo punto, introducendo la frequenza di campionamento in un tono puro possiamo creare delle note musicali. La classe **makeWave**, estensione della classe **sequenze**, infatti prevede l'utilizzo della frequenza di campionamento in un tono puro. Si può fare riferimento alla libreria ESA-DSP per gli esempi sulla creazione di note e accordi facendo uso di oggetti istanziati da questa classe. La classe è descritta nella Tabella 9.16.2 e il codice C++ è il seguente

```
#ifndef MAKEWAVE_H_
#define MAKEWAVE_H_
```

	makeWave			
#	name	:	string	nome della sequenza
+	makeWave (int, double, vector < double > &)			costruttore
+	∼makeWave ()			distruttore
+	wave(int, double, vector < double > &)	:	void	nota musicale

Tabella 2.2: Classe **sinusoidale**.

```
template <class Type> class makeWave : public  sequenze<Type>{
public:
    // Costruttore e distruttore
    makeWave(int n, double f_c, const vector<double>& v_freq):
    sequenze<Type>(n) {
        name = "makeWave";
        wave(f_c, v_freq);
    }
    ~makeWave(){}
    // Metodo per il calcolo di una nota
    void wave(int f_c, const vector<double>& vecFreq) {
        const double TWO_PI = 8.0 * atan(1.0); int i;
        for(i = 0; i < vecFreq.size(); i++ )
            {
            // Add the sinusoid to the waveform
            double arg = TWO_PI * vecFreq[i];
            for( int j = 0; j < sequenze<Type>::size; j++ ){
                sequenze<Type>::t[j] = (double)j/f_c;
                sequenze<Type>::x[j] +=
                (Type) sin( sequenze<Type>::t[j] * arg);
                }
            }
        for(i = 0; i < sequenze<Type>::size; i++ )
            sequenze<Type>::x[i] /= vecFreq.size();
       }
private:
    string name;
}; #endif //MAKEWAVE_H
```

Capitolo 3

Dominio del tempo

Un segnale audio può essere rappresentato graficamente osservando la sua ampiezza al variare del tempo ($x(t)$). Come abbiamo discusso precedentemente, un segnale continuo per essere elaborato da un sistema digitale, ha bisogno di una digitalizzazione. Un segnale audio, quindi, non è altro che una sequenza numerica discreta che può essere facilmente memorizzata (array mono-dimensionale). In questo Capitolo descriveremo alcune caratteristiche dei segnali nel dominio del tempo e approfondiremo il concetto dei sistemi lineari ed invarianti alla traslazione.

3.1 I segnali

Matematicamente i segnali sono rappresentati come funzioni di una o più variabili indipendenti. Un segnale vocale, ad esempio, è descritto in funzione del tempo e una fotografia può essere rappresentata come una luminosità in funzione di due variabili spaziali. La variabile indipendente può essere continua o discreta. I *segnali a tempo continuo* sono definiti su un insieme temporale continuo e quindi rappresentato da funzioni di variabili continue. I *segnali a tempo discreto* sono definiti su un insieme discreto di tempi e la variabile indipendente assume solo valori discreti. I segnali a tempo discreto sono rappresentati come sequenze di numeri. Nei segnali numerici sia il tempo che l'ampiezza sono discreti. I sistemi di elaborazione a loro volta possono essere classificati come *sistemi a tempo continuo*, cioè sia l'ingresso che l'uscita sono segnali a tempo continuo e *sistemi a tempo discreto* dove l'ingresso e l'uscita sono segnali a tempo discreto.

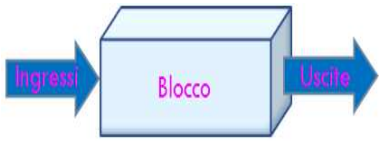


Figura 3.1: Definizione di blocco tramite la relazione Ingresso/Uscita.

3.2 Sistemi

I *controlli automatici* rivestono un ruolo essenziale nel progresso tecnologico della civiltà umana come ad esempio per la costruzione di lavatrici, frigoriferi, forni domestici, piloti automatici di aerei, i robot, ecc.. Un *sistema* è un insieme o un raggruppamento di elementi e cioè i sosttosistemi. Un sistema può essere complesso (e.g sistema planetario) o semplice (e.s. sistema di controllo della temperatura dell'acqua). Un elemento è la parte più piccola di un sistema che può essere trattata come un'*entità*. Il *blocco* è un insieme di elementi che possono essere raggruppati e descritti da una relazione ingresso/uscita (input/output) (vedi Figura 3.1). Un *sistema di controllo* è un insieme di componenti fisici disposti in modo tale da controllare se stessi oppure un altro sistema. Il *sistema di controllo automatico* è un sistema di controllo che si autoregola, senza la necessità di intervento umano. Il sistema di controllo in *anello aperto* è un sistema di controllo nel quale l'azione di controllo è indipendente dall'uscita. Un sistema di controllo in *anello chiuso*, è un sistema nel quale l'azione di controllo è influenzata dall'uscita. Successivamente questi due ultimi concetti ci serviranno per caratterizzare i filtri.

3.2.1 Tipi di sistemi

La *retroazione* in un sistema di controllo in anello chiuso permette al segnale di uscita al sistema di essere riportato in ingresso (feedback) (e.g. autista che guida). Un sistema *SISO* è un sistema singolo-ingresso, singola-uscita (SISO, Single-Input, Single-Output) variabili in ingresso e uscita scalari (e.g.: sistema di riscaldamento di una casa). Un sistema *MIMO* è un sistema multi-ingresso, multi-uscita (MIMO,Multiple-Input, Multiple-Output) in cui le variabili in ingresso e uscita vettoriali. Un *sistema lineare* è un un sistema in cui le relazioni di ingresso/uscita possono essere rappresentate in modo lineare (e.g. la legge di Ohm (V = R I)). Un *sistema tempo-invariante* è un sistema descrivibile con coefficienti

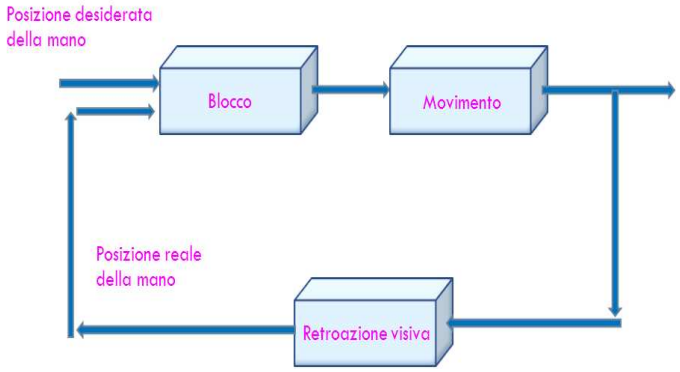


Figura 3.2: Schema di un sistema con retroazione: persona che prende il bicchiere sul tavolo.

costanti (e.g. sistema molla, massa e smorzatore). Un *sistema tempo-variante* è un sistema descrivibile con coefficienti variabili (e.g. razzo che brucia carburante). In Figura 3.2 viene presentato uno schema di un sistema a retroazione. Esso descrive le fasi di una persona che tenta di prendere un bicchiere su di un tavolo.

3.3 Sequenze discrete

Nei sistemi a tempo discreto i segnali sono rappresentati numericamente. Una sequenza di numeri $\mathbf{x}$, in cui l'n-esimo numero della sequenza è indicato con $x(n)$, è rappresentata formalmente come

$$\mathbf{x} = \{x(n)\} \quad -\infty < n < \infty \tag{3.1}$$

La sequenza $x(n)$ è definita solo per valori interi di n e una sequenza finita può essere rappresentata nella memoria di un elaboratore digitale come un array monodimensionale. Per capire il concetto di sequenza discreta finira in Figura 3.3 viene presentato il grafico di una sequenza. La sequenza in questo caso è finita ed è composta da 21 punti nell'intervallo $-9 \leq n \leq 11$ con n intero. Le ampiezze $x(n)$ sono indicate con un cerchio blue. Vediamo ora l'implementazione dei costruttori della classe **sequenze** introdotta nel Capitolo 1. La classe presenta l'overload di due costruttori e sono descritti nel seguente codice

```
// Classe sequenze
template <class Type> class sequenze{
```

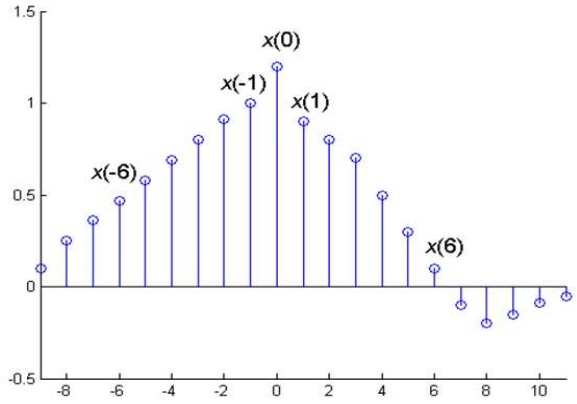


Figura 3.3: Esempio di sequenza discreta.

```
public:
  // Costruttori
sequenze(int n)
      {
       size = n;
       t = vector<double>(n,0.0);
       x = vector<Type>(n,0.0);
       for(int i=0;i<n;i++){t[i] = i;}
      }

sequenze(double inf, double sup, double step)
      {
       int n = (int)(sup - inf)/step + 1;
       double c;
       size = n;
       t = vector<double>(n,0.0);
       x = vector<Type>(n,0.0);
       for(int i=0, c = inf; c < sup ; i++, c +=
       step){t[i] = c;}
    }
```

Lo scopo dell'array t (oggetto *vector*) è quello di contenere l'informazione temporale n della sequenza discreta. Ad esempio $t = [-9, -8, -7, \ldots, 11]$ per la sequenza in Figura

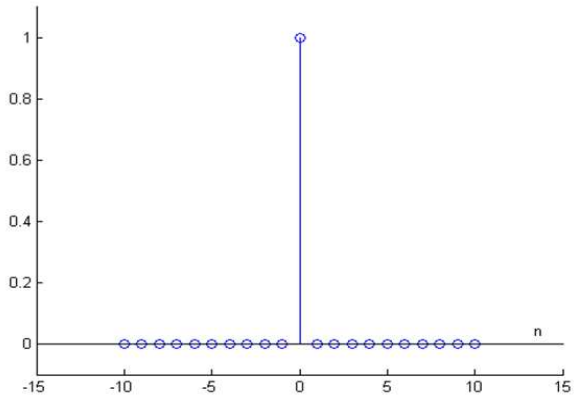


Figura 3.4: Sequenza discreta: impulso.

3.3. L'array x contiene le ampiezze del segnale discreto e cioè $[x(-9), x(-8), \ldots, x(11)]$ dello stesso esempio. Il primo costruttore della classe dato un intero n crea un intervallo temporale da 0 ad $n-1$ con passo 1 ($t = [0, 1, \ldots, n-1]$). Il secondo costruttore dato un intervallo inferiore inf, un intervallo superiore sup e un passo step crea un intervallo temporale $t = [\text{inf}, \text{inf} + \text{step}, \text{inf} + 2\text{step}, \ldots, \text{sup} - \text{step}, \text{sup}]$.
Successivamente verranno introdotte le sequenze più usate nell'ambito dell'elaborazione dei segnali audio e la loro implementazione in C++.

3.3.1 Campione unitario

La sequenza discreta *campione unitario* è spesso indicata come *impulso* a tempo discreto o semplicemente come impulso ed ha la seguente forma

$$x(n) = \delta(n) = \begin{cases} 0 & \text{se } n \neq 0 \\ 1 & \text{se } n = 0 \end{cases} \tag{3.2}$$

In Figura 3.4 viene rappresentata la sequenza impulso. L'implementazione della classe **impulso** viene presentata successivamente. Essa estende la classe **sequenze**. La classe presenta due costruttori con le stesse caratteristiche dei costruttori introdotti precedentemente per la classe **sequenze**. La classe è descritta dalla Tabella 9.16.2 e il codice è il seguente

```
template <class Type> class impulso : public  sequenze<Type>{
```

impulso				
#	name	:	string	nome della sequenza
+	impulso (int)			costruttore
+	impulso (double, double, double)			costruttore
+	getname()	:	string	nome
+	delta(int)	:	double	sinusoide

Tabella 3.1: Classe **impulso**.

```
public:
    // Funzione impulso
    double delta(int n){ if( n == 0) return 1;
         else
    return 0; }
    // Funzione impulso iterata
    impulso(int n) : sequenze<Type>(n) {
    name = "Impulso";
    for(int i=0; i<size; i++)
    {
     x[i] = delta((int) t[i]);
    }
    }
    impulso(double inf, double sup, double step):
    sequenze<Type>(inf,sup,step) { name = "Impulso";
    for(int i=0;i<size;i++)
    {
     x[i] = delta((int) t[i]);
    }
    }
    string get_name() {
        return name;
    }
private:
    string name;
};
```

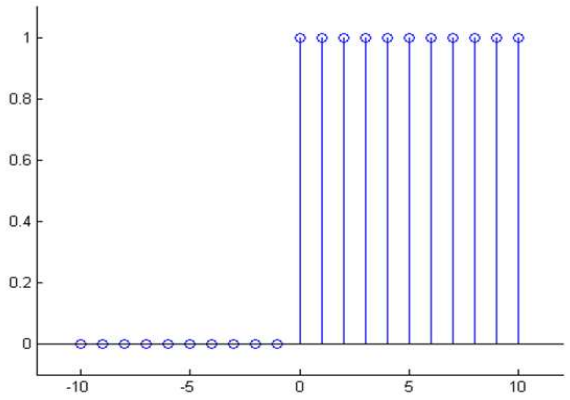


Figura 3.5: Sequenza discreta: gradino unitario.

unitaria				
#	name	:	string	nome della sequenza
+	unitaria (int)			costruttore
+	unitaria (double, double, double)			costruttore
+	getname()	:	string	nome
+	unit(int)	:	int	sinusoide

Tabella 3.2: Classe **unitaria**.

3.3.2 Gradino unitario

La sequenza *gradino unitario*, invece, assume i seguenti valori

$$x(n) = \begin{cases} 0 & \text{se } n < 0 \\ 1 & \text{se } n \geq 0 \end{cases} \tag{3.3}$$

In Figura 3.5 viene rappresentato un esempio di sequenza del gradino unitario. L'implementazione in C++ della classe **unitaria** è la seguente (la classe è descritta nella Tabella 5.5.4)

```
template <class Type> class unitaria : public  sequenze<Type>{
public:
    // Funzione unitaria
    int unit(double x) {
       if (x >= 0)
              return 1;
       else
              return 0;
    }
```

esponenziale				
#	name	:	string	nome della sequenza
+	esponenziale (int, double)			costruttore
+	esponenziale (double, double, double, double)			costruttore
+	getname()	:	string	nome
+	realexp(double, int)	:	int	sinusoide

Tabella 3.3: Classe **esponenziale**.

```
    // Funzione impulso iterata
    unitaria(int n) : sequenze<Type>(n) { name = "unitaria";
    for(int i=0;i<size;i++)
        {
     x[i] = unit(t[i]);
    }
    }
    unitaria(double inf, double sup, double step):
    sequenze<Type>(inf,sup,step) { name = "unitaria";
    for(int i=0; i<size; i++)
    {
     x[i] = unit(t[i]);
    }
    }
    string get_name() {
        return name;
    }
private:
    string name;

};
```

3.3.3 Esponenziale reale

Una sequenza *esponenziale a valori reali* è una qualsiasi sequenza i cui valori sono della forma

$$x(n) = a^n \quad \forall n \in \Re \tag{3.4}$$

In Figura 3.6 viene presentata la sequenza esponenziale reale in cui è stato scelto $a = 0.9$. La Tabella 4.5.2 descrive la classe **esponenziale** e l'implementazione C++ risulta

```
template <class Type> class esponenziale : public  sequenze<Type>{
```

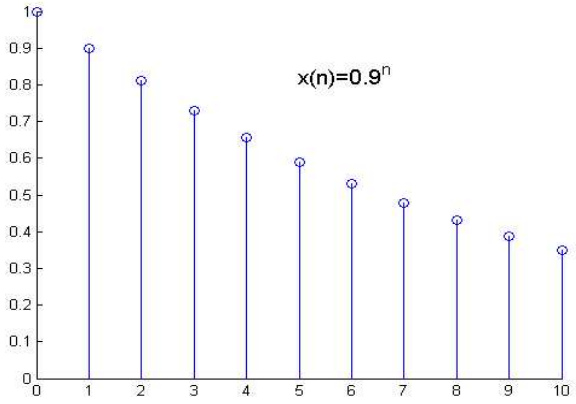


Figura 3.6: Sequenza discreta: esponenziale reale.

```
public:
    // Funzione esponenziale reale
    double real_exp(double x, int n) {
    return pow(x,n);
    }
    // Costruttori
    esponenziale (int n, double a) : sequences<Type>(n) {
    name ="esponenziale reale";
    for(int i=0; i<size; i++)
    {
     x[i] = real_exp(t[i],a);
    }
    }
    esponenziale(double inf, double sup, double step, double a):
    sequenze<Type>(inf,sup,step) { name = "esponenziale reale";
    for(int i=0; i<size; i++)
    {
     x[i] = real_exp(t[i], a);
    }
    }
    string get_name() {
    return name;
    }
private:
```

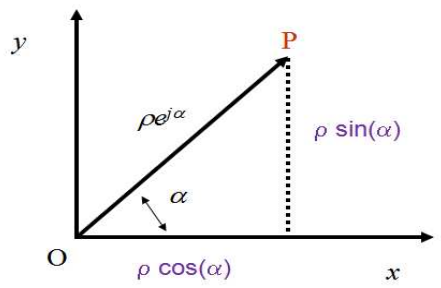


Figura 3.7: Operatore esponenziale e numeri complessi.

```
        string name;
};
```

3.3.4 Sequenza esponenziale complessa

La sequenza *esponenziale complessa* risulta meno immediata delle precedenti. Per costruire questa sequenza dobbiamo fare riferimento ai numeri complessi. Sia dato un numero complesso di modulo ρ e argomento α abbiamo la seguente relazione (formula di Eulero)

$$\rho\left(\cos(\alpha) + j\sin(\alpha)\right) = \rho e^{j\alpha} \tag{3.5}$$

dove e è il numero di Nepero, $e = 2.718281828$. Lo scalare $e^{j\alpha}$ prende il nome di operatore e permette di ruotare di un angolo o fase α il modulo ρ nel senso antiorario (vedi Figura 3.7).
La sequenza discreta esponenziale complessa viene descritta come

$$x(n) = Ae^{j2\pi fn+\phi_0} \tag{3.6}$$

dove la fase $\alpha = 2\pi fn+\phi_0$ e l'ampiezza $|x(n)| = A = \rho$, ϕ_0 è la fase iniziale, f la frequenza, $\omega = 2\pi f$ è la frequenza angolare e $T = 1/f$ il periodo. In Figura 3.8 viene presentata la corrispondenza tra le funzioni trigonometriche seno e coseno e la forma esponenziale che varia nel tempo. Dalla formula di Eulero abbiamo inoltre

$$x(n) = Ae^{j\omega n+\phi_0} = A\left(cos(\omega n) + j\sin(\omega n)\right) \tag{3.7}$$

Nelle Figure 3.9 e viene presentata la parte reale (rosso) e la parte immaginaria (blue) (coseno e seno, rispettivamente) della sequenza $x(n) = e^{j2\pi 0.3n}$. In Figura 3.10 l'andamento nel tempo della fase (rosso) e dell'ampiezza (nero). Il codice C++ della classe

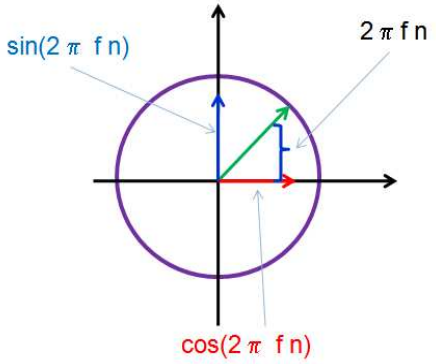


Figura 3.8: Forma esponenziale e fase nel dominio del tempo.

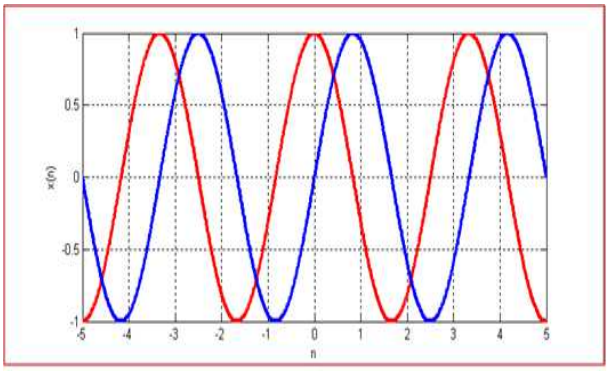


Figura 3.9: Parte reale (rosso) e immaginaria (blu) di una sequenza esponenziale complessa.

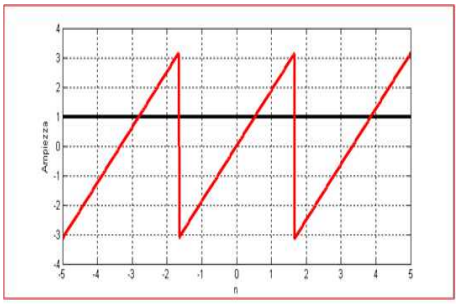


Figura 3.10: Fase (rosso) e ampiezza (nero) di una sequenza esponenziale complessa.

cesponenziale				
#	name	:	string	nome della sequenza
#	fase	:	vector¡double¿	vettore fase
#	modulo	:	vector¡double¿	vettore modulo
+	cesponenziale (int, double, double, double)			costruttore
+	cesponenziale (double, double, double, double, double, double, double)			costruttore
+	getreal()	:	vector<double>	parte reale
+	getimag()	:	vector<double>	parte immaginaria
+	getfase()	:	vector<double>	fase
+	getmodulo()	:	vector<double>	modulo
+	getname()	:	string	nome

Tabella 3.4: Classe **cesponenziale**.

cesponenziale è il seguente (la Tabella 3.3.4 contiene la sua descrizione)

```
template <class Type>
class c_exponential : public  sequenze<Type>{
public:


c_exponential(int n, double f, double phase_0, double A) : sequenze<Type>(n)
{
name = "esponenziale complesso";


modulo = vector<double>(n,0.0);
fase = vector<double>(n,0.0);


for(int i=0;i<size;i++)
    {
     x[i].c_exp(A, 2*PI*f*t[i] + phase_0);
     modulo[i] = x[i].cabs();
     fase[i] =  atan2(x[i].imag(),x[i].real()) ;
    }
}


c_exponential(double inf, double sup, double step,
double f, double phase_0, double A) : sequenze<Type>(inf,sup,step)
{
name = "esponenziale complesso";
int n = (sup - inf)/step +1;
modulo = vector<double>(n,0.0);
fase = vector<double>(n,0.0);
```

```
int p = 0;
for(int i=0; i<size; i++)
    {
      x[i].c_exp(A, 2*PI*f*t[i] + phase_0);
      modulo[i] = x[i].cabs();
      fase[i] = atan2(x[i].imag(), x[i].real()) ;
    }
}

// Metodi per ottenere la parte reale e immaginaria

    vector<double> get_real() {
        vector<double> v(size,0.0);
        for(int i=0;i<x.size();i++) v[i] =
        x[i].real();

        return v;

    }

    vector<double> get_imag() { vector<double> v(size,0.0);
        for(int i=0;i<x.size();i++) v[i] = x[i].imag();

        return v;
    }

    vector<double> get_modulo() { return modulo; }
    vector<double> get_fase() { return fase; }

string get_name()
{
    return name;
}

private:
    vector<double> fase, modulo;
    string name;
};
```

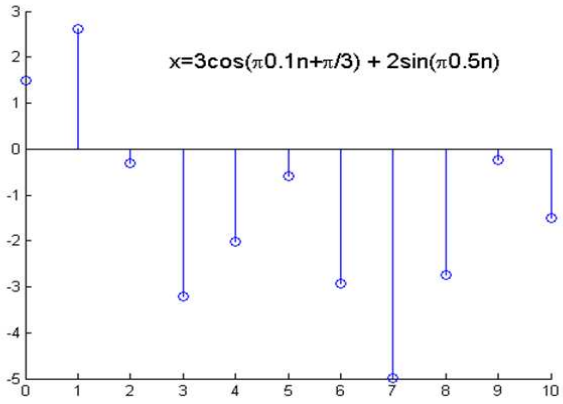


Figura 3.11: Sequenza discreta: sinusoidale.

sinusoidale				
#	name	:	string	nome della sequenza
+	sinusoidale (int, double, double, double)			costruttore
+	sinusoidale (double, double, double, double, double, double, double)			costruttore
+	sin(double, double, double, int)	:	double	funzione seno
+	getname()	:	string	nome

Tabella 3.5: Classe **sinusoidale**.

3.3.5 Sequenza sinusoidale

L'ultima sequenza che introduciamo ma che abbiamo già incontrato nel Capitolo 1 è la *sequenza sinusoidale*. La funzione è definita come

$$x(n) = A\cos(2\omega n + \phi) \tag{3.8}$$

dove A è l'ampiezza, ω è la frequenza in radianti, ϕ è la fase iniziale in radianti. La frequenza ω può essere scelta in un insieme continuo di valori. Possiamo vincolare ω ad essere continua nel campo $0 \leq \omega \leq 2\pi$ o a quello equivalente $-\pi \leq \omega \leq \pi$. Le ω nel campo $-2\pi k \leq \omega \leq 2\pi(k+1)$ sono del tutto identiche, per qualsiasi k, a quelle ottenute facendo variare ω nell'intervallo $-\pi \leq \omega \leq \pi$. In Figura 3.11 viene presentata una sequenza sinusoidale. La classe **sinusoidale** è descritta nella Tabella 3.3.5 e la sua implementazione risulta

```
template <class Type> class sinusoidale : public  sequenze<Type>{
public:
    // Funzione sinusoidale
```

```
    double sin(double A, double f, double phase, int n) {
        return A * cos(2*PI*f*n + phase);
    }
    // Funzione sinusoidale iterata
    sinusoidale(int n, double A, double f, double phase):
    sequences<Type>(n) { name = "sinusoidale";
    for(int i=0; i<=size; i++)
    {
     x[i] = sin(A,f,phase,t[i]);
    }
    }
    sinusoidale(double inf, double sup, double step, double A, double f,
    double phase) : sequenze<Type>(inf,sup,step) { name = "sinusoidale";
    for(int i=0; i<=size; i++)
    {
     x[i] = sin(A,f,phase,t[i]);
    }
    }
    string get_name() {
        return name;
    }
private:
    string name;
};
```

3.4 Operazioni di base

Diamo ora uno sguardo alle operazioni principali che si possono effettuare su una sequenza discreta. Le operazioni di base sono quelle di somma, prodotto e prodotto scalare. Supponendo di avere due sequenze $x(n)$ e $y(n)$ esse sono definite come

$$\begin{array}{lcll} x & + & y & = & \{x(n) + y(n)\} & \text{somma} \\ x & \cdot & y & = & \{x(n) \cdot y(n)\} & \text{prodotto} \\ c & \cdot & x & = & \{c \cdot x(n)\} & \text{prodotto scalare} \end{array} \tag{3.9}$$

Le operazioni sono definite nella classe **sequenze** e la sua implementazione è la seguente

```
sequenze& somma(sequenze *s)
```

```
    {
        for(int i=0; i< size; i++) x[i] += s->get_x(i);
        return *this;
    }
sequenze& prodotto(sequenze *s)
    {
        for(int i=0; i< size; i++) x[i] *= s->get_x(i);
        return *this;
    }
sequenze& s_prodotto(double alpha)
    {
        for(int i=0; i< size; i++) x[i] *= alpha;
        return *this;
    }
```

Un'informazione importante legata ad una singola sequenza $x(n)$ risulta l'energia E. Essa è definita come

$$E = \sum_{n=-\infty}^{\infty} |x(n)|^2 \tag{3.10}$$

Anche questo metodo è definito nella classe **sequenze** e l'implementazione è la seguente

```
Type energia(){
        Type en = (Type) 0;
        for(int i=0; i< size; i++) en += pow(abs(x[i]),2);
        return en;
    }
```

3.5 Schema a blocchi

Sulle sequenze possono essere effettuate diverse operazioni più complesse di quelle già considerate. Prima di introdurre queste nuove operazioni ci soffermiamo ora sul concetto di *diagramma a blocchi*. In un diagramma a blocchi ogni componente è rappresentato da un blocco SISO o MIMO e tutti i blocchi sono interconnessi da frecce che rappresentano la direzione di flusso del segnale. L'operazione di un blocco o componente è descritta matematicamente (funzione $T[.]$) all'interno del blocco stesso. Questa risulta una trasformazione univoca di una sequenza di ingresso $x(n)$ in una sequenza di uscita $y(n)$ (vedi Figura 3.12).

Figura 3.12: Schema a blocchi.

3.5.1 Sequenza ritardata

Si dice che la sequenza y è una versione ritardata o traslata della sequenza x se y ha valori

$$y(n) = x(n - n_0) \tag{3.11}$$

dove n_0 è un intero.
Anche questo metodo è definito in C++ nella classe **sequenze** ed è implementato nel seguente modo

```
// ritardo n_0
sequenze& ritardo(int no)
    {
    y = vector<Type(size,0.0);
    if (no >= 0)
    {
        for(int i=0; i < no; i++) { y[i] = 0;}
        for(i=no; i< size; i++) { y[i] = x[i-no];}
    }
     else  cout << "errore" << "\n" ;
    x = y;
    return *this;
    }
```

3.5.2 Rappresentazione tramite impulsi

Descriviamo ora un risultato che ci sarà molto utile successivamente. Una sequenza arbitraria può essere espressa come la somma di campioni unitari scalati e ritardati

$$y(n) = \sum_{k=-\infty}^{\infty} x(k)\delta(n - k) \tag{3.12}$$

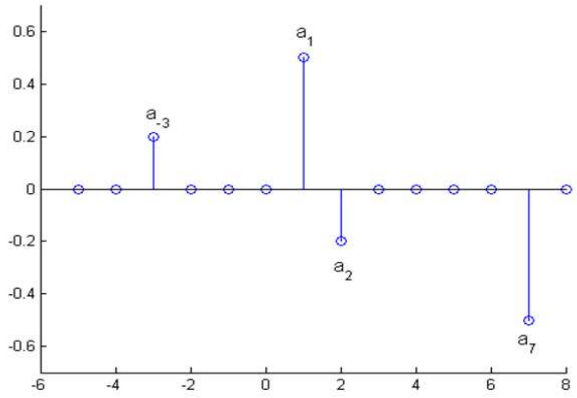


Figura 3.13: Rappresentazione tramite impulsi.

Ad esempio in Figura 3.13 viene rappresentata la sequenza $x(n) = a_{-3}\delta(n+3)+a_1\delta(n-1)+a_2\delta(n-2)+a_7\delta(n-7)$. Il metodo è definito nella classe **sequenze** e la sua implementazione risulta

```
// Funzione impulso
double delta(int n){ if( n == 0) return 1;
     else
return 0; }
// rappresentazione tramite impulsi
sequenze& rapp_impulsi()
    {
    y = vector<Type>(size,0.0);
    int k;
    for(int n=0; n < size; n++)
      for(y[n] = 0,k=0; k< size; k++)
              y[n] += x[k]*delta(n-k);
    x = y;
   return *this;
}
```

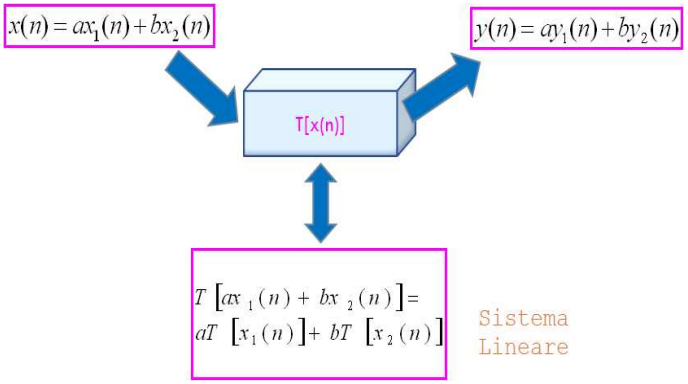


Figura 3.14: Principio di sovrapposizione.

Figura 3.15: Sistema tempo-invariante.

3.6 Sistemi LTI

La classe dei *sistemi lineari* è di fondamentale importanza nel campo dell'elaborazione dei segnali. Essa viene definita mediante il principio di sovrapposizione. Supponiamo di avere una sequenza $x(n)$ che è data dalla sovrapposizione di $x_1(n)$ e $x_2(n)$ tale che $x(n) = ax_1(n) + bx_2(n)$. Un sistema è lineare se otteniamo (vedi Figura 3.14)

$$y(n) = T\left[x(n)\right] = ay_1(n) + by_2(n) \tag{3.13}$$

dove $y_1(n) = T\left[x_1(n)\right]$ e $y_2(n) = T\left[x_2(n)\right]$ e cioè

$$T\left[x(n)\right] = T\left[ax_1(n) + bx_2(n)\right] = aT\left[x_1(n)\right] + bT\left[x_2(n)\right] \tag{3.14}$$

Inoltre abbiamo che un sistema è a *tempo-invariante* se data una risposta $y(n)$ di un sistema a $x(n)$ allora $y(n-k)$ è la risposta a $x(n-k)$ dove k è un intero positivo o negativo (vedi Figura 3.15).

3.7 Risposta all'impulso

Facendo riferimento ai risultati introdotti ed in particolare quelli relativi ai sistemi LTI caratterizziamo una *risposta all'impulso*. La risposta all'impulso è alla base dei filtri numerici che vedremo nei Capitoli successivi. Sapendo che una sequenza arbitraria $x(n)$ può essere rappresentata come somma di sequenze di campioni unitari ritardati e scalati, possiamo affermare che un sistema lineare può essere completamente caratterizzato dalla sua risposta impulsiva (o risposta al campione unitario). In altre parole abbiamo che l'output $y(n)$ del sistema può essere descritto come

$$y(n) = T\left[x(n)\right] = T\left[\sum_{k=-\infty}^{\infty} x(k)\delta(n-k)\right]. \tag{3.15}$$

Se consideriamo il sistema LTI possiamo riscrivere l'equazione nel seguente modo

$$y(n) = \sum_{k=-\infty}^{\infty} T\left[x(k)\delta(n-k)\right] = \sum_{k=-\infty}^{\infty} x(k)T\left[\delta(n-k)\right] \tag{3.16}$$

se poniamo $T\left[\delta(n-k)\right] = h(n-k)$ che corrisponde alla risposta del sistema all'impulso $\delta(n-k)$ otteniamo

$$y(n) = \sum_{k=-\infty}^{\infty} x(k)h(n-k) \tag{3.17}$$

La risposta h viene chiamata risposta all'impulso. La precedente equazione descrive la convoluzione di una sequenza $x(n)$ e di una risposta all'impulso $h(n)$. Ogni sistema lineare invariante alla traslazione è completamente caratterizzato dalla risposta al campione unitario $h(n)$. Per semplicità la convoluzione di $x(n)$ e $h(n)$ è indicata come

$$y(n) = x(n) \otimes h(n) \tag{3.18}$$

Abbiamo inoltre che dalla proprietà di invarianza alla traslazione se $h(n)$ è la risposta a $\delta(n)$ allora $h(n-k)$ è la risposta a $\delta(n-k)$. Quindi possiamo equivalentemente scrivere

$$y(n) = \sum_{k=-\infty}^{\infty} x(k)h(n-k) = \sum_{k=-\infty}^{\infty} h(n)x(n-k) = h(n) \otimes x(n) \tag{3.19}$$

Anche questo metodo è definito nella classe **sequenze** e questa è la sua implementazione

```
// convoluzione
sequenze& conv(vector<double> h, int N)
```

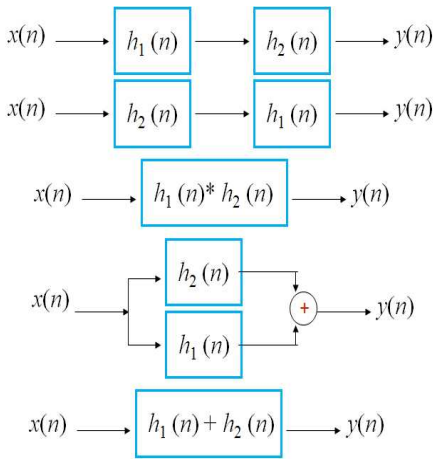


Figura 3.16: Operazioni di convoluzione.

```
    {
     y = vector<Type>(size+N,0.0);
     int m;
     for(int n = 0; n < size+N; n++)
      for (y[n] = 0, m = max(0, n-size+1); m <= min(n,N); m++)
                    y[n] += h[m] * x[n-m];
    x = y;
    return *this;
}
```

In Figura 3.16 vengono presentate le operazioni che si possono ottenere considerando due risposte all'impulso $h_1(n)$ e $h_2(n)$. Nei Capitoli successivi useremo il concetto di convoluzione per costruire un filtro nel dominio del tempo. Infatti, è possibile ottenere un filtraggio di un segnale (sequenza) composto da più frequenze facendo la convoluzione dello stesso con una risposta all'impulso $h(n)$ che descrive un seno cardinale.

3.8 Equazioni alle differenze

Un ultimo risultato che descriviamo è quello delle *equazioni alle differenze*. Esse possono essere considerate come lo schema matematico generale per la descrizione di sistemi con o senza feedback. Come vedremo nei prossimi Capitoli esse ci serviranno per progettare ed implementare i filtri basati su risposte impulsive infinite o finite. Consideriamo un sistema

con input $x(n)$ e output $y(n)$. Un'equazione alle differenze lineare a coefficienti costanti soddisfa la seguente relazione

$$\sum_{k=0}^{N} a_k y(n-k) = \sum_{r=0}^{M} b_r x(n-r) \tag{3.20}$$

La relazione esplicita tra ingresso e singola uscita $y(n)$ può essere ricavata semplicemente e risulta

$$y(n) = \sum_{r=0}^{M} \frac{b_r}{a_0} x(n-r) - \sum_{k=1}^{N} \frac{a_k}{a_0} y(n-k) \tag{3.21}$$

In altre parole, l'n-esimo valore dell'uscita può essere calcolato usando l'n-esimo valore dell'ingresso e gli N ed M valori precedenti, rispettivamente dell'uscita e dell'ingresso. Un sistema lineare invariante alla traslazione può avere una risposta al campione unitario di durata finita o infinita. Se la risposta all'impulso è di durata finita, parliamo di sistema con risposta all'impulso finita (sistema *FIR*). Se la risposta all'impulso è di durata infinita parliamo di sistema con risposta all'impulso infinita (sistema *IIR*). Un sistema FIR può essere sempre descritto da un'equazione alle differenze con $N = 0$. Invece per un sistema IIR dobbiamo avere $N > 0$. Possiamo notare che dall'equazione alle differenze ponendo $N = 0$ il sistema FIR corrispondente risulta

$$y(n) = \sum_{r=0}^{M} \frac{b_r}{a_0} x(n-r). \tag{3.22}$$

Ponendo

$$h(r) = \tfrac{b_n}{a_0} \quad \forall r = 0, 1, \ldots, M \tag{3.23}$$

otteniamo la seguente convoluzione

$$y(n) = \sum_{r=0}^{M} h(r) x(n-r). \tag{3.24}$$

Il metodo per il calcolo delle equazioni alle differenze è definito nella classe **sequenze** e il suo codice C++ è il seguente

```
// equazioni alle differenze
sequenze& eq_diff(vector<double> a, int N, vector<double> b, int M)
{
 y = vector<Type> (size+N,0.0);
 int k,r;
```

```
 for (int n = 0; n < size+N; n++)
 {
  for (y[n] = 0, r = max(0, n-size+1); r <= min(n, M); r++)
                y[n] += b[r]/a[0] * x[n-r];

   for (k = max(1, n-size+1); k <= min(n, N); k++)
                y[n] += a[k]/a[0] * x[n-k];
 }
x = y;
return *this;
}
```

Capitolo 4

Dominio delle frequenze

Nel Capitolo precedente abbiamo analizzato e caratterizzato i segnali nel dominio tempo. Molte caratteristiche di un segnale audio però possono essere evidenziate solo analizzandolo nel dominio delle frequenze. Ad esempio nel dominio del tempo risulta difficile distinguere suoni complessi composti dalla sovrapposizione di diversi toni puri. Se si fa ricorso alla Trasformata di Fourier, nel dominio delle frequenze, questo risulta molto più semplice. In questo Capitolo introdurremo i meccanismi per la rappresentazione dei segnali nel dominio delle frequenze. Partendo dal Teorema di Fourier, ci concentreremo sulla Trasformaza z, la Trasformata di Fourier a tempo discreto (Discrete Time Fourier Transform, DTFT) e la Trasformata di Fourier Discreta (Discrete Fourier Transform, DFT).

4.1 Teorema di Fourier

Come abbiamo introdotto nel Capitolo 1, il Teorema di Fourier afferma che un segnale periodico qualsiasi può essere considerato come la sovrapposizione di onde sinusoidali semplici, ciascuna con la sua ampiezza e fase, e le cui frequenze sono armoniche della frequenza fondamentale del segnale.
Inoltre abbiamo denominato come *Analisi di Fourier* il meccanismo che permette l'individuazione dei segnali semplici che compongono un segnale complesso. Viceversa abbiamo che la ricostruzione di un segnale audio a partire da sinusoidi semplici è detta *Sintesi di Fourier*. L'insieme delle componenti di un segnale, con la propria ampiezza e fase, è detto *Spettro di Fourier* (in particolare abbiamo lo spettro di ampiezza e lo spettro di fase). Le *armoniche* vengono dette anche *componenti* di Fourier. Vediamo ora un risultato importante nella teoria dei segnali, cioè la *Trasformata di Fourier*. La *Trasformata inversa di*

Fourier invece permette di effettuare il percorso opposto, dal dominio della frequenza al dominio del tempo. Successivamente vedremo sia matematicamente che algoritmicamente le Trasformate di Fourier, partendo da quelle per segnali continui.

4.2 Serie e Trasformata di Fourier continua

Nella teoria dei segnali, la trasformata di Fourier è lo strumento che permette di scomporre un segnale generico in una somma di sinusoidi con frequenze, ampiezze e fasi diverse. Se il segnale in oggetto è un segnale periodico, la sua trasformata di Fourier è un insieme discreto di valori (spettro discreto, o a pettine) in cui la frequenza più bassa è detta fondamentale ed è pari a quella del segnale stesso mentre tutte le altre frequenze sono multipli della fondamentale e prendono il nome di armoniche.
Consideriamo un suono o segnale audio continuo $x(t)$ reale, periodico con periodo T_0. Possiamo quindi definire la frequenza come $f_0 = 1/T_0$ e la pulsazione come $\omega_0 = 2\pi f_0$. Facendo ricorso alla *serie di Fourier*, questo segnale può essere descritto in questo modo

$$x(t) = A_0 + \sum_{k=1}^{\infty} A_k \cos(2\pi k f_0 t + \theta_k) = A_0 + \sum_{k=1}^{\infty} A_k \cos(\omega_0 k t + \theta_k) \tag{4.1}$$

dove $A_0, A_1, \ldots, A_k$ e θ_k sono valori reali che si possono calcolare in funzione dell'indice k e della funzione $x(t)$. D'altra parte conoscendo soltanto i valori $A_0, A_1, \ldots, A_k$ e θ_k per ogni k, relativi ad una data funzione $x(t)$ incognita e possibile, mediante la formula precedente, ricostruire la $x(t)$. Il secondo membro dell'uguaglianza prende il nome di *sviluppo in serie di Fourier* di $x(t)$. Si può allora pensare che la $x(t)$ è data da una sommatoria infinita (serie) di cosinusoidi con frequenze multiple della frequenza fondamentale f_0 e cioè la serie di Fourier. Possiamo ora osservare che se vogliamo scomporre in "sinusoidi" un segnale $x(t)$ non periodico dobbiamo usare invece la trasformata di Fourier, che si può vedere come una generalizzazione della serie di Fourier.
La *trasformata continua di Fourier* permette di analizzare il contenuto in frequenze di un segnale analogico. La fase di *sintesi* risulta

$$x(t) = \int_{-\infty}^{\infty} X(\omega) e^{j\omega t} d\omega = \int_{-\infty}^{\infty} X(f) e^{j2\pi f t} df. \tag{4.2}$$

Se osserviamo questa uguaglianza notiamo che essa e simile all'uguaglianza relativa alla serie di Fourier, discussa in precedenza, $X(f)df$ è l'analogo di A_k, dove $X(f)$ è proprio la

DTFT				
#	name	:	string	nome della sequenza
+	DTFT (double, double, int)			costruttore
+	dtft(vector<Type>,int)	:	void	Algoritmo DTFT
+	getname()	:	string	nome

Tabella 4.1: Classe **DTFT**.

trasformata continua di Fourier (*analisi*), mentre il termine complesso $e^{j2\pi ft}$ è l'operatore di rotazione nel dominio complesso che corrisponde alla sinusoide classica precedente. La trasformata continua di Fourier nel dominio delle pulsazioni diventa quindi

$$X(\omega) = \int_{-\infty}^{\infty} x(t)e^{-j\omega t}dt \tag{4.3}$$

o nel dominio delle frequenze a

$$X(f) = \int_{-\infty}^{\infty} x(t)e^{-j2\pi ft}dt. \tag{4.4}$$

La trasformata di Fourier è continua nel tempo t e nelle frequenze ω. Successivamente approfondiremo il concetto di trasformata di Fourier considerando sequenze discrete.

4.3 Trasformata di Fourier a tempo discreto

Come abbiamo osservato nei Capitoli precedenti nella fase di digitalizzazione un segnale analogico viene trasformato in un segnale discreto $x(n)$. Per una generica sequenza $x(n)$ la *Trasformata di Fourier a tempo discreto* (Discrete Time Fourier Transform - DTFT) diventa

$$X(\omega) = \sum_{n=-\infty}^{\infty} x(n)e^{-j\omega n} \tag{4.5}$$

La trasformata inversa invece risulta

$$x(n) = \frac{1}{2\pi}\int_{-\pi}^{\pi} X(\omega)e^{j\omega n}d\omega \tag{4.6}$$

In accordo con il Teorema di Fourier abbiamo che $x(n)$ può essere scomposto nella somma (integrale) di esponenziali complessi le cui ampiezze (infinitesime) e le fasi iniziali in funzione della frequenza sono date dalla Trasformata di Fourier in $\omega(X(\omega))$.
La classe **DTFT** è descritta nella Tabella 9.16.2. La sua implementazione in C++ risulta

```
template <class Type, class Type_1> class DTFT : public
sequenze<Type>{


public:
    DTFT(double wa, double wb, int n): sequenze<Type>(n){
        t = vector<double>(size);
        x = vector<Type>(size);
       double dw = (wb-wa)/size;
       for (int k=0; k<size; k++)
        {
            t[k] = (double)wa + k*dw;
            x[k].put(0.0,0.0);
       }
    }
void dtft( vector<Type_1> x_i, int L){
     complex Z(0.0,0.0);
     for (int k=0; k<size; k++)
        {
        Z.c_exp(1.0,-t[k]);
       for (int i=0; i<L; i++)
           x[k].prod(Z,x[k]).add(complex(x_i[i],0.0));
     }
}
private:
 string Name;
};
```

Essa estende la classe *sequenze*. La classe presenta un costruttore che ha come input le frequenze ω_a e ω_b che definiscono l'intervallo entro cui vanno scelte n pulsazioni (e.g. $\omega_a = 0$, $\omega_b = 2\pi$ e $n = 1000$). Il metodo *dtft* calcola i coefficienti della trasformata $DTFT$. Per mostrare le caratteristiche della DTFT in Figura 4.1a viene visualizzato un segnale composto dalla sovrapposizione di due sinusoidi con frequenze $f_1 = 150$ Hz e $f_2 = 440$ Hz, rispettivamente. In Figura 4.1b viene visualizzato lo spettro di Fourier ($|X(f)|^2$) nel quale sono evidenti due picchi reltivi alle due componenti del segnale.

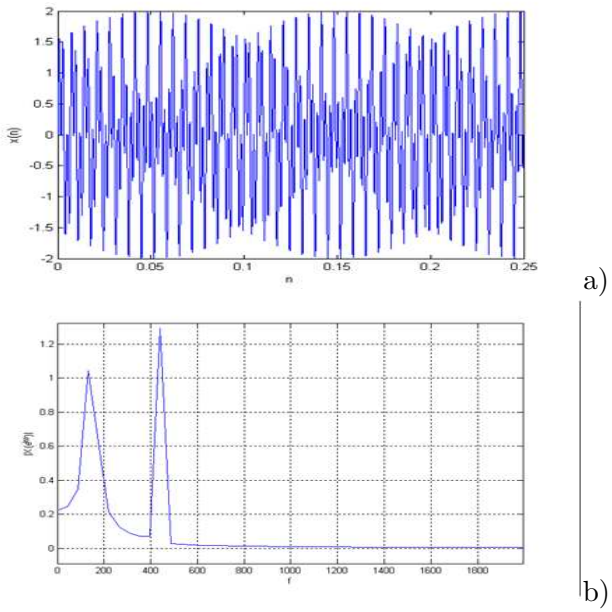


Figura 4.1: DTFT: a) segnale complesso; b) spettro di Fourier.

4.3.1 Risposta in frequenza

Uno dei risultati fondamentali, già visto nel Capitolo precedente, è quello della descrizione di un sistema mediante la sua risposta all'impulso. In questa sessione rappresentiamo un sistema tramite una *risposta in frequenza* e vediamo la sua relazione con la risposta all'impulso.

Supponiamo di avere una risposta all'impulso $h(n)$. Possiamo calcolare la *funzione di trasferimento* o *risposta in frequenza* della risposta $h(n)$ mediante la DTFT

$$H(\omega) = \sum_{n=-\infty}^{\infty} h(n)e^{-j\omega n} \tag{4.7}$$

Sottolinemao che la risposta all'impulso $h(n)$ può essere ottenuta dalla risposta in frequenza facendo la trasformata inversa di Fourier. Essa risulta, infatti

$$h(n) = \frac{1}{2\pi}\int_{-\pi}^{\pi} H(\omega)e^{j\omega n}d\omega. \tag{4.8}$$

La relazione che intercorre tra le due risposte di un sistema è evidente quando si cerca di caratterizzare un filtro passa-basso ideale come vedremo successivamente.

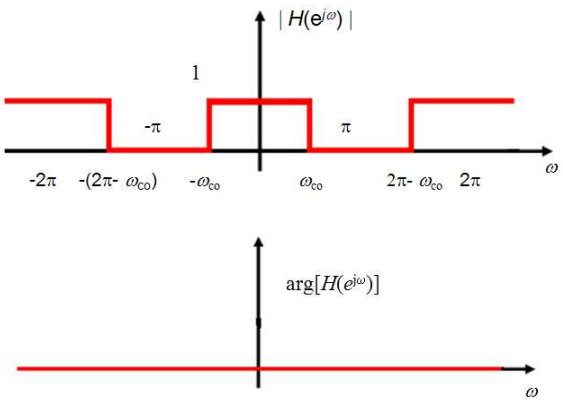


Figura 4.2: Filtro passa-basso ideale.

4.3.2 Filtro passa-basso ideale

Un filtro è un dispositivo o un metodo che data una sequenza permette di effettuare una trasformazione dello stesso. Esso può essere descritto come un sistema che prende in input una sequenza discreta $x(n)$ e ritorna una sequenza modificata $y(n)$. Come abbiamo osservato precedentemente un sistema può essere descritto mediante una risposta all'impulso nel dominio del tempo o tramite una risposta in frequenza nel dominio della frequenza. Un filtro *passa-basso ideale* permette di eliminare tutte le frequenze sotto una soglia ω_c dal segnale $x(n)$. Esso è caratterizzato da una risposta in frequenza $H(\omega)$ con ampiezza diversa da zero in una banda di frequenze simmetrica rispetto all'origine. In dettaglio abbiamo che il filtro ideale passa-basso a tempo discreto con frequenza di taglio ω_c ha come risposta in frequenza un rettangolo di ampiezza unitaria e base $2\omega_c$. Cioè abbiamo che $H(\omega)$ per $-\pi < \omega < \pi$ è definito matematicamente come (vedi Figura 4.2)

$$H(\omega) = \begin{cases} 1 & |\omega| \leq \omega_c \\ 0 & \omega_c < |\omega| < \pi \end{cases} \tag{4.9}$$

Applicando la Trasformta inversa di Fourier otteniamo la risposta all'impulso che corrisponde al filtro nel dominio del tempo che in questo caso è il *seno cardinale*

$$h(n) = \frac{1}{2\pi} \int_{-\omega_c}^{\omega_c} e^{j\omega n} d\omega = \frac{\sin(\pi w_c n)}{\pi n} \tag{4.10}$$

Il codice C++ della classe che implementa il seno cardinale (**sinc**) è il seguente (La Tabella 5.5.4 descrive la classe)

sinc		
+	sinc (double, int)	costruttore

Tabella 4.2: Classe **sinc**.

```
template<class Type> class sinc : public sequenze<Type>{
public:
    sinc(double wc, int M) : sequenze<Type>(M)
    {
    double alpha = (M-1)/2;
    for(int i=0;i<M;i++)
    x[i] =
      float (sin((wc * (i - alpha + 2.2204e-016)))/
     (PI*(i - alpha + 2.2204e-016)));
    }
};
```

4.3.3 Proprietà della trasformata

La trasformata di Fourier ha diverse proprietà che la rendono fondamentale ed applicabile a diversi contesti. Due proprietà importanti sono quelle di linearità e di traslazione che vedremo successivamente quando studieremo le propietà della Trasformata z. Vediamo ora alcuni Teoremi importanti.

Teorema di convoluzione

Un importante risultato è quello che vine chiamato "Teorema di convoluzione". Supponiamo di avere una sequenza discreta $x(n)$ e la sua trasformata $X(\omega)$ e una risposta all'impulso $w(n)$ con trasformata $W(\omega)$ allora definiamo la *convoluzione* di $x(n)$ e $h(n)$ come

$$y(n) = \sum_{k=-\infty}^{\infty} x(k)h(n-k) = x(n) \otimes h(n). \tag{4.11}$$

Si può dimostrare che l'operazione di convoluzione corrisponde al seguente prodotto nel dominio delle frequenze

$$Y(\omega) = X(\omega)H(\omega). \tag{4.12}$$

Questo risultato è di grande importanza poichè come vedremo successivamente sfruttando un algoritmo per la trasformata di Fourier veloce, è possibile ottenere una convoluzione e quindi un filtraggio con una complessità di tempo minore.

Teorema di modulazione

Il "Teorema di modulazione" può essere considerato l'inverso del teorema di convoluzione. In altre parole la DTFT del prodotto di sequenze corrisponde alla convoluzione periodica delle DTFT corrispondenti. Supponiamo di avere una sequenza discreta $x(n)$ e la sua trasformata $X(\omega)$ e una risposta all'impulso $w(n)$ con trasformata $W(\omega)$ allora il prodotto delle sequenze è espresso da

$$y(n) = x(n)w(n). \tag{4.13}$$

Si può dimostrare che questo risultato corrisponde ad una convoluzione nel dominio delle frequenze

$$Y(\omega) = \frac{1}{2\pi}\int_{-\pi}^{\pi} X(\theta)W(\omega-\theta)d\theta \tag{4.14}$$

4.3.4 Teorema di Parseval

Il "Teorema di Parseval" permette di introdurre un concetto fondamentale che è legato allo spettro della trasformata di Fourier. Supponiamo di avere una sequenza discreta $x(n)$ e la sua trasformata $X(\omega)$. L'*energia* di un segnale può essere espressa come

$$E = \sum_{k=-\infty}^{\infty} |x(n)|^2 = \frac{1}{2\pi}\int_{-\pi}^{\pi} |X(\theta)|^2 d\omega \tag{4.15}$$

Dove la funzione $|X(\theta)|^2$ è chiamata *densità spettrale di energia*.

4.4 Trasformata z

Mentre nei sistemi a tempo continuo la trasformata di Laplace può essere considerata una generalizzazione della trasformata di Fourier, nei sistemi a tempo discreto la Trasformata z può essere considerata una generalizzazione della Trasformata di Fourier per segnali a tempo discreto (DTFT). Risulta quindi che la Trasformata z è un caso di trasformazione di dominio più ampio di quello della DTFT.

4.4.1 Caratteristiche

La Trasformata z (*bilatera*) di una sequenza $x(n)$ è definita come

$$X(z) = \sum_{n=-\infty}^{\infty} x(n)z^{-n} \tag{4.16}$$

dove z è una variabile complessa. Spesso si ricorre alla Trasformata z *unilatera destra*

$$X(z) = \sum_{n=0}^{\infty} x(n)z^{-n} \tag{4.17}$$

o *unilatera sinistra*

$$X(z) = \sum_{n=-\infty}^{0} x(n)z^{-n}. \tag{4.18}$$

Notiamo che esprimendo la variabile complessa z in forma polare e quindi come $z = re^{j\omega}$ otteniamo

$$X(re^{j\omega}) = \sum_{n=-\infty}^{\infty} x(n)r^{-n}e^{-j\omega n} \tag{4.19}$$

Osserviamo che per $r = 1$ e cioè per $|z| = 1$ la Trasformata z coincide con la trasformata di Fourier a tempo discreto della sequenza. Notiamo inoltre che data la moltiplicazione per l'esponente reale r^{-n} è possibile che la Trasformata z converga anche se non converge la trasformata di Fourier.

4.4.2 Regione di Convergenza

Per ogni sequenza assegnata l'insieme dei valori di z per cui la Trasformata z converge si chiama *Regione di Convergenza* (Region of Convergence - ROC). La convergenza uniforme richiede che la sequenza sia assolutamente sommabile e quindi

$$\sum_{k=-\infty}^{\infty} |r^{-n}e^{-j\omega n}| < \infty. \tag{4.20}$$

In generale la serie di potenze della Trasformata z convergerà in una regione anulare del piano z (vedi Figura 4.3) indentificata da

$$R_{x^-} < |z| < R_{x^+} \tag{4.21}$$

dove R_{x^-} può essere piccolo fino ad annullarsi ed R_{x^+} può essere grande fino all'infinito. La ROC corrisponde ad un cerchio nel piano z centrato nell'origine. In altre parole, il confine

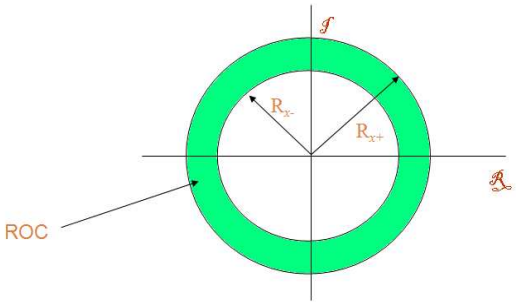


Figura 4.3: Regione di Convergenza della Trasformata z.

più esterno sarà un cerchio o si può estendere fino all'infinito. Il confine più interno sarà un cerchio e si può estendere fino a diventare l'origine. Se la ROC include il cerchio unitario la convergenza della Trasformata z implica che anche la trasformata di Fourier converge. Se la ROC non include il cerchio unitario la trasformata di Fourier non è assolutamente convergente.

4.4.3 Proprietà di Convergenza

Le proprietà della sequenza $x(n)$ determinano la regione di convergenza di $X(z)$. Possiamo elencarle in questo modo

- la ROC è un anello o un disco nel piano z centrato nell'origine
- La trasformata di Fourier di $x(n)$ converge assolutamente se e solo se la ROC della Trasformata z di $x(n)$ comprende il cerchio unitario
- La ROC non può contenere alcun polo ed è limitata da poli o da zeri o da infinito
- Se $x(n)$ è una sequenza di durata finita la ROC è l'intero piano z eccetto possibili $z = 0$ e $z = \infty$
- se $x(n)$ è monolatera destra la ROC è l'esterno di un cerchio (polo con ampiezza maggiore fino a (possibilmente ∞). Inoltre se tale regione include $z = \infty$, la sequenza è causale
- se $x(n)$ è monolatera sinistra la ROC è l'interno di un cerchio (polo diverso da zero con ampiezza minore fino (possibilmente) a 0)

- Se $x(n)$ è bilatera la ROC consiste di un anello nel piano z limitato dall'interno e dall'esterno da un polo ed in accordo con la proprietà 3 non contiene nessun polo
- la ROC deve essere una regione connessa.

4.4.4 Zeri e poli

Una classe importante di Trasformate z è quella per cui $X(z)$ è una funzione razionale e cioè un rapporto di polinomi in z

$$X(z) = \frac{P(z)}{Q(z)} \tag{4.22}$$

Per il polinomio numeratore le radici sono quei valori di z per cui $X(z) = 0$ e sono chiamati gli *zeri* di $X(z)$. Per il polinomio denominatore le radici per valori finiti di z sono chiamati *poli* di $X(z)$ e cioè i valori di z per cui $X(z)$ è infinita. Possono aversi poli anche in $z = 0$ e in $z = \infty$.

Esempio

Per chiarire il concetto di zeri e poli consideriamo la seguente sequenza

$$x(n) = a^n u(n). \tag{4.23}$$

La trasformata z è data da

$$X(z) = \sum_{n=-\infty}^{\infty} a^n u(n) z^{-n} = \sum_{n=-\infty}^{\infty} (az^{-1})^n \tag{4.24}$$

che converge a

$$X(z) = \frac{1}{1 - az^{-1}} = \frac{z}{z - a} \tag{4.25}$$

per $|z| \geq |a|$. $X(z)$ ha uno zero in $z = 0$ e un polo in $z = a$ e la ROC include l'intero piano z per $|z| > a$. La ROC corrispondente è raffigurata in Figura 4.4

4.4.5 Trasfromata z inversa

Ci sono diversi modi formali ed informali per ottenere la Trasformata z inversa. La versione formale è basata sul teorema dell'integrale di Cauchy (che non vedremo). Procedure meno formali sono preferibili. I metodi meno formali possono essere

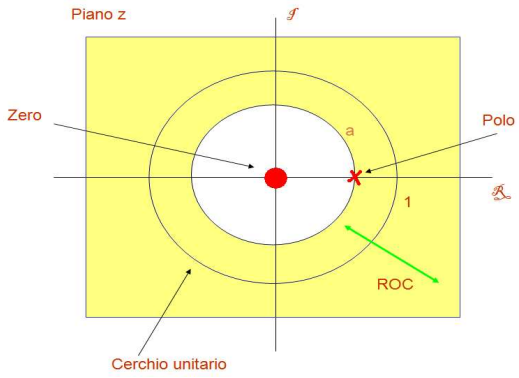


Figura 4.4: ROC della trasformata z.

- *Ispezione* - richiede di diventare familiari con alcune trasformate libreria di trasformate note. Osservata una sequenza possiamo fare la trasformata ispezionando e riconoscendo la trasformata adatta. Nuove trasformate si possono ottenere applicando le proprietà della Trasformata z.

- *Serie di Potenze* - Trasformata z in forma di serie di potenze. Data una serie si può ricavare la corrispondente serie di potenze o rifarsi a un'espressione in serie di potenze precedentemente ricavata.

- *Espansione in fratti semplici* - Trasformate z razionali. Si effettua un'espansione in fratti semplici e si identifica la Trasformata z inversa dei termini più facilmente trattabili.

4.4.6 Proprietà

Come già fatto con la DTFT introduciamo ora alcune proprietà della Trasformata z. Nel seguito denotiamo con $X(z)$ la Trasformata z della sequenza discreta $x(n)$ e la ROC definita nell'intervallo $[R_{x^-}, R_{x^+}]$.

Linearità

Supponiamo di avere due sequenze $x_1(n)$ e $x_2(n)$ e le rispettive Trasformate z $X_1(z)$ con $ROC_1 = R_{x_1}$ e $X_2(z)$ con $ROC_2 = R_{x_2}$. Abbiamo quindi che la Trasformata z della mistura delle due sequenze $ax_1(n) + bx_2(n)$ risulta come la mistura delle singole

Trasformate z e cioè $aX_1(z)+bX_2(z)$ con $ROC = R_{x_1} \cap R_{x_2}$. Questa proprietà è rispettata anche dalla trasformta DTFT.

Traslazione

Supponiamo di avere una sequenza $x(n)$ e la rispettiva Trasformata z $X(z)$ con $ROC = R_x$, allora $x(n - n_0)$ corrisponde nel domino z a $z^{-n_0}X(z)$ con $ROC = R_x$. Anche in questo caso la proprietà è rispettata dalla trasformta DTFT.

Convoluzione

Supponiamo di avere due sequenze $x_1(n)$ e $x_2(n)$ e le rispettive Trasformate z $X_1(z)$ con $ROC_1 = R_{x_1}$ e $X_2(z)$ con $ROC_2 = R_{x_2}$. Abbiamo quindi che la Trasformata z della convoluzione delle due sequenze $x_1(n) \otimes x_2(n)$ risulta il prodotto delle singole trasformate z e cioè $X_1(z)X_2(z)$ con $ROC = R_{x_1} \cap R_{x_2}$.

Prodotto

Nel caso di segnali a tempo continuo e delle trasformate di Fourier esiste una dualità tra il dominio del tempo e quello della frequenza per quanto riguarda prodotto e convoluzione. Nel caso della Trasformata z il prodotto di sequenze ha una forma "simile" alla convoluzione (non esiste dualità).

Relazione di Parseval

Supponiamo di avere due sequenze $x_1(n)$ e $x_2(n)$ e le rispettive Trasformate z $X_1(z)$ con $ROC_1 = R_{x_1}$ e $X_2(z)$ con $ROC_2 = R_{x_2}$. Sussiste la seguente relazione

$$\sum_{n=-\infty}^{\infty} x_1(n)x_2^*(n) = \frac{1}{2\pi}\int_{-\pi}^{\pi} X_1(z)X_2^*(z)dz \tag{4.26}$$

4.4.7 Trasformata z e funzione di trasferimento

Introduciamo ora un altro risultato fondamentale che ci aiuterà a descrivere i filtri e a caratterizzare gli effetti sonori. Se un sistema è descrivibile con un'equazione alle differenze lineare a coefficienti costanti, la funzione di trasferimento è esprimibile in termini di un rapporto tra polinomi. Questo fondamentale risultato ci permette di definire un sistema

(e.g. filtri IIR e FIR) in termini di caratteristiche geometriche. Per dimostrare questo risultato partiamo dall'equazione alle differenze in forma generale

$$\sum_{k=0}^{N} a_k y(n-k) = \sum_{r=0}^{M} b_r x(n-r) \tag{4.27}$$

Applicando la Trasformata z, otteniamo

$$\mathbf{Z}\left[\sum_{k=0}^{N} a_k y(n-k)\right] = \mathbf{Z}\left[\sum_{r=0}^{M} b_r x(n-r)\right] \tag{4.28}$$

che in base alla propietà di lineartà diventa

$$\sum_{k=0}^{N} a_k \mathbf{Z}\left[y(n-k)\right] = \sum_{r=0}^{M} b_r \mathbf{Z}\left[x(n-r)\right] \tag{4.29}$$

inoltre, sapendo che

$$\mathbf{Z}\left[y(n-k)\right] = z^{-k} Y(z) \tag{4.30}$$

$$\mathbf{Z}\left[x(n-k)\right] = z^{-k} X(z) \tag{4.31}$$

otteniamo

$$\left(\sum_{k=0}^{N} a_k z^{-k}\right) Y(z) = \left(\sum_{r=0}^{M} b_r z^{-k}\right) X(z) \tag{4.32}$$

e in definitiva poniamo

$$H(z) = \frac{Y(z)}{X(z)} = \frac{\sum_{r=0}^{M} b_r z^{-k}}{\sum_{k=0}^{N} a_k z^{-k}} \tag{4.33}$$

La caratteristica dei sistemi descrivibili con equazioni lineari alle differenze a coefficienti costanti è quella che le loro funzioni di trasferimento sono rapporti di polinomi in z^{-1}. La funzione di trasferimento può essere descritta da un diagramma di zeri e poli nel piano z. Per un dato rapporto di polinomi, ogni possibile scelta per la regione di convergenza conduce a una risposta all'impulso diversa. Se si assume che il sistema sia stabile, occorre scegliere la regione anulare che comprende il circolo unitario.

Se il sistema è stabile, tutti i poli cadono all'interno del cerchio unitario e la regione di convergenza includerà il circolo unitario. Nel caso particolare in cui $N = 0$ il sistema non ha poli eccetto per $z = 0$ ed ha una risposta all'impulso di durata finita. Quando N è maggiore di zero, il sistema ha poli, ciascuno dei quali contribuisce con una sequenza esponenziale alla risposta al campione unitario. Se la funzione di trasferimento ha poli, la risposta all'impulso è di durata infinita. Vantaggi della funzione di trasferimento in termini

di poli e zeri conduce a un metodo geometrico per ricavare la risposta in frequenza del sistema. Per valutare la funzione di trasferimento sul circolo unitario, in corrispondenza di una frequenza di eccitazione ω_0 occorre sostituire $z = e^{j\omega_0}$ nell'equazione.

4.5 Trasformata di Fourier discreta

Come abbiamo visto nella sessione precedente, la Trasformata z e quindi la corrispondente trasformata DTFT permettono di analizzare sequenze discrete ma di durata infinita. Lo scopo che vogliamo raggiungere ora è quello di introdurre una trasformata, la *Trasformata di Fourier Discreta* (DFT), che permette di analizzare sequenze di durata finita. Vediamo come costruire questa trasformata partendo dalla sua relazione con la Trasformata z.

4.5.1 Trasformata z ed esponenziale complesso

Consideriamo una sequenza discreta $x(n)$ di lunghezza N tale che $x(n) = 0$ per valori fuori all'intervallo $0 \leq n \leq N-1$. La Trasformata z di $x(n)$ è data da

$$X(z) = \sum_{n=-\infty}^{+\infty} x(n)z^{-n} = \sum_{n=0}^{N-1} x(n)z^{-n}. \tag{4.34}$$

Come abbiamo mostrato precedentemente considerando $z = e^{j\omega}$ otteniamo la Trasformata DTFT che è definita per tutte le frequenze ω su un cerchio di raggio unitario. La DTFT risulta infatti

$$X(\omega) = \sum_{n=-\infty}^{+\infty} x(n)e^{-j\omega n} = \sum_{n=0}^{N-1} x(n)e^{-j\omega n}. \tag{4.35}$$

Per ottenere un numero finito di frequenze sul cerchio unitario vengono scelte N frequenze da 0 a 2π con passo corrispondente a $\frac{2\pi}{N}$ come è mostrato in Figura 4.5. In questo modo l'esponenziale $e^{-j\omega}$ della DTFT viene denotato con W_N e risulta

$$W_N = e^{-j\frac{2\pi}{N}} \tag{4.36}$$

ed inoltre denotiamo W_N^k come

$$W_N^k = e^{-j\frac{2\pi}{N}k} \tag{4.37}$$

In Figura 4.6 vengono rappresentate le $N = 8$ radici dell'unità.

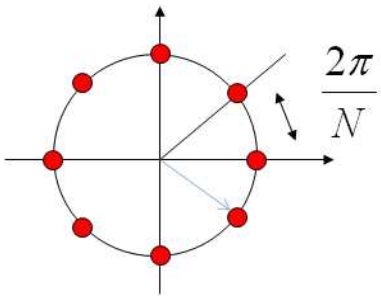


Figura 4.5: DTFT e cerchio unitario.

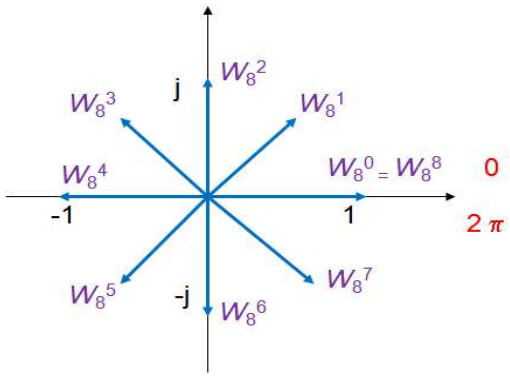


Figura 4.6: $N = 8$ radici del cerchio unitario.

DFT				
#	name	:	string	nome della sequenza
#	X	:	vector<Type>	dominio frequenze
#	xn	:	vector<Type>	dominio del tempo
+	DFT (vector<Type>)			costruttore
+	dft()	:	void	trasformata
+	idft()	:	void	antitrasformata
+	getxnr()	:	vector<double>	parte reale xn
+	getxni()	:	vector<double>	parte immaginaria xn
+	getXr()	:	vector<double>	parte reale X
+	getXi()	:	vector<double>	parte immaginaria X

Tabella 4.3: Classe **DFT**.

4.5.2 Trasformata di Fourier discreta

Da quanto detto precedentemente possiamo concludere che la fase di analisi della Trasformata di Fourier discreta (DFT) risulta

$$X(k) = \begin{cases} \sum_{n=0}^{N-1} x(n) W_N^{kn} & 0 \leq k \leq N-1 \\ 0 & \text{altrove} \end{cases} \tag{4.38}$$

invece la fase di sintesi risulta

$$x(n) = \begin{cases} \sum_{k=0}^{N-1} X(k) W_N^{-kn} & 0 \leq n \leq N-1 \\ 0 & \text{altrove} \end{cases} \tag{4.39}$$

Nella Tabella 4.5.2 viene presentato lo schema della classe **DFT** e l'implementazione è la seguente

```
template <class Type> class DFT : public  sequenze<Type>{ public:
DFT(vector<Type> vIn) : sequenze<Type>(vIn){};


void dft() {};


void idft() {};
 . . .


private: string name;


// componenti di Fourier


vector<Type> X;
```

```
// Sequeza stimata


vector<Type> xn;
};
```

Essa estende la classe **sequenze**. Ha due metodi che permettono di effettuare la trasformata DFT diretta (**dft**)e inversa (**idft**) come è riportato nel sguente codice C++

```
void dft() {
    static vector<complex> cf;
    static int nstore = 0;
    if( x.empty() )
    {cf = vector<complex>(size);
     nstore = 0;}
    else{
    // Crea il vettore di Fourier
    X = vector<complex>(size);
    // Verifica se la lunghezza è cambiata
    if( size != nstore )
    {   cf = vector<complex>(size);
        nstore = size;
        double arg =  8.0 * atan( 1.0 ) / size;
        for( int i = 0; i < size; i++ ) cf[i].put(cos( arg *
            i), - sin( arg * i));}
    vector<complex> dataIn(size);
    // Calcolo della DFT
    for( int k = 0; k < size; k++ )
    {   dataIn = x;
        X[k] = dataIn[0];
        for( int n = 1; n < size; n++ ) {
            int p = ((long)(n * k) % size );
            complex cfm = cf[p];
            X[k] += cfm * dataIn[n];
        }
    }
  }
}

void idft() {   static vector<complex> cf;
    static int nstore = 0;
```

```
    if( x.empty() )
    {
        cf = vector<complex>(size);
        nstore = 0;
    }
    else {
     xn = vector<complex>(size);
    if( size != nstore )
    {
        cf = vector<complex>(size);
        nstore = size;
        double arg = 8.0 * atan( 1.0 ) / size;
        for( int i = 0; i < size; i++ )
        cf[i].put(cos(arg * i)/size , sin(arg * i)/size );
    }
vector<complex> dataIn(size);
    // Calcolo IDFT
    for( int k = 0; k < size; k++ )
    {
        dataIn = X;
        xn[k].prod(dataIn[0],cf[0]);
        for( int n = 1; n < size; n++ )
        {
            int p = ((n * k )  %  size);
            complex cfm = cf[p];
            xn[k] += cfm * dataIn[n];
        }
    }
  }
};
```

4.5.3 Esempio

A questo punto mostriamo un esempio di utilizzo della Trasformata DFT. Consideriamo la seguente sequenza di 4 punti

$$x(n) = \begin{cases} 1 & 0 \leq n \leq 3 \\ 0 & \text{altrove} \end{cases} \tag{4.40}$$

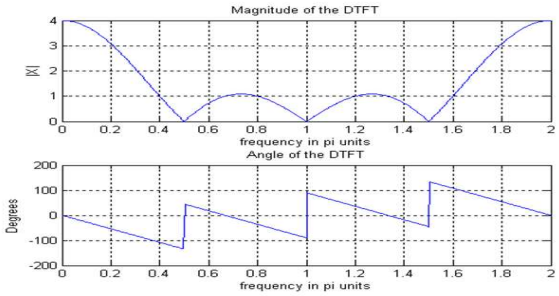


Figura 4.7: DTFT della sequenza $x(n)$.

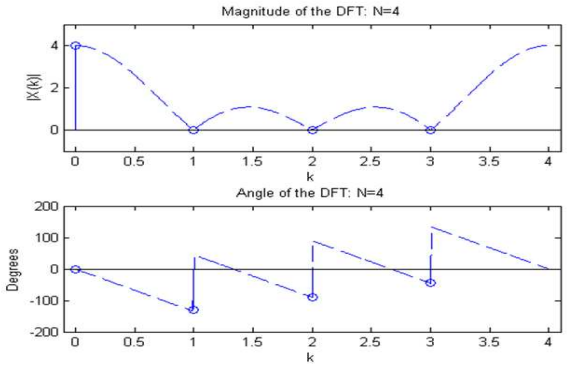


Figura 4.8: DFT della sequenza $x(n)$.

Nella Figura 4.7 vengono rappresentati il modulo e la fase della DTFT relativa alla sequenza $x(n)$. Nella Figura 4.8, invece, viene rappresentato il modulo e la fase della DFT sulla stessa sequenza.

Dal grafico si evince che la DFT della sequenza di 4 punti risulta un campionamento della corrispondente DTFT. Consideriamo ora un numero di campioni maggiore e cioè poniamo $N = 8$. Questo può essere fatto aggiungendo 4 zero alla fine della sequenza

$$x(n) = [1\ 1\ 1\ 1\ 0\ 0\ 0\ 0] \tag{4.41}$$

L'operazione è chiamata riempimento di zero ($zero-padding$) ed è necessaria per ottenere un spettro di potenza più denso. Questo può essere chiarito dando uno sguardo alle Figure 4.9 e 4.10 dove vengono considerati un numeri crescente di zeri. Come abbiamo appena visto il riempimento di zero aumenta la capacità risolutiva della DFT e rende lo spettro più denso. Per aumentare la capacità risolutiva dello sppettro, comunque, dobbiamo

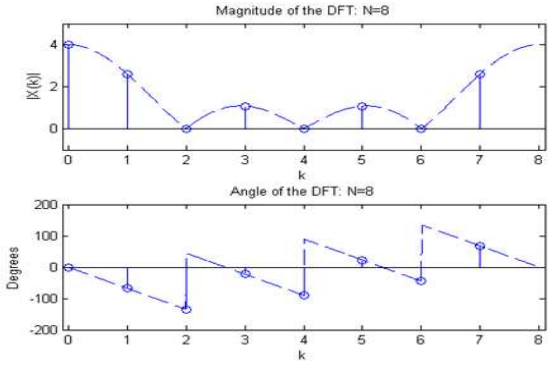


Figura 4.9: Zero-padding con 8 punti.

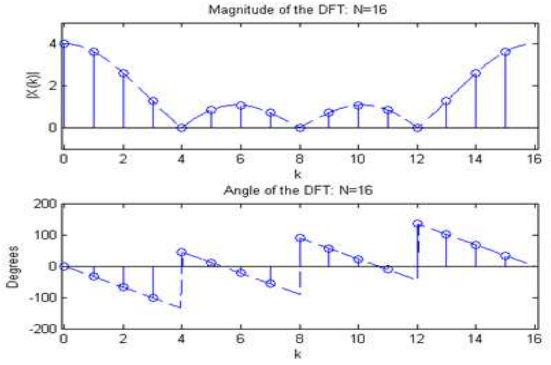


Figura 4.10: Zero-padding con 16 punti.

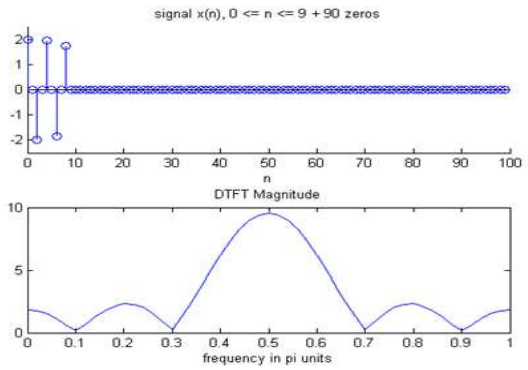


Figura 4.11: Sequenza di equazione 4.42 con 90 zero di riempimento.

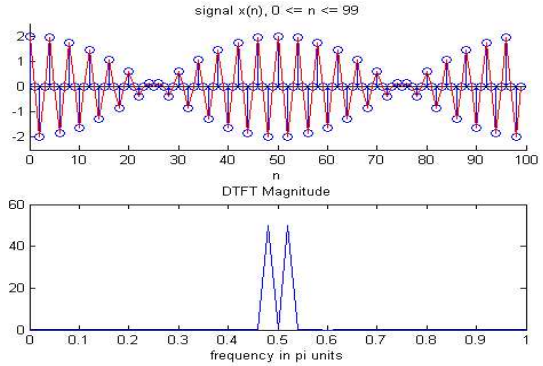


Figura 4.12: Sequenza di equazione 4.42 con 100 punti della sorgente originale.

considerare un numero maggiore di campioni della sequenza originale. Come esempio consideriamo il seguente segnale sinusoidale con frequenze rispettivamente di 0.48 e di 0.52

$$x(n) = \cos(0.48\pi n) + \cos(0.52\pi n) \tag{4.42}$$

In Figura 4.11 viene visualizzato lo spettro ottenuto considerando 10 punti della sequenza e 90 zero di riempimento. In Figura 4.12, invece, viene rappresentato lo spettro della sequenza in cui si considerano 100 campioni della sequenza originale. Possiamo sicuramente notare che lo spettro risulta ad alta risoluzione ed identifica le due componenti frequenziali del segnale.

Nel seguente codice viene presentato un esempio per il calcolo dello spettro di una sequenza tramite DTFT

```
T.dft();
    vector<complex> X = T.get_X();
    vector<double> w((int)n_coeff/2);
    vector<double> y((int)n_coeff/2);
   for(int i=0;i<(int)n_coeff/2;i++)
    {
     y[i] =  pow(X[i].cabs(),2)/T.massimo();
     w[i] = T.get_w(i) * fc;  // in frequenze e non pulsazioni
    }
```

4.5.4 Proprietà

Diamo ora uno sguardo alle proprietà della DFT come precedentemente fatto per la DTFT e la Trasformata z. In seguito definiamo $X(k)$ come la DFT della sequenza discreta $x(n)$.

Linearità

Supponiamo di avere due sequenze $x_1(n)$ e $x_2(n)$ e le rispettive trasformate DFT $X_1(k)$ e $X_2(k)$. Abbiamo quindi che la trasformata DFT della mistura delle due sequenze $ax_1(n)+bx_2(n)$ risulta alla mistura delle singole Trasformate z e cioè $aX_1(k)+bX_2(k)$.

Traslazione (circolare)

Immaginiamo la sequenza di durata finita $x(n)$ disposta lungo la circonferenza di un cilindro in modo che il cilindro abbia una circonferenza di esattamente N punti. Se si percorre più volte la circonferenza del cilindro, la sequenza che si vede è proprio la sequenza periodica di periodo N, di cui $x(n)$ è un periodo. Essa sarà indicata come

$$\widetilde{x}(n) = x(n\%N) \tag{4.43}$$

dove % è l'operazione di modulo. La sequenza di durata finita $x(n)$ si ricava estraendo un periodo

$$x(n) = \begin{cases} \widetilde{x}(n) & 0 \leq n \leq N-1 \\ 0 & \text{altrove} \end{cases} \tag{4.44}$$

E' utile introdurre la sequenza rettangolare $R_N(n)$ definita come

$$R_N(n) = \begin{cases} 1 & 0 \leq n \leq N-1 \\ 0 & \text{altrove} \end{cases} \tag{4.45}$$

La relazione precedente può essere quindi espressa come

$$x(n) = \widetilde{x}(n)R_N(n) \tag{4.46}$$

Sostanzialmente una traslazione lineare della sequenza periodica corrisponde a una rotazione del cilindro. Dalla DFT abbiamo quindi che le due realazioni di traslazione sono

$$x(n + m\%N)R_N(n) = W_N^{-km}X(k) \tag{4.47}$$

$$W_N^{rn}x(n) = X(k + r\%N)R_N(k) \tag{4.48}$$

Convoluzione (circolare)

Supponiamo di avere due sequenze $x_1(n)$ e $x_2(n)$ e le rispettive trasformate $X_1(k)$ e $X_2(k)$. Abbiamo quindi che la trasformata DFT della convoluzione circolare delle due sequenze $x_1(n) \otimes x_2(n) =$

$$\left[\sum_{m=0}^{N-1} x_1(m\%N)x_2(n - m\%N)\right] R_N(n) \tag{4.49}$$

risulta il prodotto delle singole trasformate DFT e cioè $X_1(k)X_2(k)$.

Convoluzione lineare

Esistono algoritmi efficienti per calcolare la DFT (come vedremo successivamente). Può convenire, dal punto di vista computazionale, effettuare la convoluzione di due sequenze calcolando la loro DFT, farne il prodotto e calcolare la DFT inversa. Nella maggior parte delle applicazioni interessa eseguire la convoluzione lineare di due sequenze (e.g. filtraggio voce o segnali radar). Consideriamo due sequenze di lunghezza N, $x_1(n)$ e $x_2(n)$ ed indichiamo con $x_3(n)$ la convoluzione lineare

$$x_3(n) = \sum_{m=0}^{N-1} x_1(m)x_2(n - m) \tag{4.50}$$

Si può subito verificare che $x_3(n)$ ha lunghezza $2N-1$. Come vedremo la stessa operazione può essere fatta con un prodotto nel dominio delle frequenze portando ad una complessità in tempo minore.

Prodotto

Come anche nel caso della trasformata DTFT, nel caso in cui consideriamo un prodotto di due sequenze $x_1(n)$ e $x_2(n)$, questo risulta una convoluzione circolare nel dominio delle frequenze

$$x_1(n)x_2(n) = \frac{1}{N}\left[\sum_{l=0}^{N-1} X_1(l\%N)X_2(k-l\%N)\right] R_N(k) \tag{4.51}$$

La convoluzione circolare è del tutto equivalente a costruire prima due sequenze periodiche e poi farne la convoluzione. Per due sequenze finite $x_1(n)$ e $x_2(n)$ di lunghezza N la convoluzione ha dimensione $2N-1$. Siccome la convoluzione circolare ha lunghezza N è affetta da aliasing.

Relazione di Parseval

Supponiamo di avere una sequenza $x(n)$ e la rispettiva trasformata DFT $X(k)$. Sussiste la seguente relazione tra l'energie delle sequenze

$$\sum_{n=0}^{N-1} |x(n)|^2 = \sum_{k=0}^{N-1} |X(k)|^2 \tag{4.52}$$

dove $|\frac{X(k)}{N}|^2$ è chiamato lo *spettro di potenza*.

4.6 Stima Spettrale

La *stima spettrale* è una diretta conseguenza delle trasformate appena introdotte ed è un meccanismo per studiare il contenuto frequenziale di un segnale audio discreto. Considerando la trasformata di Fourier a tempo discreto di una sequenza di lunghezza finita $x(n)$

$$X(\omega) = \sum_{n=0}^{N-1} x(n)e^{-jwn} \tag{4.53}$$

possiamo definire la stima dello spettro, chiamato *Periodogramma*, come

$$I_N(\omega) = \frac{1}{N}|X(\omega)|^2 \tag{4.54}$$

Il Periodogramma non è una stima "consistente" dello spettro e il suo comportamento al crescere di N è del tutto insoddisfacente. Occorre studiare delle modifiche che diano risultati migliori.

4.6.1 Metodo di Bartlett

Un metodo classico per ridurre la varianza delle stime è quello di fare la media di numerose stime indipendenti. L'applicazione di questo concetto alla stima dello spettro è comunemente attribuita a Bartlett. Una sequenza dati $x(n)$, viene suddivisa in K segmenti di M campioni ciascuno ($N = KM$)

$$x^{(i)}(n) = x(n + iM - M) \tag{4.55}$$

per $0 \leq n \leq M-1$ e $0 \leq i \leq K$ si calcolano i K periodogrammi

$$I_M^i(\omega) = \frac{1}{M} |\sum_{n=0}^{M-1} x^{(i)}(n) e^{-jwn}|^2 \tag{4.56}$$

e la stima dello spettro è definita come

$$P(\omega) = \frac{1}{K} \sum_{i=1}^{K} I_M^i(\omega) \tag{4.57}$$

Possiamo notare che la varianza diminuisce al crescere del numero dei Periodogrammi e quindi diminuisce anche M e quindi la risoluzione dello spettro. Occorre raggiungere un compromesso tra risoluzione dello spettro da un lato e varianza della stima dall'altro. La scelta effettiva di M ed N per la stima dello spettro in una situazione reale è orientata in generale dalle conoscenze a priori sul segnale da analizzare. Se sappiamo che lo spettro ha un picco molto stretto, e se è importante risolverlo, dobbiamo scegliere M abbastanza grande per ottenere la risoluzione in frequenza desiderata.

4.6.2 Metodo di Welch

Welch ha introdotto una modifica al metodo di Bartlett. La sequenza dati è suddivisa in $K = N/M$ sottosequenze di M campioni. La finestra $w(n)$ viene applicata direttamente alle sottosequenze dei dati, prima del calcolo del Periodogramma

$$J_M^i(\omega) = \frac{1}{MU} |\sum_{n=0}^{M-1} x^{(i)}(n) w(n) e^{-jwn}|^2 \tag{4.58}$$

dove

$$U = \frac{1}{M} \sum_{n=0}^{M-1} w^2(n) \tag{4.59}$$

e il Periodogramma diventa

$$P(\omega) = \frac{1}{K} \sum_{i=1}^{K} J_M^i(\omega) \tag{4.60}$$

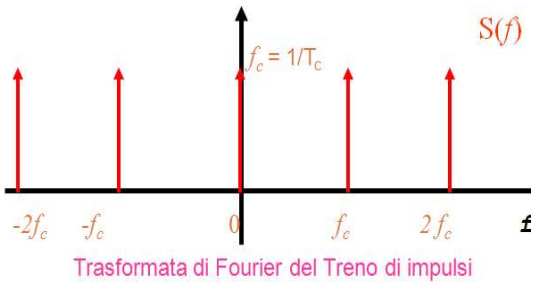


Figura 4.13: Treno di impulsi $s(t)$ nel dominio delle frequenze $S(f)$.

4.6.3 Teorema di Campionamento - dominio delle frequenze

Come abbiamo già visto nel Capitolo 2 per ottenere una sequenza discreta $x_c(t)$ da un segnale continuo $x(t)$ si usa la modulazione ad impulsi (o PCM) (vedi Figura 2.7, Capitolo 2). Dal punto di vista matematico il campionamento è il prodotto tra il segnale continuo $x(t)$ da campionare e una funzione di campionamento $s(t)$ rappresentata da una sequenza periodica di impulsi $\delta(t)$ (vedi Figura 2.8, Capitolo 2).
Come è mostrato in Figura 4.13 la Trasformata di Fourier di un treno di impulsi è a sua volta un treno di impulsi. Risulta, infatti, che dato un treno di impulsi di ampiezza unitaria in cui il periodo di campionamento è T_c, la sua Trasformata di Fourier risulta un treno di impulsi in cui l'ampiezza e il periodo sono proporzionali ad $f_c = 1/T_c$. Applichiamo a questo punto la trasformata di Fourier sia al segnale continuo che al treno di impulsi. Indichiamo con $X(f)$ la trasformata di Fourier del segnale continuo $x(t)$ e con $S(f)$ la trasformata di Fourier del treno di impulsi $s(t)$, abbiamo

$$x_c(t) = x(t)s(t) \Rightarrow X_c(f) = X(t) \otimes S(f) \tag{4.61}$$

dove $\otimes$ è la convoluzione. In Fiura 4.14 vengono rappresentate le sequenze $x(t)$ e $x_c(t)$ nel dominio delle frequenze. Osserviamo che la trasformata di Fourier $X(f)$ del segnale $x(t)$ ha una banda limitata B. Inoltre la Trasformata di Fourier $X_c(f)$ del segnale campionato $x_c(t)$ è una ripetizione infinita della banda di frequenza B del segnale originale intorno alla frequenza di campionamento f_c (per effetto della convoluzione con $S(f)$). In dettaglio abbiamo che essendo il treno di impulsi $s(t)$ pari a

$$s(t) = T \sum_{k=-\infty}^{\infty} \delta(t - kT_c) \tag{4.62}$$

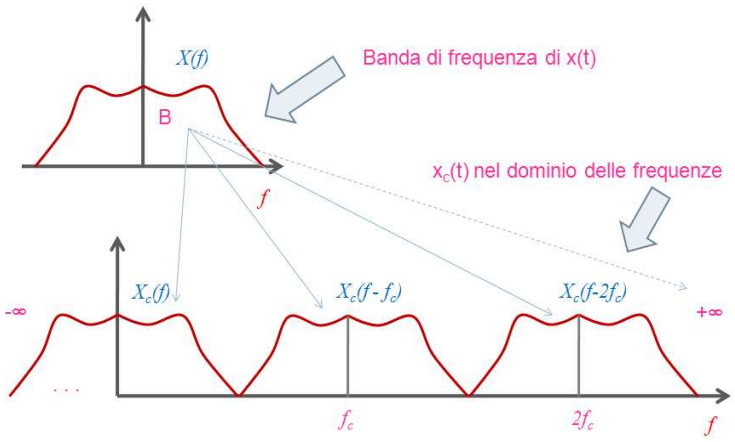


Figura 4.14: La sequenza $x(t)$ e $x_c(t)$ nel dominio delle frequenze.

implica che la sua trasformata di Fourier diventa

$$S(f) = \sum_{k=-\infty}^{\infty} \delta(f - k\frac{1}{T_c}) = \sum_{k=-\infty}^{\infty} \delta(f - kf_c) \tag{4.63}$$

e quindi la trasformata di Fourier del segnale campionato diventa

$$X_c(f) = X(f) \otimes S(f) = \sum_{k=-\infty}^{\infty} X(f - kf_c) \tag{4.64}$$

o in altre parole la convoluzione delle due trasformate di Fourier.

4.6.4 Teorema di Campionamento

Nel Capitolo 2 abbiamo introdotto il Teorema di Campionamento e abbiamo descritto come esso influenza la scelta della frequenza di campionamento nella fase di digitalizzazione del segnale. Esso afferma che un segnale a tempo continuo $x(t)$ con una banda spettrale B strettamente limitata ($X(f) = 0$ per $|f| \geq B$), può essere univocamente ricostruito dalla sua versione campionata $x(n) = x(nT_c)$ ($n = 0, \pm 1, \pm 2, \pm 3, \ldots$) se la frequenza di campionamento $f_c = 1/T_c$ soddisfa la seguente relazione

$$f_c = \frac{1}{T_c} \geq 2B \tag{4.65}$$

Per dimostrare graficamente il significato del teorema di campionamento consideriamo il

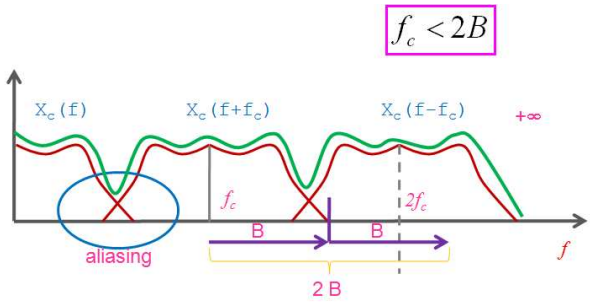


Figura 4.15: Teorema di Campionamento: aliasing

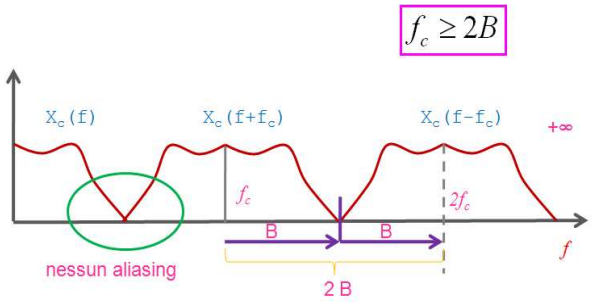


Figura 4.16: Teorema di Campionamento: senza aliasing.

segnale continuo precedente con banda limitata B. Nel caso in cui la frequenza di campionamento è minore di $2B$ abbiamo che le ripetizioni periodiche dello spettro del segnale campionato si sovrappongono. Producono in questo caso *aliasing* e cioè alcune frequenze basse si sovrappongono a quelle alte (vedi Figura 4.15). Nel caso in cui la frequenza di campionamento f_c è maggiore di $2B$ non abbiamo sovrapposizione e soddisfacendo il Teorema di Campionamento non abbiamo aliasing (vedi Figura 4.16). Questo ci garantisce il recupero tramite filtraggio dell'intera banda di frequenze B e quindi della perfetta ricostruzione del segnale analogico. Infatti, dato il segnale $x(t)$ a banda B strettamente limitata ($X(f) = 0$ per $|f| \geq B$), e la condizione di Nyquist $f_c = 1/Tc \geq 2B$ è soddisfatta, allora il segnale originale $x(t)$ può essere ricostruito a partire dalla sua versione campionata $x(n)$ con un filtro passa-basso ideale avente guadagno T_c e frequenza di taglio f_p tale che

$$B < f_p < f_c - B \tag{4.66}$$

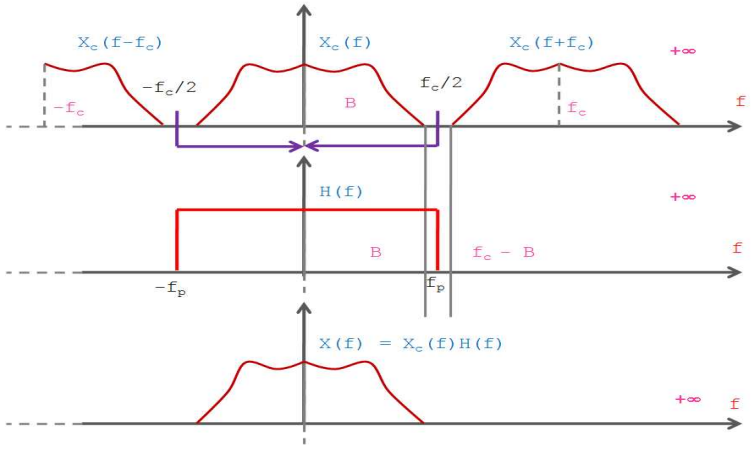


Figura 4.17: Ricostruzione del segnale.

4.6.5 Ricostruzione del segnale

Come abbiamo precisato precedentemente il segnale continuo $x(t)$ può essere riscostruito dal segnale campionato $x_c(t)$. Per ottenere ciò la frequenza di campionamento f_c deve essere scelta in modo da non produrre aliasing e quindi in modo che soddisfa il Teorema di Nyquist. A questo punto possiamo riottenere la banda di frequenze B del segnale originale analogico $x(t)$ applicando un filtro passa-basso ideale H come mostrato in Figura 4.17. Dagli schemi in Figura 4.18 possiamo anche notare che il segnale continuo può essere ottenuto effetuando sia un filtraggio nel dominio del tempo che nel dominio delle frequenze. Infatti, dalla relazione secondo cui un prodotto nel dominio delle frequenze corrisponde ad una convoluzione nel dominio del tempo possiamo ottenere il segnale originale dalla convoluzione della sequenza con la risposta all'impulso del filtro (questo concetto sarà chiarito nei prossimi Capitoli). In dettaglio abbiamo che se consideriamo la trasformata inversa della risposta in frequenza di un filtro passa-basso TF^{-1} otteniamo la seguente risposta all'impulso

$$h(t) = TF^{-1}\left(H(f)\right) = \text{sinc}(tf_c) = \frac{\sin(\pi t f_c)}{\pi t f_c} \tag{4.67}$$

che corrisponde al seno cardinale. Dalla convoluzione, inoltre, otteniamo

$$x(t) = x_c(t) \otimes h(t) = \left[\sum_{n=-\infty}^{\infty} x(n)\delta(t - nT_c)\right] \otimes \text{sinc}(tf_c) = \sum_{n=-\infty}^{\infty} x(n)\frac{\sin(\pi(t - nT_c)f_c)}{\pi(t - nT_c)f_c} \tag{4.68}$$

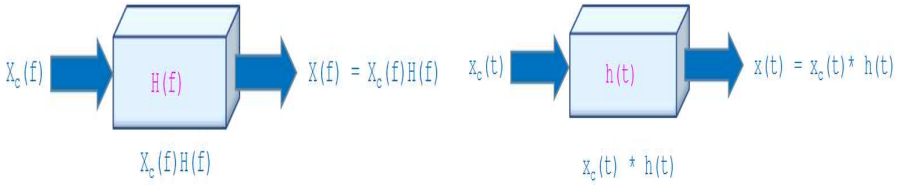


Figura 4.18: Filtraggio: a) nelle frequenze; b) nel tempo.

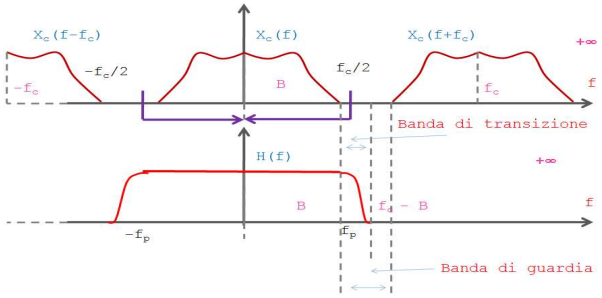


Figura 4.19: Banda di guardia.

4.6.6 Campionamento in pratica

Nel caso reale ci sono molte condizioni che non vengono soddisfatte. Ad esempio gli impulsi devono essere considerati si ampiezza e durata finita, il filtro passa-basso non è ideale e il segnale non è a banda limitata. Generalmente si considera una banda di guardia per il filtro in cui deve rientrare la banda di transizione come rappresentato in Figura 4.19.

4.7 I sistemi omomorfi

Fino ad ora ci siamo occupati della rappresentazione matematica dei segnali a tempo discreto e dei sistemi lineari invarianti alla traslazione (LTI). Qui presentiamo una classe di tecniche non lineari per l'elaborazione dei segnali e cioè i *sistemi omomorfi*. Le tecniche hanno trovato applicazione in numerosi campi come per lo studio di segnali con riverberazione, l'analisi della voce, le esplorazioni sismiche, il miglioramento di qualità delle immagini, ecc..

I sistemi omomorfi obbediscono ad un principio di sovrapposizione generalizzato. In molti casi il segnale audio può essere rappresentato come il prodotto di due o più segnali (o componenti). Ad esempio un segnale modulato in ampiezza rappresentato dal prodotto di

un segnale audio e di un inviluppo. In dettaglio supponiamo che un segnale sia il prodotto di due distinti segnali

$$x(n) = [x_1(n)]^a [x_2(n)]^b \tag{4.69}$$

Un sistema omomorfo può avere la seguente proprietà

$$T[x(n)] = aT[x_1(n)] + bT[x_2(n)] \tag{4.70}$$

Ad esempio una funzione che formalmente gode di questa proprietà è la funzione logaritmo

$$\log[x(n)] = a \log[x_1(n)] + b \log[x_2(n)] \tag{4.71}$$

Di particolare interesse potrebbe essere quello nello studio di segnali convolutivi. Infatti, sappiamo che usando un sistema lineare (LTI) non è possibile stimare le singole componenti di un segnale convoluto.
Allora, supponiamo di avere la convoluzione di due segnali $x(n) = x_1(n) \otimes x_2(n)$ e inoltre sappiamo che la convoluzione corrisponde ad un prodotto delle trasformate z e cioè

$$X(z) = X_1(z)X_2(z) \tag{4.72}$$

e quindi dalle proprietà dei sistemi omomorfi possiamo ottenere

$$\widetilde{X}(z) = \log[X(z)] = \log[X_1(z)X_2(z)] = \log[X_1(z)] + \log[X_2[z]] \tag{4.73}$$

Nel 1963 Bogert, Healy e Tukey pubblicarono un lavoro che introduceva il concetto di questi sistemi. Il logaritmo dello spettro di potenza di un segnale contenente un'eco ha una componente periodica additiva dovuta all'eco la trasformata di Fourier del logaritmo dello spettro di potenza deve avere un picco in corrispondenza del ritardo d'eco. Essi chiamarono questa funzione il *cepstrum*, anagrammando il termine inglese per spettro.

Capitolo 5

Filtri numerici

Uno dei meccanismi maggiormente usati nell'ambito dell'analisi dei segnali audio è quello del filtraggio. Si intende con filtro ogni operazione svolta su un segnale. In particolare possiamo vederlo come un dispositivo che aumenta o attenua l'energia connessa a certe regioni dello spettro di un suono. Il filtraggio è usato sia nei sistemi analogici che in quelli digitali. Un esempio di meccanismo di filtraggio è quello svolto dagli equalizzatori all'interno di un mixer, come abbiamo visto nel Capitolo 1. Infatti l'equalizzatore non è altro che un banco di filtri passa-banda. Inoltre, anche le bande critiche della membrana uditiva dell'orecchio umano si comportano come una serie di filtri pass-banda. Questo aspetto verrà approfondito nel Capitolo 6.
In questo Capitolo introdurremo la rappresentazione tramite grafi di un sistema LTI. La rappresentazione ci servirà per introdurre l'algoritmo per la trasformata veloce di Fourier (FFT) e per caratterizzare i filtri IIR e FIR.

5.1 Rappresentazione

Nel Capitolo 3 abbiamo introdotto i sistemi LTI e la loro descrizione tramite equazioni alle differenze. Le equazioni alle differenze descrivono un sistema e mettono in relazione le sue sequenze di ingresso e uscita. Inoltre abbiamo visto che le funzioni di trasferimento mettono in relazione le Trasformate z delle medesime sequenze. Per realizzare un filtro numerico occorre trasformare la relazione ingresso-uscita in un algoritmo di calcolo. Questa realizzazione è possibile tramite l'identificazione di un insieme di operazioni elementari. Graficamente queste operazioni possono essere descritte da *blocchi*. L'algoritmo di calcolo può essere definito tramite una struttura o rete di interconnessione tra tre operazioni di

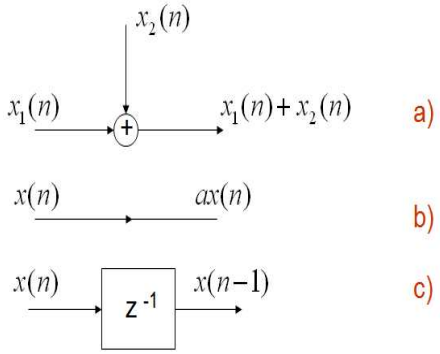


Figura 5.1: Blocchi elementari per la realizzazione di un filtro.

base. Le operazioni base richieste per realizzare un filtro numerico sono le seguenti (con riferimento alla Figura 5.1)

- un **sommatore** (a)
- un **moltiplicatore** per una costante (b)
- un dispositivo per memorizzare il valore precedente di una sequenza (o **ritardatore**) (c)

La rappresentazione del filtro, invece, è possibile ottenerla sia mediante la funzione caratteristica

$$H(z) = \frac{Y(z)}{X(z)} = \frac{\sum_{k=0}^{M} b_k z^{-k}}{\sum_{k=0}^{N} a_k z^{-k}} \tag{5.1}$$

sia mediante l'equazione alle differenze che lega l'ingresso e l'uscita

$$y(n) = \sum_{k=1}^{N} a_k y(n-k) + \sum_{k=0}^{M} b_k x(n-k) \tag{5.2}$$

La rappresentazione generale di un filtro risulta quella rappresentata in Figura 5.2.

5.1.1 Esempio

Come esempio consideriamo la seguente equazione

$$y(n) = a_1 y(n-1) + a_2 y(n-2) + bx(n). \tag{5.3}$$

Il diagramma a blocchi corrispondente è rappresentato in Figura 5.3. La funzione di

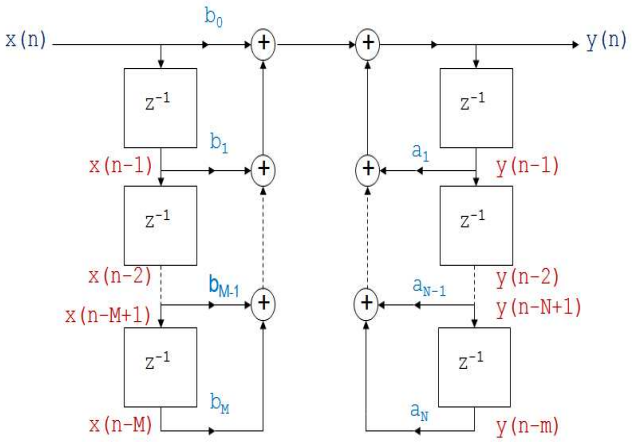


Figura 5.2: Rappresentazione generale di un filtro.

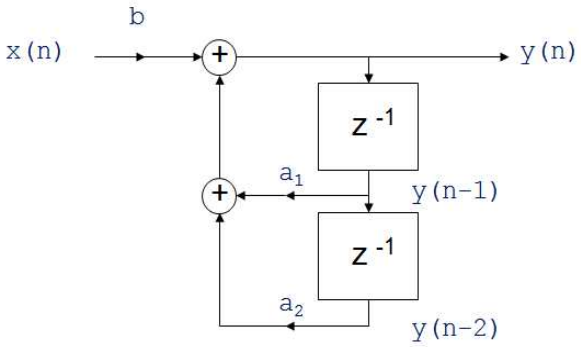


Figura 5.3: Esempio di rete di blocchi.

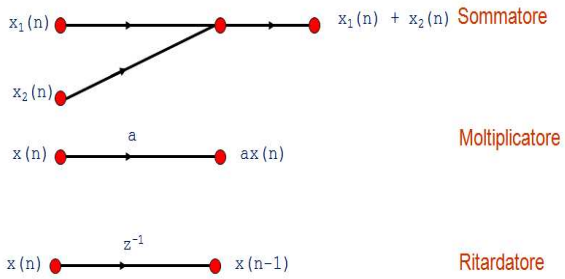


Figura 5.4: Operazioni base in un grafo di flusso lineare.

trasferimento invece risulta

$$H(z) = \frac{b}{1 + a_1 z^{-1} + a_2 z^{-2}}. \tag{5.4}$$

5.2 Grafi lineari

Molto spesso invece di rappresentare i sistemi tramite diagrammi a blocchi si usano i *grafi di flusso lineari*. I sistemi sono equivalenti ma risultano più compatti e possono essere messi in relazione con le matrici. Un grafo di flusso di segnale è una rete di rami orientati che si connettono in corrispondenza di nodi. Ad ogni nodo è associata una variabile o valore del nodo. In Figura 5.4 sono rappresentati gli schemi delle operazioni base come già avevamo introdotto per gli schemi a blocchi.

Spesso nei grafi vengono inseriti dei pesi sui nodi e sugli archi. Il valore associato al nodo k è w_k. Il ramo (ik) indica un ramo che ha origine nel nodo i e termina nel nodo k, con la direzione da i a k indicata con una freccia sul ramo stesso e il peso denotato w_{ik}.

Dato un grafo importante è la complessità di calcolo. Questa fase prevede la ricerca di un minimo numero di moltiplicatori e un minimo numero di rami di ritardo. La moltiplicazione, infatti, è un'operazione che richiede tempo e ogni elemento di ritardo corrisponde all'occupazione di un registro di memoria. Una riduzione nel numero dei moltiplicatori significa un aumento della velocità. Una diminuzione nel numero dei ritardi significa una riduzione nell'occupazione di memoria.

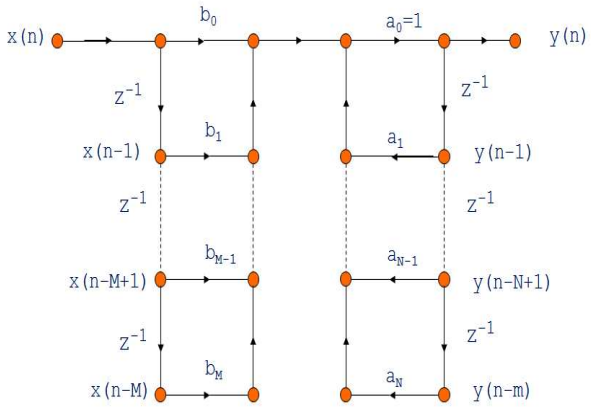


Figura 5.5: I forma diretta del filtro IIR.

5.3 Struttura dei filtri digitali

Per caratterizzare un filtro dobbiamo definire il tipo e la struttura della sua implementazione. I filtri *IIR* sono caratterizzati da una risposta all'impulso infinita. Sono chiamati *Auto-Regressivi a Media Mobile* (Auto-Regressive Moving Average, ARMA) o generalmente chiamati filtri ricorsivi. D'altra parte i filtri *FIR* sono caratterizzati da una risposta all'impulso di durata finita. Essi sono denotati come filtri a *Media Mobile* (Moving Average, MA).

5.3.1 Filtri IIR

Un filtro IIR è descritto dalla funzione di trasferimento dell'equazione 5.1 e quindi anche dall'equazione alle differenze dell'equazione 5.2.
Il modello più semplice che possiamo ottenere deriva direttamente dalle due equazioni precedenti. Viene chiamato schema della *I Forma Diretta*. In Figura 5.5 viene rappresentato lo schema generale. Il filtro può essere pensato come l'insieme di due reti poste in cascata: la prima realizza gli zeri e la seconda i poli.
Possiamo però notare che nei sistemi LTI la relazione ingresso-uscita complessiva della cascata è indipendente dall'ordine in cui sono disposti i sottosistemi. Queste considerazioni portano al filtro IIR in *II Forma Diretta* che è rappresentato in Figura 5.6.
Esistono anche altre strutture per la realizzazione di un filtro IIR come quella a *cascata* o quella in *parallelo*.

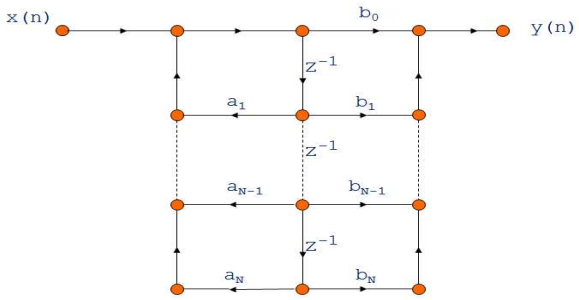


Figura 5.6: II forma diretta del filtro IIR.

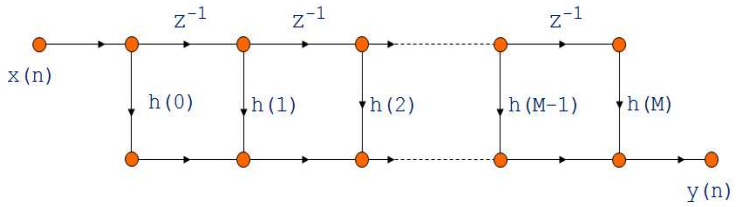


Figura 5.7: Forma diretta del filtro FIR.

5.3.2 Filtri FIR

Nei sistemi a media variabile, invece, la risposta all'impulso risulta di durata finita e le realizzazioni prendono la forma di algoritmi di calcolo non ricorsivi. In questo caso la funzione di trasferimento ha la forma

$$H(z) = \sum_{k=0}^{M} b_k z^{-k} \tag{5.5}$$

Se la risposta all'impulso ha una durata di N campioni, allora $H(z)$ è un polinomio di grado M. $H(z)$ ha M poli in $z = 0$ e M zeri che possono essere ovunque nel piano z al finito. La forma diretta può essere espressa osservando la seguente somma di convoluzione (vedi Figura 5.7)

$$y(n) = \sum_{k=0}^{M} b_k x(n-k) = \sum_{k=0}^{M} h(k) x(n-k) \tag{5.6}$$

Una struttura alternativa è quella a *cascata*.

5.4 Trasformata di Fourier Veloce

La DFT ricopre un ruolo importante nell'analisi dei segnali per questo motivo sono stati studiati diversi algoritmi efficienti per il calcolo della DFT. Runge e più tardi Danielson e Lanczos hanno descritto algoritmi la cui complessità è all'incirca proporzionale a $N \log N$ invece che a N^2. La possibilità di ridurre notevolmente i tempi di calcolo divenne chiara verso il 1965 quando Cooley e Tukey pubblicarono un noto algoritmo. Da allora sono stati introdotti numerosi algoritmi di calcolo noti come algoritmi per la trasformata di Fourier veloce, o semplicemente FFT (Fast Fourier Transform).
Possiamo distinguere principalmente due tipi di algoritmi

- *Decimazione nel tempo* - nello scomporre il calcolo in trasformate di dimensioni più piccole, la sequenza $x(n)$ viene suddivisa in sequenze sempre più corte
- *Decimazione in frequenza* - la sequenza DFT $X(k)$ viene scomposta in sottosequenze sempre più corte.

Successivamente ci concentreremo sull'approccio basato sulla decimazione nel tempo.
Per chiarire il concetto di complessità, consideriamo la trasformata discreta di Fourier
$X(k) =$

$$\sum_{n=0}^{N-1} \Big\{ (Re[x(n)]Re[W_n^{kn}] - Im[x(n)]Im[W_n^{kn}]) + j(Re[x(n)]Re[W_n^{kn}] + Im[x(n)]Im[W_n^{kn}]) \Big\} \tag{5.7}$$

per ogni $k = 0, \ldots, N-1$. Nel calcolo diretto $X(k)$ richiede $4N$ moltiplicazioni reali e $(4N-1)$ addizioni reali per ogni valore di k. In totale abbiamo $4N^2$ moltiplicazioni reali e $N(4N-1)$ addizioni reali.

5.4.1 Decimazione nel tempo

In questo tipo di approccio lo scopo principale è quello di dividere la DFT in due parti. Per fare ciò sfruttiamo sia la simmetria che la periodicità dell'esponenziale complesso $W_n^{kn} = e^{-j\frac{2\pi}{N}kn}$. Inoltre consideriamo un caso particolare e cioè quando N è potenza intera di 2 ($N = 2^v$). Se N non è una potenza di 2 possiamo aggiungere degli zero fino ad arrivare alla lunghezza desiderata (vedi il Capitolo precedente per lo zero-padding). Siccome N è un intero pari possiamo pensare di calcolare $X(k)$ dividendo $x(n)$ in due

sequenze di $N/2$ punti, una con indice pari e l'altra con indice dispari

$$X(k) = \sum_{n=0}^{N-1} x(n)e^{-j\frac{2\pi}{N}kn} = \sum_{n \text{ pari}}^{N-1} x(n)e^{-j\frac{2\pi}{N}kn} + \sum_{n \text{ dispari}}^{N-1} x(n)e^{-j\frac{2\pi}{N}kn} \tag{5.8}$$

Con la sostituzione della variabile n in modo tale che $n = 2r$ per n pari e $n = 2r + 1$ per n dispari otteniamo

$$X(k) = \sum_{r=0}^{N/2-1} x(2r)e^{-j\frac{2\pi}{N}k2r} + \sum_{r=0}^{N/2-1} x(2r+1)e^{-j\frac{2\pi}{N}k(2r+1)}. \tag{5.9}$$

Con semplici passaggi otteniamo

$$X(k) = \sum_{r=0}^{N/2-1} x(2r)(W_N^2)^{kr} + \sum_{r=0}^{N/2-1} x(2r+1)(W_N^2)^{kr} \tag{5.10}$$

da cui considerando $W_N^2 = e^{-j\frac{2\pi}{N/2}} = W_{N/2}$ abbiamo la seguente relazione

$$X(k) = \sum_{r=0}^{N/2-1} x(2r)W_{N/2}^{kr} + W_N^k \sum_{r=0}^{N/2-1} x(2r+1)W_{N/2}^{kr} = G(k) + W_N^k H(k) \tag{5.11}$$

Ciascuna delle due sommatorie $G(k)$ e $H(k)$ è una DFT di $N/2$ punti (vedi Figura 5.8a). Se $N/2$ è pari il calcolo di ciascuna DFT su $N/2$ punti si può effettuare mediante il calcolo e la successiva combinazione di due DFT su $N/4$ punti (vedi Figura 5.8b). Ripetiamo il procedimento fino ad arrivare al calcolo della DFT di due punti (vedi Figura 5.8c). Questo calcolo base viene chiamato calcolo a *farfalla* e fa riferimento ad alcune semplificazioni che si possono ottenere considerando le proprietà di W_N. Infatti abbiamo che $W_N^0 = 1$ e $W_2 = W_N^{N/2} = -1$. In Figura 5.8d viene rappresentato lo schema completo per il calcolo della DFT.

Versione ricorsiva

E' possibile realizzare una versione ricorsiva con una strategia divide-et-impera dell'algoritmo FFT. Qui denotiamo con

$$X[k] = \sum_{j=0}^{N/2-1} x[j]W_N^{kj} \tag{5.12}$$

la DFT, dove $X = (X[0], X[1], \ldots, X[N-1])$ sono i coefficienti DFT e $\mathbf{x} = (x[0], x[1], \ldots, x[N-1])$ la sequenza di input. La strategia si basa sull'applicazione della suddivisione tra i coefficienti di indice pari e quelli di indice dispari come abbiamo visto precedentemente. Lo pseudocodice dell'algoritmo risulta (vedi la libreria ESA-DSP per il codice in C++)

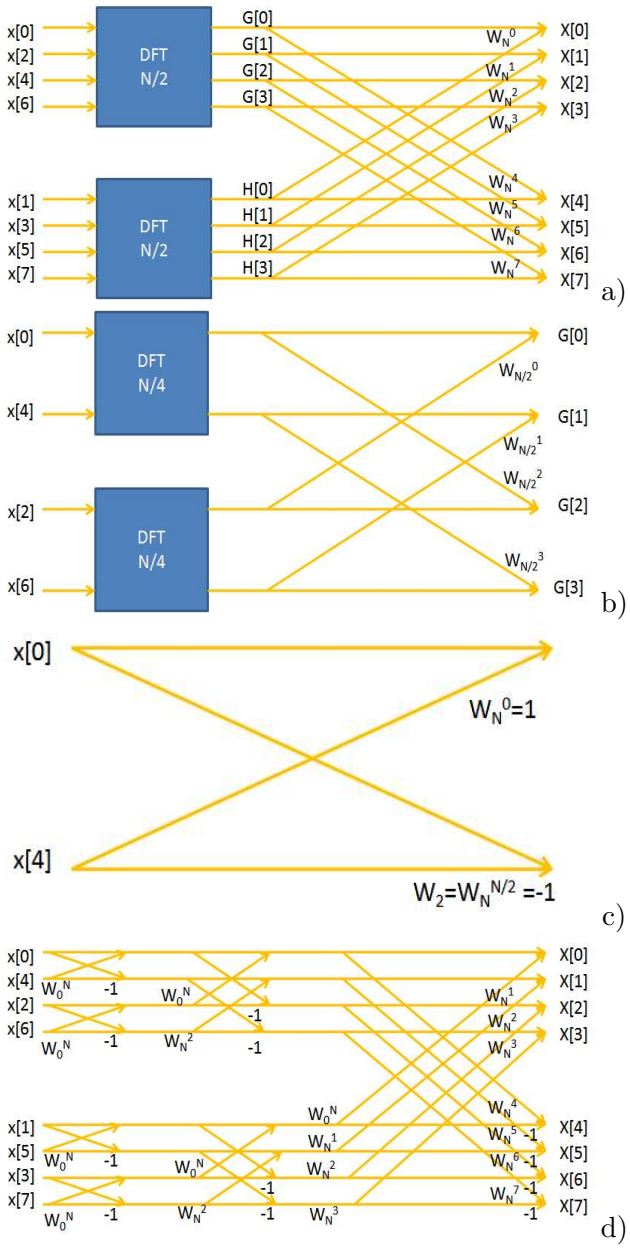


Figura 5.8: Calcolo incrociato per la FFT.

```
FFT-Ricorsiva(x)
   N = length(x);
   if N==1
      then return x[0]
   WN = exp(j  2 PI/N)
   W = 1
   Xp = FFT-Ricorsiva([x[0], x[2], x[4], . . . , x[N-2]])
   Xd = FFT-Ricorsiva([x[1], x[3], x[5], . . . , x[N-1]])
   for k = 0 to N/2 - 1
         do
            X[k] = Xp + W Xd
            X[k+N/2] = Xp - W Xd
            W = W W_N
     return X
```

Vediamo ora la complessità di calcolo. Notiamo che ogni invocazione richiede una complessità del tipo $\Theta(N)$ dove N è la lunghezza del vettore di input. La ricorrenza per il calcolo del tempo di esecuzione risulta

$$T(n) = 2T(n/2) + \Theta(N) = \Theta(N \log N) \tag{5.13}$$

Anche la trasformata inversa può essere calcolata in tempo $\Theta(N \log N)$.
Un risultato fondamentale che si può raggiungere sfruttndo le proprietà della FFT è quello relativo alla convoluzione di sequenze. Dal teorema della convoluzione, infatti, abbiamo che date due sequenze discrete $\mathbf{x}$ e $\mathbf{y}$ qualsiasi di lunghezza N, dove N è una potenza di 2

$$\mathbf{x} \otimes \mathbf{y} = \mathrm{DFT}_{2N}^{-1}\left(\mathrm{DFT}_{2N}(\mathbf{x}) \cdot \mathrm{DFT}_{2N}(\mathbf{y})\right) \tag{5.14}$$

dove i vettori $\mathbf{x}$ e $\mathbf{y}$ sono estesi con degli 0 fino alla lunghezza $2N$. In questo modo effettuando la convoluzione passando nel dominio delle frequenze e sfruttando la FFT possiamo ottenere una minore complessità di tempo. Questo risultato è di notevole importanza siccome la convoluzione è alla base di diversi algoritmi di elaborazione dei segnali audio. In particolare essa è usata nella fase di filtraggio come vedremo successivamente.
Possiamo, inoltre, arrivare ad una realizzazione più efficiente della FFT che impiega tempo $N \log N$ con una costante nella notazione Θ più piccola del caso precedente. Il grafo di flusso suggerisce un modo utile di memorizzare i dati originari e i risultati dei calcoli negli stadi intermedi mediante il calcolo a farfalla. Le chiamate ricorsive in una invocazione di

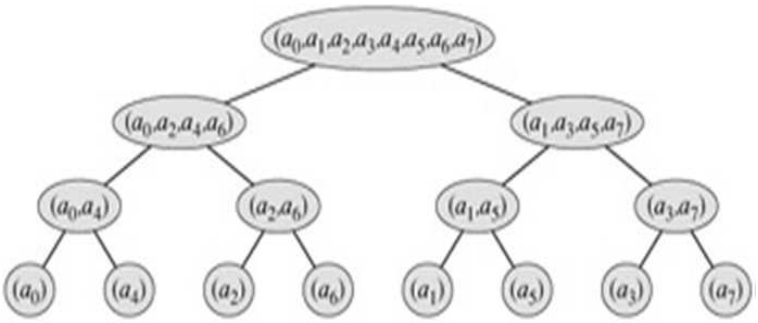


Figura 5.9: Albero di ricorsione della FFT.

FFT-Ricorsiva sono sistemati in una struttura ad albero (vedi Figura 5.9). Se si potessero sistemare gli elementi del vettore iniziale a nell'ordine in cui essi appaiono sulle foglie, si può simulare l'esecuzione della FFT-Ricorsiva in modo differente come mostrato nel grafo di flusso complessivo.

5.5 Progetto di filtri numerici

Il filtro è un dispositivo che aumenta o attenua l'energia connessa a certe regioni dello spettro del suono. Questa è un'operazione tipicamente svolta dagli equalizzatori come abbiamo accennato nel Capitolo 1. Un equalizzatore, infatti, non è altro che un banco di filtri passa-banda.
Spesso le specifiche del filtro sono date nel dominio della frequenza. In base alle loro caratteristiche possiamo classificare 4 tipi di filtri

- *passa-basso*
- *passa-banda*
- *passa-alto*
- *stop-banda*

Successivamente ci concentreremo sulla progettazione dei filtri IIR ma in maniera più dettagliata dei filtri FIR. La progettazione parte dalla descrizione del filtro passa-basso ideale poichè da questo successivamente si possono ricavare gli altri tipi di filtro.

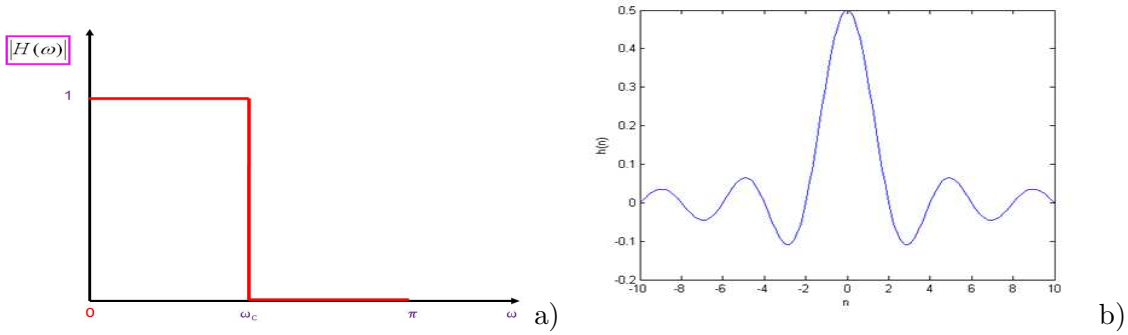


Figura 5.10: Filtro passa-basso: dominio delle frequenze; dominio del tempo.

5.5.1 Passa-basso ideale

Un filtro passa-basso *ideale* ha la risposta in frequenza $H(\omega)$ di ampiezza diversa da zero in una banda di frequenze simmetrica rispetto all'origine. Il filtro ideale passa-basso a tempo discreto con frequenza di taglio ω_c ha come risposta in frequenza un rettangolo di ampiezza unitaria e base $2\omega_c$. $H(\omega)$ per $-\pi < \omega < \pi$ risulta

$$H(\omega) = \begin{cases} 1 & |\omega| \leq \omega_c \\ 0 & \omega_c < |\omega| < \pi \end{cases} \tag{5.15}$$

In Figura 5.10a viene rappresentata solo la parte positiva del filtro ideale nell'intervallo $[0, \pi]$. Dobbiamo precisare che il limite π è frutto delle condizioni imposte sulla funzione di trasferimento periodica $H(\omega)$ come abbiamo visto nel Capitolo 4. Questo implica che nel caso delle frequenze essendo $f = \omega/2\pi$, la frequenza massima risulta $1/2 = 0.5$ Hz.
Come già sottolineato la funzione di trasferimento $H(\omega)$ corrisponde, nel dominio del tempo, a descrivere la risposta all'impulso con una sequenza discreta del tipo seno cardinale (vedi Figura 5.10b)

$$h(n) = \frac{\sin(\omega_c n)}{\pi n} \tag{5.16}$$

5.5.2 Passa-basso reale

Purtroppo non esiste un filtro ideale. Vediamo quindi come progettare un filtro passa-basso *reale*. In questo caso le specifiche prendono la forma di un insieme di limiti di tolleranza. Nella progettazione del filtro possiamo identificare tre bande: la *banda passante*, la *banda*

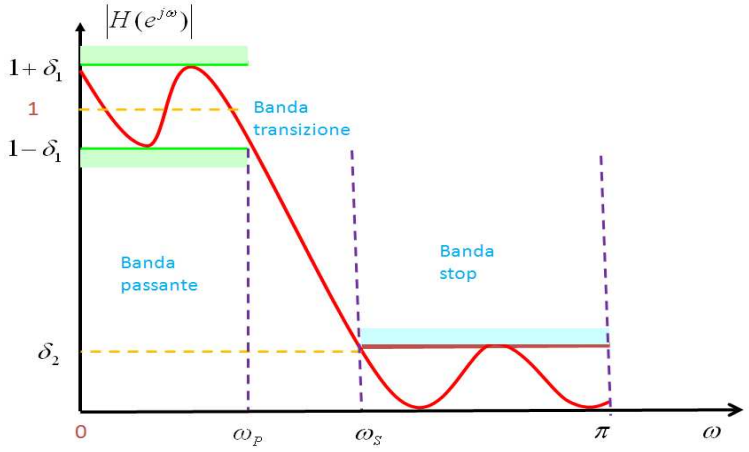


Figura 5.11: Filtro passa-basso reale.

di transizione e la *banda oscura* (o stop) (vedi Figura 5.11). Il ruolo della banda passante è quello di determinare la regione in cui la risposta in frequenza deve approssimare 1. In questa regione i parametri fondamentali sono la frequenza di taglio ω_p e il parametro di tolleranza δ_1. In base al parametro di tolleranza δ_1 la risposta in frequenza deve essere compreso tra $1-\delta_1$ e $1+\delta_1$ per tutte le frequenze minori di ω_p. La banda di transizione, invece, è definita tra ω_p e ω_s dove ω_s è la frequenza di taglio della banda oscura. La banda di transizione determina il passaggio del filtro da 1 a 0. La banda oscura, invece, determina la parte del filtro in cui il modulo della funzione di transizione deve approssimare 0. Il parametro di tolleranza di questa banda è δ_2. Il modulo della risposta in frequenza in questo caso deve essere minore uguale di δ_2 per tutte le frequenze maggiori della frequenza di taglio ω_s e minori di π.
Spesso la funzione di trasferimento $H(\omega)$ è espressa in decibel come in Figura 5.12. I nuovi parametri sono descritti come

- R_p - attenuazione dell'oscillazione nella banda passante
- A_s - attenuazione nella banda oscura

5.5.3 Filtro IIR

L'approccio tradizionale al progetto di filtri numerici IIR comporta la trasformazione di un filtro analogico in un filtro numerico soddisfacente determinate specifiche. Abbiamo

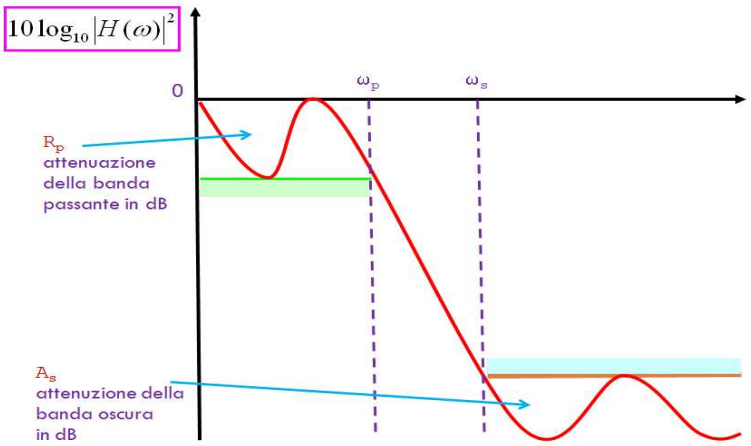


Figura 5.12: Filtro passa-basso reale in db.

quindi la necessità di descrivere le tecniche per l'implementazione di filtri analogici e successivamente di introdurre alcune metodologie per la trasformazione di un filtro analogico in uno digitale.
Ci sono diversi procedimenti per trasformare un progetto di filtro analogico in un progetto di filtro numerico

- *invarianza all'impulso*
- *soluzione numerica dell'equazione differenziale*
- *trasformazione bilineare*

Nella tecnica di invarianza all'impulso si sceglie, come risposta all'impulso, una sequenza costituita da campioni ugualmente spaziati della risposta all'impulso del filtro analogico (con T periodo di campionamento).
Nella soluzione numerica, invece, si approssimano le derivate parziali nell'equazione differenziale che rappresenta l'uscita di un sistema, con differenze finite.
L'uso della trasformazione bilineare, infine, fornisce filtri numerici stabili partendo da filtri analogici stabili. La trasformazione bilineare mappa l'intero asse immaginario sulla circonferenza unitaria nel piano z tramite la funzione tangente. Il prezzo pagato per questo è l'introduzione di una distorsione nell'asse frequenza. Il progetto di filtri numerici per mezzo della trasformazione bilineare è utile quando questa distorsione può essere tollerata o compensata.

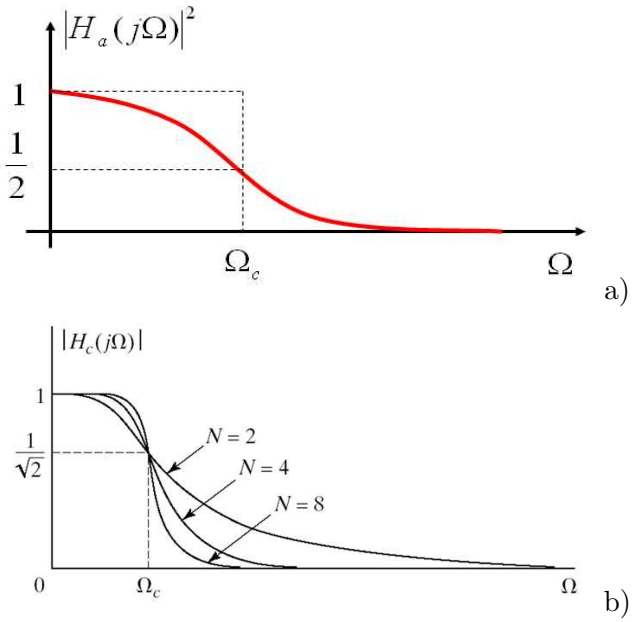


Figura 5.13: Filtro passa-basso di Butterworth analogico.

D'altra parte le tecniche di approssimazione di un filtro ideale passa-basso analogico possono essere

- tecnica di *Butterworth*
- tecnica di *Chebyshev*
- tecnica *Ellittica*.

Analizziamo in maniera più dettagliata questi tipi di filtri.

Butterworth

In questo modello la risposta in ampiezza è massimamente piatta nella banda passante. L'approssimazione è monotona sia nella banda passante che nella banda oscura. La risposta in frequenza di un filtro analogico di Butterworth è mostrata in Figura 5.13a. Nel filtro di Butterworth un coefficiente N identifica il suo "ordine". In base ad N possiamo avere una diversa approssimazione (vedi Figura 5.13b). In figura Ω identifica le frequenze nel caso di segnali continui e Ω_c la frequenza di taglio. Al crescere del parametro N, infatti, la caratteristica del filtro diventa più ripida: essa rimane, cioè, prossima all'unità su un

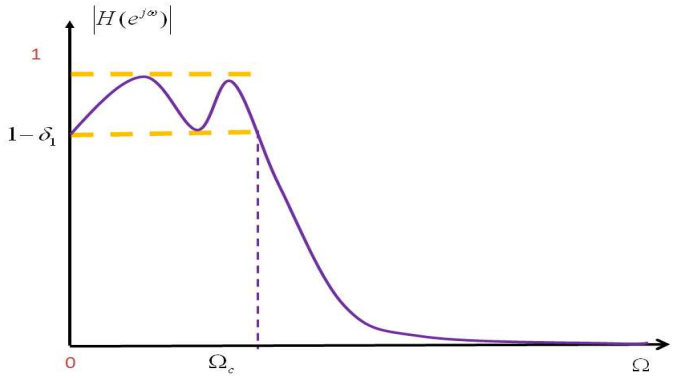


Figura 5.14: Filtro di Chebyshev di I tipo.

tratto sempre maggiore della banda passante e va sempre più rapidamente nella banda oscura.

Chebyshev

Il filtro di Chbyshev si ottiene scegliendo un'approssimazione del filtro con caratteristica di oscillazione uniforme (equiripple) invece che con caratteristica monotona. E' possibile definire due tipi di filtri

- I - il modulo della risposta in frequenza è a oscillazione uniforme nella banda passante e monotona nella banda oscura
- II - monotona nella banda passante ed a oscillazione uniforme nella banda oscura

In Figura 5.14 viene presentato un filtro di Chebyshev del I tipo.

Ellittici

C'è comunque la possibilità di ulteriori miglioramenti distribuendo anche l'errore relativo alla banda oscura uniformemente nella banda. Questo tipo di approssimazione, cioè ad oscillazione uniforme nella banda passante e in quella oscura, è la migliore che si può raggiungere per un dato ordine N del filtro. Questo tipo di filtri sono chiamati Ellittici e un esempio è mostrato in Figura 5.15.

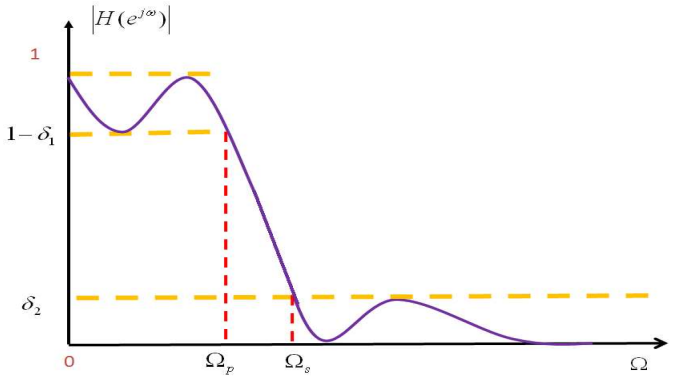


Figura 5.15: Filtro Ellittico.

Trasformazioni di frequenza

Per ottenere gli altri tipi di filtri (passa-banda, passa-alto e banda-stop), l'approccio tradizionale è quello di progettare inizialmente un prototipo di filtro passa-basso e poi con trasformazioni algebriche dal prototipo ottenere il filtro con la caratteristica desiderata.

Tecniche alternative

Negli ultimi anni sono state sviluppate delle tecniche di progetto di natura algoritmica. Esse si basano generalmente sull'uso di un elaboratore per risolvere sistemi di equazioni lineari o non lineari. Tra le tecniche più utilizzate sono quelle di *minimizzazione dell'errore quadratico medio* (EQM) o in modo più generale la minimizzazione dell'errore di ordine p.

5.5.4 Filtri FIR

Per realizzare un filtro, se si desidera sfruttare il vantaggio offerto dalla velocità di calcolo della FFT è essenziale avere una risposta all'impulso di durata finita. I filtri IIR consentono di ottenere generalmente delle eccellenti risposte di ampiezza con l'inconveniente però di risposte di fase non lineari. Al contrario, i filtri FIR possono avere fase esattamente lineare. Come nel caso di filtri IIR iniziamo ad introdurre le specifiche di un tipico filtro passa-basso in quanto gli altri filtri si possono ricavare da questi con semplici operazioni.
Abbiamo più volte ribadito che il filtro ideale passa-basso ha una frequenza di taglio ω_c e come risposta in frequenza un rettangolo di ampiezza unitaria e base $2\omega_c$. Consideriamo la

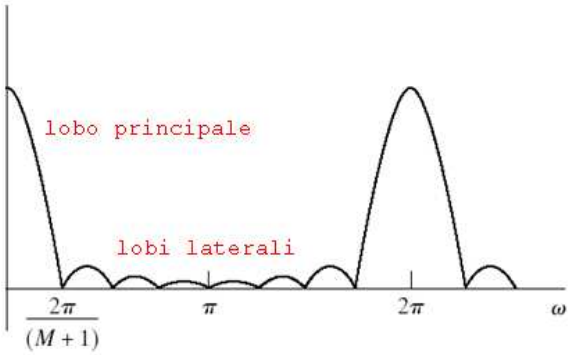


Figura 5.16: Lobo centrale e lobi laterali della funzione di trasferimento di una finestra rettangolare con 7 punti.

risposta in frequenza come nell'equazione 5.15 e denotiamola con $H_d(\omega)$ e la sua risposta all'impulso dell'equazione 5.16 e la denotiamo con $h_d(n)$. La risposta all'impulso $h_d(n)$ ha durata infinita mentre noi abbiamo bisogno di una risposta all'impulso di durata finita. La risposta all'impulso finita $h(n)$ è ottenuta come prodotto tra $h_d(n)$ e una "finestra" (window) $w(n)$ di durata finita

$$h(n) = h_d(n)w(n) \tag{5.17}$$

dove

$$w(n) = \begin{cases} 1 & 0 \leq n \leq M-1 \\ 0 & \text{altrove} \end{cases} \tag{5.18}$$

Dal Teorema di modulazione introdotto nel Capitolo 4 possiamo anche affermare che $H(\omega)$ risulta la convoluzione della risposta in frequenza ideale $H_d(\omega)$ con la trasformata di Fourier della finestra la risposta in frequenza $W(\omega)$. $H(\omega)$ sarà, quindi, una versione "smussata" della risposta desiderata $H_d(\omega)$. Il processo di filtraggio nel dominio delle frequenze è rappresentato in Figura 5.17a. Nel dominio del tempo il filtraggio può avvenire tramite la convoluzione con la sequenza finita $h(n)$ (vedi Figura 5.17b).

Un effetto inerente la scelta di finestre di lunghezza finita è che la funzione di trasferimento $W(\omega)$ si presenta costituita da un *lobo centrale* e una serie di *lobi laterali*. Nella Figura 5.16 viene rappresentata la funzione di trasferimento $W(\omega)$ di una finestra rettangolare con $M = 7$ punti.

La larghezza del lobo centrale è proporzionale a $1/M$ e i lobi laterali sono di dimensione minore. Approfondiremo questo concetto quando descriveremo il *fenomeno di Gibbs*. In

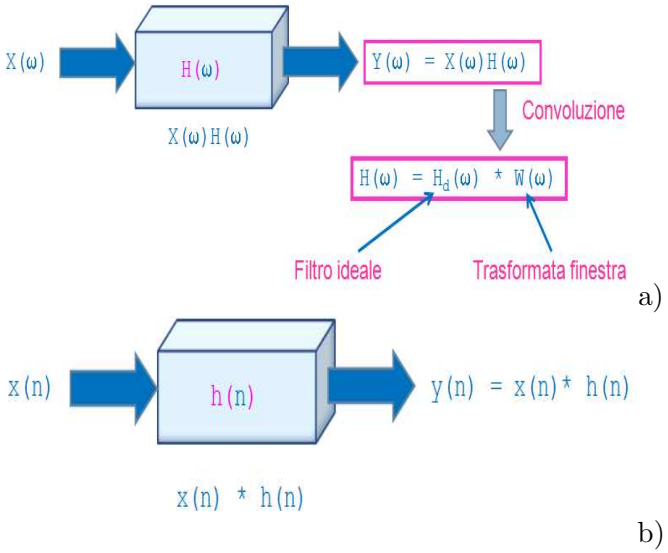


Figura 5.17: Filtro FIR: a)nelle frequenze; b) nel tempo.

FIR				
#	fc	:	double	frequenza di campionamento
#	scw	:	int	finestra
#	x ir	:	vector<double>	sequenza filtro
#	L	:	int	lunghezza finestra
#	M	:	int	lunghezza filtro
+	FIR (vector<double>)			costruttore
+	passabasso()	:	void	filtro passa basso
+	passabanda()	:	void	filtro passa banda
+	passaalto()	:	void	filtro passa alto
+	bandastop()	:	void	filtro bandastop

Tabella 5.1: Classe **FIR**.

base a queste considerazioni vedremo che per progettare il filtro FIR bisogna scegliere la lunghezza M e il tipo di finestra da adottare. Nel codice seguente viene presentata la classe **FIR** che implementa la classe principale per il filtraggio FIR basato sul windowing (la Tabella 9.16.2 descrive la classe)

```
template <class Type> class FIR : public sequenze<Type> {
public
    FIR(vector<double> in) : sequenze<double>(in){
    cout << "Inserisci la frequenza di campionamento del segnale \n";
    cin >> fc;
    cout << "Inserisci il tipo di finestra: \n"
         << "1) Rettangolare \n"
```

```
            << "2) Hamming \n"
            << "3) Hanning \n"
            << "4) Triangolare \n"
            << "5) Blackman \n"
            << "6) Harris \n" ;
        cin >> sc_w;
        cout << "Dammi il numero di coefficienti del filtro";
        cin >> M;
        L = in.size();
        x = in;
        x_ir = vector<double>(M,0.0);
    }

    . . .

    private:
        int L, M;
        vector<double> x_ir;
        double fc;
        int sc_w;
    };
```

Il codice seguente invece è relativo al metodo **passa-basso** che ci permette di ottenere il filtro passa-basso

```
    void passa_basso() {
        cout <<" \n \n FILTRO PASSA-BASSO \n \n";
        double fp;
        cout <<"Inserisci la frequenza di taglio della banda passante\n";
        cin >> fp;
         // trasformazione delle frequenze di taglio in pulsazioni
        double wc = (double)fp/fc * 2 * PI;
        // risposta all'impulso - seno cardinale
        sinc<double> ir(wc,M);
        window w(M);
        // Applicazione della finestra precedentemente scelta
        switch(sc_w)
        {
        case 1:  ir.prodotto(&w); break;
```

```
    case 2:  w.ham();   ir.prodotto(&w); break;
    case 3:  w.han();   ir.prodotto(&w); break;
    case 4:  w.triang();    ir.prodotto(&w);  break;
    case 5:  w.black();   ir.prodotto(&w);  break;
    case 6:  w.harris();   ir.prodotto(&w); break;
    }
    x_ir = ir.get_x();
    // Filtraggio tramite convoluzione
    conv(ir.get_x());
    printf("**\n");
}
```

Fenomeno di Gibbs

I filtri FIR basati su windowing sono soggetti al fenomeno di Gibbs. Questo fenomeno è correlato alla lunghezza M del filtro. Quando M aumenta, infatti, la larghezza del lobo principale diminuisce. Per la finestra rettangolare i lobi laterali non sono trascurabili quando M cresce. Infatti in questo caso la larghezza di ogni lobo diminuisce con M, le oscillazioni hanno una frequenza maggiore ma non diminuiscono in ampiezza. questo fenomeno viene chiamato *fenomeno di Gibbs*. Questo fenomeno è rappresentato nella Figura 5.18 in cui vengono proposte diverse realizzazioni di un filtro ideale FIR, con funzione di trasferimento $H_d(\omega)$, al variare del numero M di campioni.
Per diminuire l'altezza dei lobi laterali, per allargare il lobo principale e avere una transizione meno ripida dobbiamo usare finestre meno dastriche o in altre parole più "dolci". Le finestre comunemente usate sono

- **Rettangolare**
- **Bartlett**
- **Hanning**
- **Hamming**
- **Blackman**

Il seguente codice C++ è relativo alla classe **window** per creare le precedenti finestre

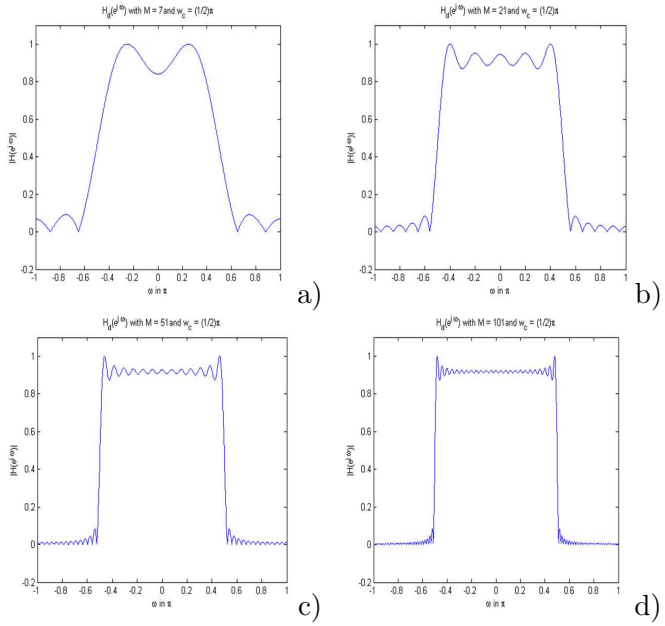


Figura 5.18: Rappresentazione del fenomeno di Gibbs su un filtro FIR variando il numeri di campioni: a) $M = 7$; $M = 21$; $M = 51$; $M = 101$.

window				
+	window (int)			costruttore (rettangolare)
+	ham()	:	void	Hamming
+	han()	:	void	Hanning
+	black()	:	void	Blackman
+	harris()	:	void	Harris

Tabella 5.2: Classe **window**.

```
class window : public sequenze<double>{
  public:
     window(int M) : sequenze<double>(M){
     x = vector<double>(M,1.0);
    };
. . . };
```

La classe è descritta in Tabella 5.5.4. Successivamente approfondiremo le caratteristiche di queste finestre e le proprietà che scaturiscono dal loro utilizzo nel filtraggio di tipo FIR.

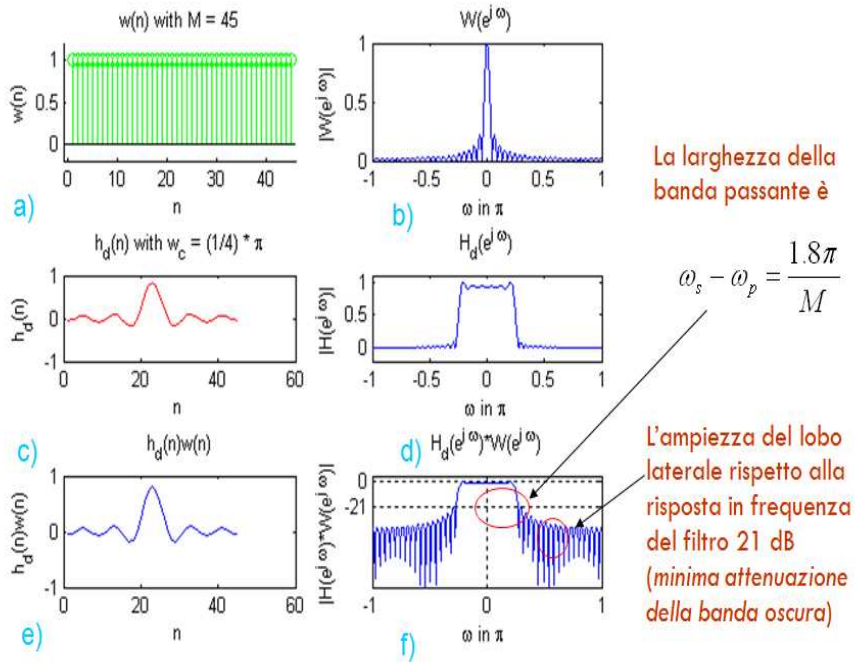


Figura 5.19: Filtraggio FIR con finestra rettangolare.

Finestra rettangolare

Per chiarire il ruolo delle finestre nel filtraggio mostriamo le diverse fasi nella costruzione di un filtro passa-basso con finestra rettangolare. Nella Figura 5.19a viene rappresentata la sequenza della finestra $w(n)$ composta da 45 punti. Nella Figura 5.19b viene visualizzato il modulo della trasformata di Fourier della finestra. Nella Figura 5.19c viene mostrato il filtro ideale nel dominio del tempo e nella Figura 5.19d il filtro nel dominio delle frequenze. Nella Figura 5.19e viene visualizzato il prodotto tra il filtro ideale e la finestra o in altri termini il filtro nel dominio del tempo. In Figura 5.19f viene visualizzato il modulo della convoluzione tra le risposte in frequenza del filtro ideale e della finestra. In altre parole il filtro nel dominio delle frequenze. Possiamo notare dalla Figura 5.19f che l'ampiezza della banda passante in questo caso è $1.8\pi/M$. Inoltre l'ampiezza del lobo laterale rispetto alla risposta in frequenza del filtro è di 21 dB. Quest'ultima definisce la minima attenuazione della banda oscura e non dipende dalla lunghezza della finestra ma dal tipo di finestra adottata.

Finestra di Bartlett (triangolare)

Siccome il fenomeno di Gibbs deriva dal fatto che la finestra rettangolare ha una transizione drastica tra 0 e 1, Bartlett introdusse una funzione con una transizione più graduale. La forma di questa finestra è triangolare e risulta

$$w(n) = \begin{cases} 2n/M & 0 \leq n \leq /2 \\ 2 - 2n/M & M/2 \leq n \leq M \\ 0 & \text{altrove} \end{cases} \tag{5.19}$$

Il codice C++ per la realizzazione della finestra triangolare (*triang*) è il seguente

```
void triang() {
  int    i;
  double tri,a;
  int   n = size;
  a = 2.0/(n-1);
  for (i = 0 ; i <= (n-1)/2 ; i++) {
    tri = i*a;
    x[i] *= (float)tri;
  }
  for ( ; i < n ; i++) {
    tri = 2.0 - i*a;
    x[i] *= (float)tri;
  }
}
```

Come nel caso precedente costruiamo un filtro passa-basso usando una finestra di Bartlett. Nella Figura 5.20a viene raffigurata la sequenza della finestra $w(n)$ di dimensioni 45. Nella 5.20b viene visualizzato il modulo della trasformata di Fourier della finestra. Nella Figura 5.20c viene mostrato il filtro ideale nel dominio del tempo e nella Figura 5.20d lo stesso nel dominio delle frequenze. Nella Figura 5.20e viene visualizzato il filtro nel dominio del tempo. Nella 5.20f viene visualizzato il modulo della convoluzione tra le risposte in frequenza del filtro ideale e della finestra (il filtro nel dominio delle frequenze). L'ampiezza della banda passante in questo caso è pari a $6.1\pi/M$. Inoltre l'ampiezza del lobo laterale rispetto alla risposta in frequenza del filtro è di 26 dB.

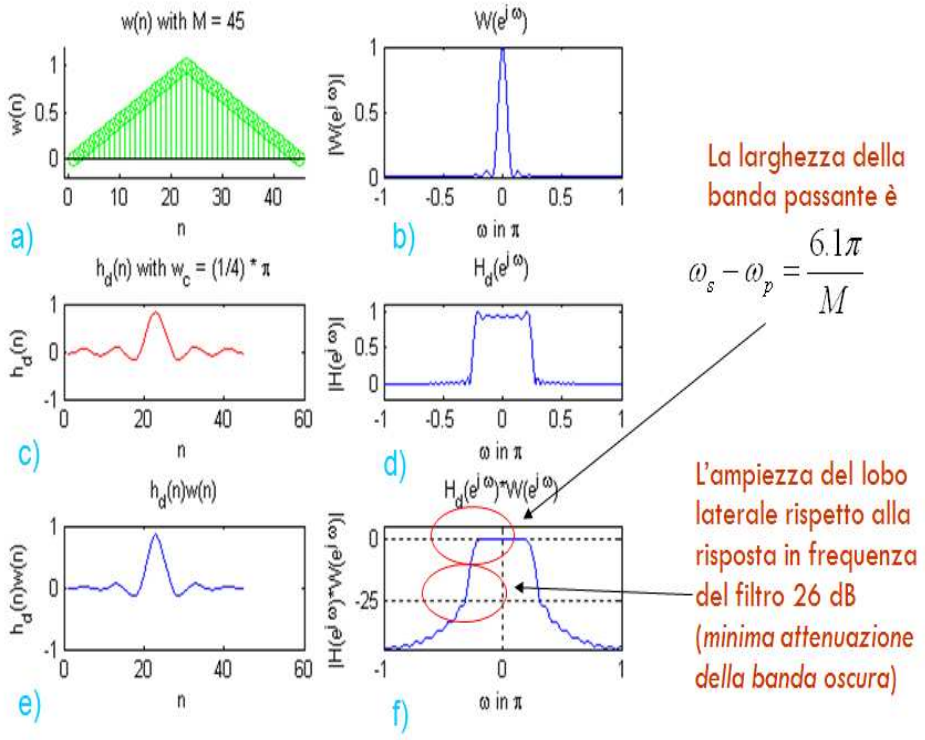


Figura 5.20: Filtraggio FIR con finestra triangolare.

Finestra di Hanning

La finestra di Hanning, invece, è basata su una funzione coseno e risulta quindi meno drastica delle due precedenti. La funzione è la seguente

$$w(n) = \begin{cases} \frac{1}{2}\left[1 - \cos(\frac{2\pi n}{M})\right] & 0 \leq n \leq M \\ 0 & \text{altrove} \end{cases} \tag{5.20}$$

Il codice C++ relativo è il seguente

```
void han() {
  int    i;
  double factor,han;
  int n = size;
  factor = 8.0*atan(1.0)/(n-1);
  for (i = 0 ; i < n ; i++){
    han = 0.5 - 0.5*cos(factor*i);
    x[i] *= (float)han;
  }
}
```

Come nei casi precedenti vediamo come costruire il filtro passa-basso usando un finestra di Hanning. Nella Figura 5.21a viene raffigurata la sequenza della finestra $w(n)$ di dimensioni 45. Nella Figura 5.21b viene visualizzato il modulo della trasformata di Fourier della finestra. Nelle Figure 5.21c e 5.21d vengono mostrati il filtro ideale nel dominio del tempo e nel dominio delle frequenze. Nella Figura 5.21e viene visualizzato il prodotto tra il filtro ideale e la finestra. In Figura 5.21f viene visualizzato il modulo della convoluzione tra le risposte in frequenza del filtro ideale e della finestra. Dalla Figura 5.21f si evince che l'ampiezza della banda passante in questo caso è $6.2\pi/M$. Inoltre l'ampiezza del lobo laterale rispetto alla risposta in frequenza del filtro è di 44 dB.

Finestra di Hamming

La finestra di Hamming è simile a quella di Hanning eccetto per il fatto che presenta una piccola discontinuità. Un questo caso abbiamo la seguente funzione

$$w(n) = \begin{cases} 0.54 - 0.46\left[1 - \cos(\frac{2\pi n}{M})\right] & 0 \leq n \leq M \\ 0 & \text{altrove} \end{cases} \tag{5.21}$$

Il codice C++ risulta

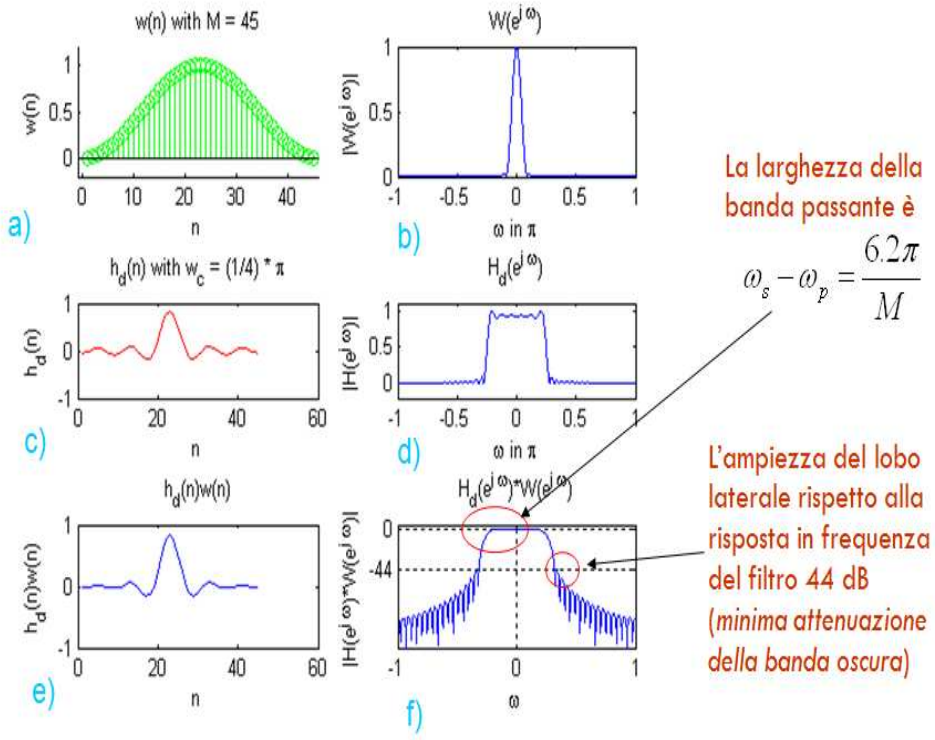


Figura 5.21: Filtraggio FIR con finestra di Hanning.

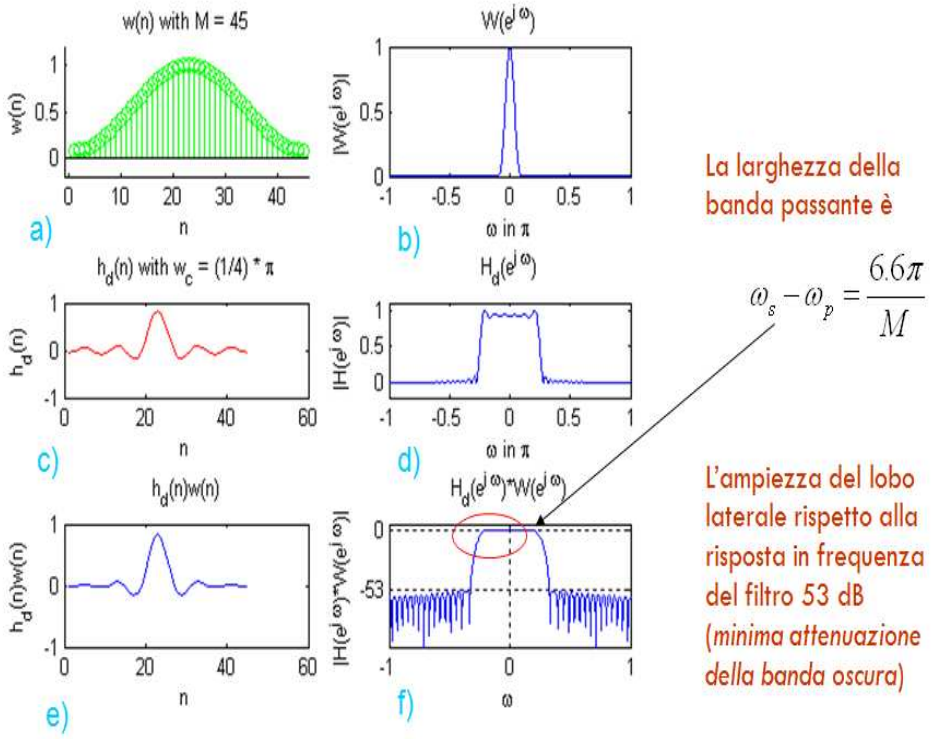


Figura 5.22: Filtraggio FIR con finestra di Hamming.

```
void ham() {
  int    i;
  double ham,factor;
  int n = size;
  factor = 8.0*atan(1.0)/(n-1);
  for (i = 0 ; i < n ; i++){
    ham = 0.54 - 0.46*cos(factor*i);
    x[i] *= (float)ham;
  }
}
```

In Figura 5.22 mostriamo le caratteristiche del filtraggio usando un finestra di Hamming con $M = 45$. La Figura ricostruisce i passi di realizzazione del filtro FIR come abbiamo visto nei casi precedenti. Concentrandosi sulla Figura 5.22f possiamo evidenziare l'ampiezza della banda passante che in questo caso è $6.6\pi/M$. Inoltre l'ampiezza del lobo laterale rispetto alla risposta in frequenza del filtro è di 53 dB.

Finestra di Blackman

La finestra di Blackman è simile alle due precedenti e in più contiene un secondo termine armonico. La finestra è descritta dalla seguente equazione

$$w(n) = \begin{cases} 0.42 - 0.5\left[1 - \cos(\frac{2\pi n}{M})\right] + 0.08\cos(\frac{4\pi n}{M}) & 0 \leq n \leq M \\ 0 & \text{altrove} \end{cases} \tag{5.22}$$

Il codice C++ corrispondente è il seguente

```
void black(){
  int    i;
  double black,factor;
  int n = size;
  factor = 8.0*atan(1.0)/(n-1);
  for (i=0; i<n; ++i){
    black = 0.42 - 0.5*cos(factor*i) + 0.08*cos(2*factor*i);
    x[i] *= (float)black;
  }
}
```

Possiamo ora costruire il filtro passa-basso come nei casi precedenti (vedi Figura 5.23). Nella Figura 5.23a viene raffigurata la sequenza della finestra. Nella Figura 5.23b viene visualizzato il modulo della trasformata di Fourier della finestra. Nella Figura 5.23c viene mostrato il filtro ideale nel dominio del tempo e nella Figura 5.23d lo stesso nel dominio delle frequenze. Nella Figura 5.23e viene visualizzato il filtro nel dominio del tempo. In Figura 5.23f viene visualizzato il filtro nel dominio delle frequenze. Possiamo notare dalla Figura 5.23f che l'ampiezza della banda passante in questo caso è $11\pi/M$. Inoltre l'ampiezza del lobo laterale rispetto alla risposta in frequenza del filtro è di 74 dB.

Quale finestra scegliere

Dopo aver con concluso la discussione sulle diverse finestre possiamo riassumere i vari risultati ottenuti. I risultati sono descritti nella Tabella di Figura 5.24. Facendo un confronto tra le varie finestre scaturisce che cambiando il tipo di finestra cambia l'attenuazione minima in banda oscura. L'attenuazione minima assoluta è ottenuta usando la finestra di Blackman. Comunque i molte applicazioni la scelta migliore ricade sulla finestra di

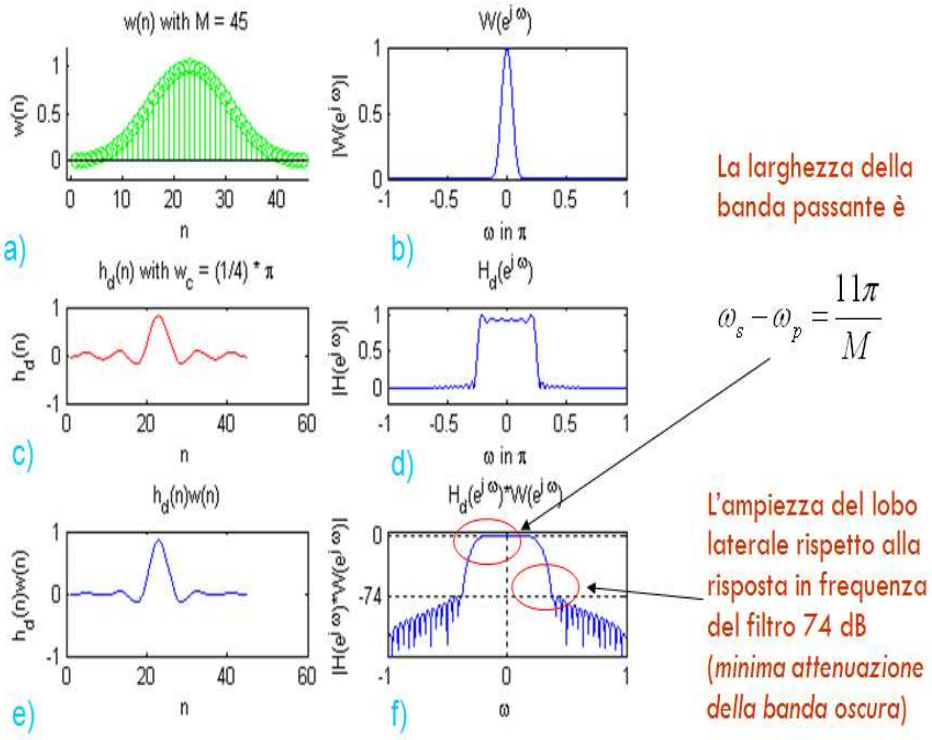


Figura 5.23: Filtraggio FIR con finestra di Blackman.

Finestra	*Altezza masima dei lobi laterali (dB)*	*Larghezza del lobo principale*	*Attenuazione minima in banda oscura (dB)*
Rettangolare	*-13*	*4π/N*	*-21*
Bartlett	*-25*	*8π/N*	*-25*
Hanning	*-31*	*8π/N*	*-44*
Hamming	*-41*	*8π/N*	*-53*
Blackman	*-57*	*12π/N*	*-74*

Figura 5.24: Tabella riassuntiva del filtraggio con varie finestre.

Hamming in quanto presenta un buon compromesso tra larghezza di banda di transizione e attenuazione nella banda oscura.

Altri tipi di filtri

Una delle caratteristiche principali dei filtri basati sul windowing risiede nella semplicità con cui è possibile usare un filtro passa-basso per ottenre un altro tipo di filtro. In Figura 5.25 vengono visualizzate varie tipologie di filtro. In dettaglio abbiamo

- il filtro passa-basso (a)
- il filtro passa-alto (b)
- il filtro elimina-banda (c)
- il filtro passa-banda (d)

Il filtro passa-banda permette di selezionare tutte le frequenze all'interno dell'intervallo identificato da due frequenze di taglio. Il filtro passa-banda si ottiene dalla somma di due filtri passa-basso come mostrato in Figura 5.27.
Nel codice seguente viene descritto il metodo **passa-banda** che implementa il filtro passa-banda ottenuto da due filtri passa-basso

```
void passa_banda() {
   cout <<" \n \n FILTRO PASSA-BANDA \n \n";
   double fp,fs;
   cout <<"Inserisci la frequenza di taglio della banda passante";
```

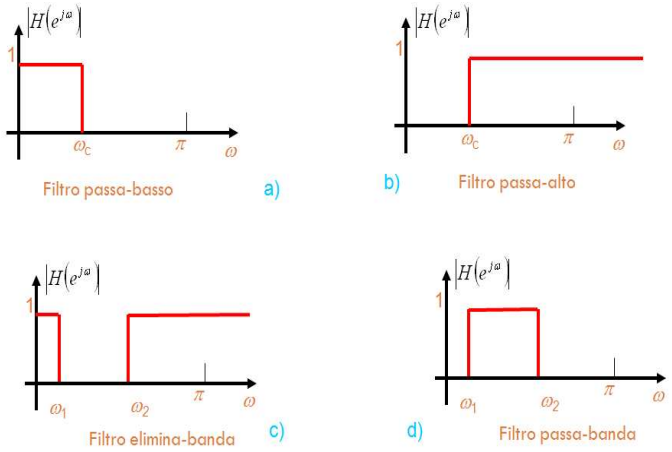


Figura 5.25: Tipologie di filtro: a) passa-basso; b) passa-alto; c) elimina banda; c) passa-banda.

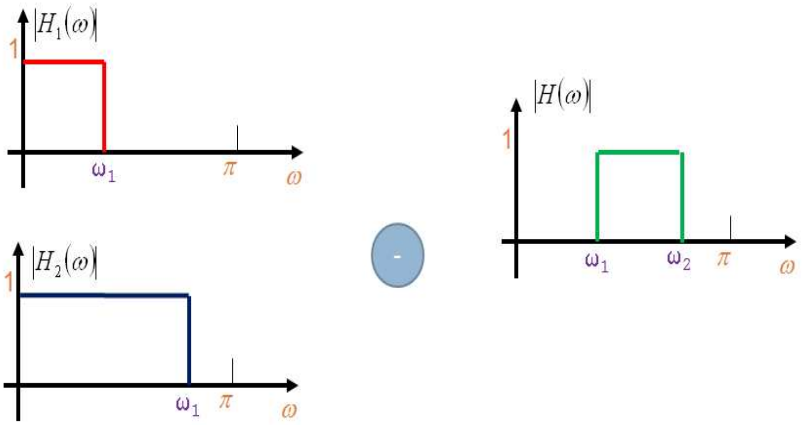


Figura 5.26: Filtro passa-banda.

```
    cin >> fp;
    cout <<"Inserisci la frequenza di taglio della banda stop";
    cin >> fs;
    // trasformazione delle frequenze di taglio in pulsazioni
    double wc_1 = (double)fp/fc * 2 * PI;
    double wc_2 = (double)fs/fc * 2 * PI;
    // risposta all'impulso - seno cardinale
    sinc<double> ir(wc_1,M);
    sinc<double> ir_1(wc_2,M);
    // Operazione di differenza tra le due risposte all'impulso.
    ir.sottrazione(&ir_1);
    window w(M);
    // Applicazione della finestra precedentemente scelta
    switch(sc_w)
    {
    case 1:  ir.prodotto(&w); break;
    case 2:  w.ham();   ir.prodotto(&w); break;
    case 3:  w.han();   ir.prodotto(&w); break;
    case 4:  w.triang();    ir.prodotto(&w);  break;
    case 5:  w.black();   ir.prodotto(&w);  break;
    case 6:  w.harris();   ir.prodotto(&w); break;
    }
    x_ir = ir.get_x();
    // Filtraggio tramite convoluzione
    conv(ir.get_x());
    printf("** \n");
};
```

Il filtro passa-alto, invece, ha lo scopo di filtrare le frequenze al disotto di una frequenza di taglio. Esso può essere implementato mediante la sottrazione di due filtri passa-basso come in Figura 5.27. Il metodo **passa-alto** che implementa il filtro passa-alto ottenuto da due filtri passa-basso è presentato nel seguente codice

```
void passa_alto() {
    cout <<" \n \n FILTRO PASSA-ALTO \n \n";
    double fp;
    cout <<"Inserisci la frequenza di taglio della banda passante";
    cin >> fp;
     // trasformazione della frequenza di taglio in pulsazione
```

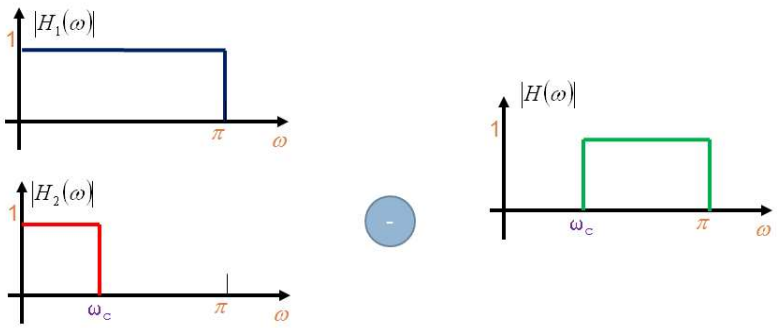


Figura 5.27: Filtro passa-alto.

```
    double wc_1 = (double)fp/fc * 2 * PI;
    // risposta all'impulso - seno cardinale
    sinc<double> ir(wc_1,M);
    sinc<double> ir_1(PI,M);
   // Operazione di differenza tra le due risposte all'impulso.
    ir.sottrazione(&ir_1);
    window w(M);
    switch(sc_w)
    {
    case 1:  ir.prodotto(&w); break;
    case 2:  w.ham();   ir.prodotto(&w); break;
    case 3:  w.han();   ir.prodotto(&w); break;
    case 4:  w.triang();    ir.prodotto(&w);  break;
    case 5:  w.black();   ir.prodotto(&w);  break;
    case 6:  w.harris();   ir.prodotto(&w); break;
    }
    x_ir = ir.get_x();
    // Filtraggio tramite convoluzione
    conv(ir.get_x());
    printf("** \n");
};
```

Il filtro elimina-banda ha lo scopo di eliminare le frequenze contenute nel range identificato da due frequenze di taglio. In Figura 5.28 viene presentato il filtro elimina-banda ottenuto dall'operazione somma e sottrazione di 3 filtri passa-basso. Nel codice seguente viene presentato il metodo **banda-stop** che implementa il filtro elimina-banda ottenuto usando tre filtri passa-basso

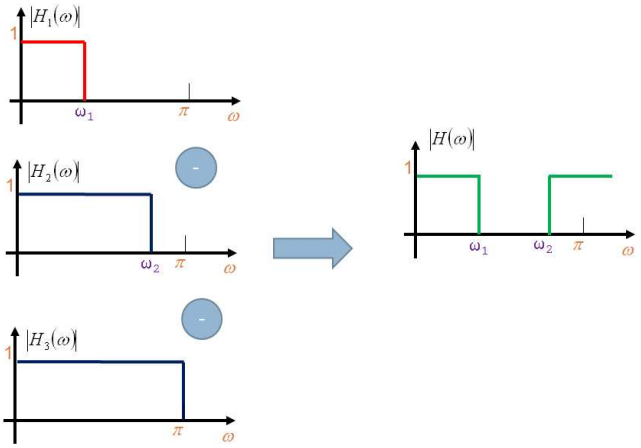


Figura 5.28: Filtro elimina-banda.

```
void banda_stop() {
    cout <<" \n \n FILTRO BANDA-STOP \n \n";
    double fp,fs;
    cout <<"Inserisci la frequenza di taglio della banda passante";
    cin >> fp;
    cout <<"Inserisci la frequenza di taglio della banda stop";
    cin >> fs;
     // trasformazione delle frequenze di taglio in pulsazioni
    double wc_1 = (double)fp/fc * 2 * PI;
    double wc_2 = (double)fs/fc * 2 * PI;
    // risposta all'impulso - seno cardinale
    sinc<double> ir(wc_1,M);
    sinc<double> ir_1(wc_2,M);
    sinc<double> ir_2(PI,M);
    // Operazioni di somma e differenza tra le tre risposte all'impulso.
    ir.somma(&ir_2).sottrazione(&ir_1);
    window w(M);
    switch(sc_w)
    {
    case 1:  ir.prodotto(&w); break;
    case 2:  w.ham();   ir.prodotto(&w); break;
    case 3:  w.han();   ir.prodotto(&w); break;
    case 4:  w.triang();    ir.prodotto(&w);  break;
    case 5:  w.black();   ir.prodotto(&w);  break;
```

```
      case 6:  w.harris();   ir.prodotto(&w); break;
      }
      x_ir = ir.get_x();
       // Filtraggio tramite convoluzione
      conv(ir.get_x());
      printf("** \n");
  };
```

5.5.5 Altri tipi di progetti

E' possibile comunque usare altri tipi di progetti per l'implementazione di un filtro numerico FIR. Possiamo ricordare il *metodo di campionamento* e i metodi delle *oscillazioni uniformi* (Herrmann e Schuessler, Parks e McClellan).

Capitolo 6

Compressione

In questi ultimi anni si è avuta una crescente diffusione delle applicazioni multimediali. Alcuni effetti hanno comportato la necessità di una quantità enorme di spazio di memoria e una maggiore larghezza di banda su Internet. Per questo motivo sono nati diversi formati per la rappresentazione e/o compressione dei dati. Nelle tecniche di compressione audio si cerca di garantire una riduzione del volume dei dati con una distorsione limitata dell'informazione. Questo Capitolo sarà incentrato sulle tecniche di rappresentazione del un segnale audio e in particolare sulle tecniche di compressione oggi più usate.

6.1 Tecniche di rappresentazione

Come abbiamo visto nel Capitolo 2 un segnale audio analogico viene convertito in un segnale digitale secondo il seguente schema

- campionamento
- quantizzazione

Consideriamo ad esempio un segnale audio analogico campionato ad una frequenza di 8000 Hz e ciascun campione è quantizzato con 8 bit. In questo caso il segnale digitale che ne risulta avrà un bit-rate di 64.000 bps. Per ridurre l'informazione possiamo ridurre il bit-rate e quindi potremmo campionare ad un tasso minore o usare un numero minore di bit. Questo comporta comunque una qualità scadente del segnale audio. La tecnica che viene usata è invece quella della compressione che ci permette di risparmiare in quantità di dati ma con una qualità dell'audio elevata.

Prima di descrivere le tecniche più usate nel campo della compressione vediamo quali sono i metodi più comuni per la rappresentazione di un segnale audio digitale.

6.1.1 Wave, AIFF

Sono i formati più semplici e sono creati rispettivamente da Microsoft-IBM e Apple. Si basano sulla tecnica PCM introdotta nel Capitolo 2. Possiamo considerarla come una registrazione in formato digitale fedele ai suoni originali analogici.

6.1.2 Midi

E' uno standard per la comunicazione di strumenti musicali. Un file *.mid* può essere visto come uno spartito interpretato da un *sequencer*. Il file midi contiene una sequenza di comandi che indicano quale nota far suonare, da quale strumento, con quale intensità e per quanto tempo. Comporta un notevole risparmio di spazio.

6.1.3 Moduli musicali: MOD, XM, IT, S3M

Questi formati audio consentono di coniugare gli aspetti positivi del campionamento e della modulazione. I file dei moduli contengono le informazioni per l'esecuzione audio: note, tempi, strumenti. Occupano poco spazio ma sono memorizzati anche i veri campioni degli strumenti originari (suono e timbro). Il suono di una chitarra in un MOD risulta reale e fedele al suo timbro. Questi file possono essere usati solo in casi circosritti.

6.1.4 Streaming audio: RAM, RM, ASF, ASX

Questi formati vengono usati generalmente per lo *streaming*. Lo streaming è il meccanismo di trasferimento in rete dei dati audiovisivi in tempo reale. Nel caso dello streaming non viene scaricato l'intero file ma viene avviata la riproduzione dopo la ricezione del primo pacchetto. Per ascoltare un file audio in streaming è necessario un player che è dotato di bufferizzazione.

6.1.5 DAB

E' un sistema di trasmissione di segnali radio digitali via etere. Il DAB si basa su un algoritmo di compressione audio (simile all' MP3). La qualità di trasmissione è variabile secondo la banda occupata. Durante la trasmissione sono usati i codici CRC per correggere gli errori.

6.2 Modulazione a codifica di impulso

La modulazione a codifica di impulso (o PCM, Pulse Code Modulation), è la tecnica base per il campionamento e la quantizzazione di un segnale (vedi Capitolo 2 per maggiori dettagli su questa tecnica). Ad esempio i CD audio usano PCM con un tasso di 44.100 campioni al secondo e 16 bit per campione. Questo porta ad un bit-rate di $705,6$ Kbps per un segnale audio mono e di 1411 Mbps per uno stereo a due tracce.
La tecnica PCM consente di campionare un segnale audio ad un tasso minimo che corrisponde a due volte la massima frequenza del segnale (Teorema di Campionamento o Nyquist, Capitolo 2). Se la larghezza di banda di un canale di comunicazione è minore di quella del segnale, il campionamento viene determinato in base alla larghezza di banda del canale (questi vengono chiamati *segnali limitati in banda*).
Come esempio possiamo considerare la codifica PCM di un segnale vocale. In questo caso la massima frequenza è 4 KHz e per cui il minimo tasso di campionamento è 8 KHz. Per l'audio in generale, come la musica, invece abbiamo che la massima frequenza è 20 kHz e il tasso è 40 kHz. Inoltre abbiamo che il numero di bit per la quantizzazione può variare. Generalmente abbiamo 12 bit per il parlato e 16 bit per l'audio in generale. Sostanzialmente questo produce un bit rate di 4240 Kbps per il parlato e 1.28 Mbps per file audio generali.
Per ridurre l'informazione possiamo ad esempio campionare ad un rate minore o usare un numero minore di bit ottenendo una qualità bassa del segnale audio. Come vedremo invece la tecnica più usata è quella della compressione.

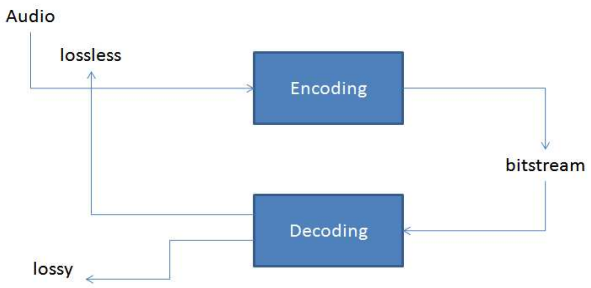


Figura 6.1: Schema di compressione e decompressione.

6.3 Schema di compressione

Le operazioni di *compressione* (o *encoding*) e *decompressione* (*decoding*) del segnale audio si possono riassumere come nello schema in Figura 6.1. Durante la fase di compressione, i dati audio digitali sono campionati, *codificati* e trasformati una sequenza di bit (bitstream) in modalità compressa rispetto all'originale. Durante la fase di decompressione il segnale viene *decodificato* e si ricostruisce il segnale. Il segnale risultante è identico a quello di partenza e quindi la nostra compressione risulta senza perdita (*lossless*) oppure il segnale ricostruito risulta un segnale "simile" all'originale e quindi abbiamo una compressione con perdita (*lossy*).
Gli schemi di compressione includono più tipi di elaborazione del segnale

- *Eliminazione dei dati ridondanti* - come ad esempio Run Length Encoding (RLE), Huffman, LZ(W) in software comuni come WinZip e StuffIt
- Basati su *principi di tipo percettivo* - come ad esempio MPEG 1 Layer3 (mp3), PASC

6.4 Codifiche μ-law e A-law

Le codifiche μ-law e A-law sono entrambe applicate nella telefonia. Esse realizzano le specifiche contenute nella raccomandazione G.711 rilasciata dal CCITT, un comitato internazionale per gli standard nelle comunicazioni. La codifica μ-law è in uso in Nord America e Giappone per i servizi di telefonia digitale su linee ISDN. La codifica A-law è invece in uso in Europa e sul traffico internazionale ISDN. La linea ISDN consente una trasmissione di 64 Kbps e il tasso di campionamento è di 8000 campioni/sec.

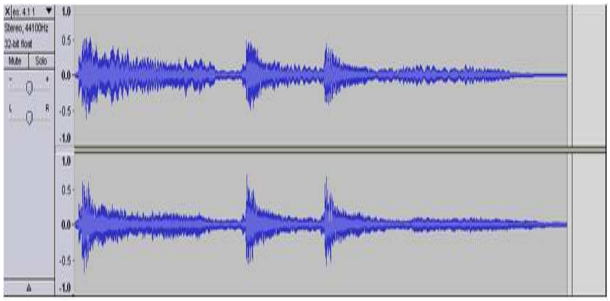


Figura 6.2: Segnale audio.

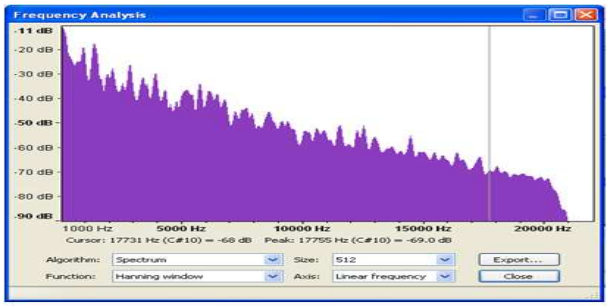


Figura 6.3: Spettro di potenza del segnale originale.

Vediamo il risultato di questo tipo di compressione su un segnale audio stereo (vedi Figura 6.2). Per capire la differenza tra il segnale originale e il segnale compresso ricorriamo alle caratteristiche degli stessi nel dominio delle frequenze. Consideriamo lo spettro di potenza del segnale originale in Figura 6.3 e di quello compresso con μ-law in Figura 6.4. Possiamo notare che il segnale originale ha un contenuto frequenziale notevole che va oltre i 20 KHz rispetto a quello compresso.
La caratteristica principale comune ai due approcci è quella dell'utilizzo della quantizzazione del tipo logaritmica che prevede 8 bit per campione.

6.5 Differential Pulse Code Modulation

Una semplice tecnica di compressione è quella della Differential Pulse Code Modulation (DPCM) che deriva direttamente dallo standard PCM. L'approccio si basa sull'idea di memorizzare le differenze tra i campioni invece che i campioni stessi. Come possiamo immaginare in questo modo abbiamo bisogno di un numero minore di bit per la rap-

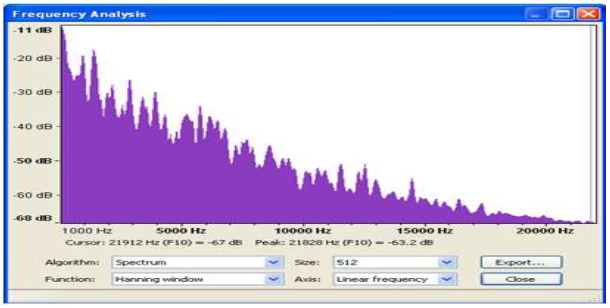


Figura 6.4: Spettro di potenza del segnale compresso con la tecnica μ-law.

presentazione di una sequenza. Sostanzialmente ciò che accade è che per due elementi della sequenza viene considerata la differenza tra i due. L'informazione da memorizzare è identificata da un bit di segno e da alcuni bit che rappresentano il valore assoluto della differenza. Per calcolare le differenze vengono usati dei registri temporanei. La fase del calcolo della differenza è chiamata di *timing*. Essenzialmente il campione del segnale analogico di input viene immagazzinato nel registro temporaneo R_0. Le differenze (DPCM) sono ottenute sottraendo i contenuti correnti del registro R_0 dal nuovo campione PCM. Il valore nel registro è quindi aggiornato aggiungendo al corrente contenuto il DPCM calcolato. Dall'altra parte il decodificatore semplicemente addiziona il DPCM al segnale calcolato in precedenza immagazzinato nel registro PCM. Nel circuito in Figura 6.5 viene rappresentato lo schema generale di un codificatore DPCM. Il primo passo che viene effettuato è quello dell'applicazione di un filtro passa-banda (anti-aliasing). Viene applicato successivamente un covertitore da Analogico a Digitale per ottenere i campioni su cui viene effettuato il calcolo dei coefficienti DPCM tramite l'utilizzo di un registro. Dopo avere fatto la conversione da parallelo a seriale il segnale viene immesso sulla rete nel formato *bitstream*.

Nel circuito di Figura 6.6, invece, viene rappresentato lo schema di un decodificatore DPCM. Il bitstream ricevuto dalla rete viene convertito da seriale in una sequenza in parallelo. Viene applicata la procedura di ripristino da DPCM a PCM e successivamente convertito in un segnale analogico. L'ultima fase è quella dell'applicazione di un filtro passa-basso. Dobbiamo comunque notare che nella codifica DPCM è inevitabile il fenomeno dello *slope overload*, cioè che differenze elevate (nel caso di alte frequenze) non si possono rappresentare con un numero piccolo di bit. Gli errori introdotti porterebbero

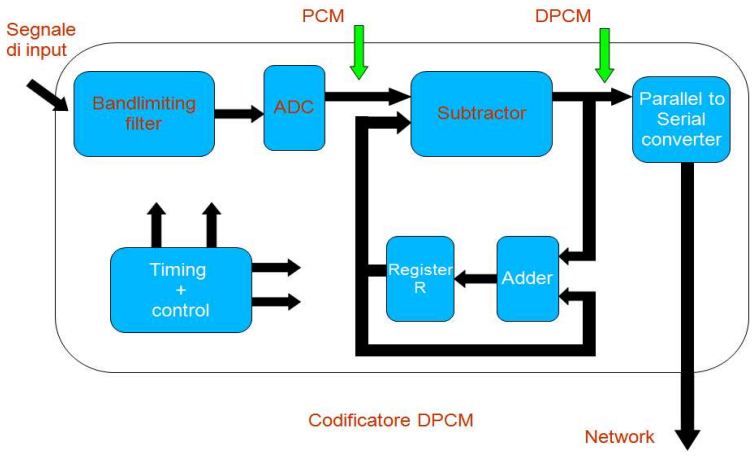


Figura 6.5: Codificatore DPCM.

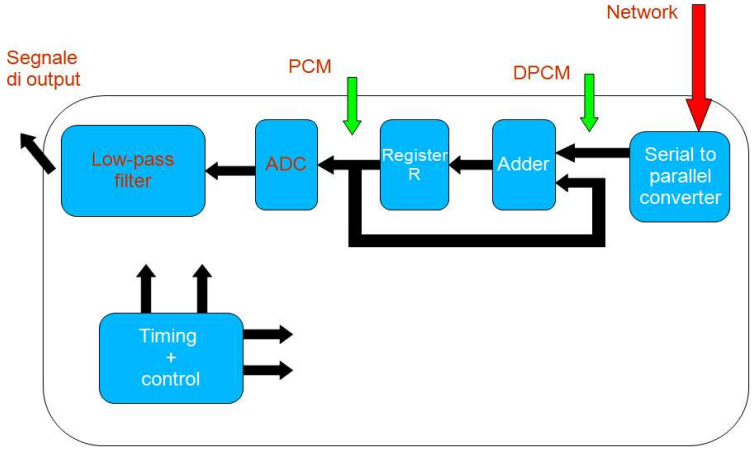


Figura 6.6: Decodificatore DPCM.

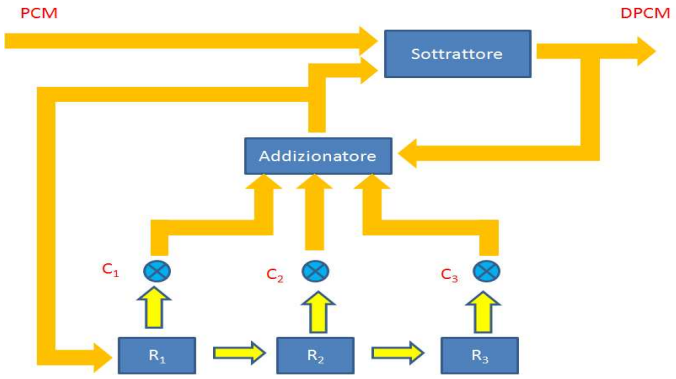


Figura 6.7: DPCM Predittivo.

a distorsioni sulle alte frequenze.

6.6 DPCM predittivo

Il problema maggiore dello schema DPCM deriva dallo slope overload inerente all'accuratezza delle differenze calcolate nella fase di timing. Il segnale residuo, infatti, ad ogni istante è determinato dall'accuratezza del valore precedente immagazzinato nel registro. Una possibile soluzione è quella di applicare una tecnica di tipo predittivo. Lo schema generale risulta identico a quello usato nel caso del DPCM. In Figura 6.7 viene rappresentata la parte relativa al timing. Lo schema utilizza più registri R e differenti coefficienti C per predire il valore del segnale. L'obiettivo è quello di avere una stima più accurata del campione della sequenza che deriva dalle stime precedentemente effettuate.

6.7 DPCM Adattivo

Un ulteriore miglioramento al DPCM può essere ottenuto variando il numero di bit usati per le differenze in modo che essi dipendano dalla sua ampiezza. L'idea, quindi, potrebbe essere quella di usare pochi bit per codificare piccole differenze e più bit per codificare grandi differenze. Questo è il principio del DPCM adattivo (adaptive differential PCM). Esistono diversi standard ADPCM definiti nelle raccomandazioni ITU-T G.721, G.722 e G.726. In Figura 6.8 viene presentato lo schema di un codificatore DPCM (è semplice ottenere il decodificatore) secondo lo standard G. 722.

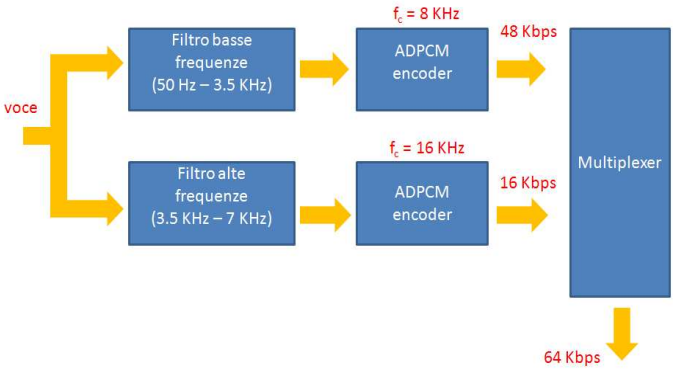


Figura 6.8: Decodificatore ADPCM.

Vediamo il suo funzionamento. Supponiamo di avere un segnale vocale. Prima di tutto esso viene filtrato prima di essere campionato. Nell'architettura ci sono principalmente due filtri passa-banda. Idealmente un segnale è scomposto in due bande: *sottobanda bassa* e *sottobanda alta*. Lo scopo è quello di filtrare le frequenze in due intervalli: 50 Hz - 3.5 kHz e 3.5- 7 kHz. Ogni banda viene elaborata indipendentemente dall'altra usando l'approccio ADPCM. Il vantaggio di usare due distinte bande è quello di poter usare due differenti bit rate. Infatti si può notare che le componenti di frequenza che sono presenti nella banda bassa del segnale hanno un'importanza maggiore. Ad esempio con un frequenza di campionamento di 8 KHz la banda bassa può avere un bit rate di 48 Kbps e la banda alta con una frequenza di campionamento di 16 KHz e un bit rate di 16 Kbps. Il bit rate totale risulta di 64 Kbps.

6.8 IMA ADPCM

L'IMA (Interactive Multimedia Association) è un consorzio di produttori hardware e software per lo sviluppo di standard per i dati multimediali. Lo schema IMA ADPCM produce un rapporto discreto di compressione ed è diffuso in ambiente Windows. ADPCM di Apple è chiamato ACE/MACE. Il predittore non è adattivo e la quantizzazione è adattiva in modo logaritmico. I passi di quantizzazione sono contenuti in una tabella a 89 voci.

6.9 Linear Predictive Coding (LPC)

Un approccio alternativo alla compressione basata su PCM è il *Linear Predictive Coding*. L'idea base è quella di selezionare le caratteristiche percettive del segnale stesso. Una volta quantizzate, le caratteristiche vengono inviate alla destinazione che le usa insieme ad un sintetizzatore per rigenerare un suono che risulta percettivamente comparabile con il segnale audio sorgente. La chiave di questo approccio è quella di identificare un insieme di caratteristiche percettive. Ad esempio se consideriamo la voce le tre caratteristiche di percezione del segnale all'orecchio umano sono

- *il pitch* - esso è correlato alla frequenza del segnale e risulta importante in quanto l'orecchio è più sensibile alle frequenze nell'intervallo $2-5$ kHz rispetto alle altre frequenze.
- *periodo* - la durata del segnale
- *loudness* - determinata dall'ammontare di energia nel segnale.

Inoltre sono importanti anche i parametri di eccitazione del tratto vocale e sono classificati come

- *Voce espressa* - queste sono generate dalle corde vocali come ad esempio i suoni relativi alle lettere m,v, l
- *Voce inespressa* - le corde vocali sono aperte ed esempi includono i suoni delle lettere f e s.

Una volta che questi parametri sono ottenuti in un *codificatore* è possibile usarli, insieme ad un modello del tratto vocale per generare una versione sintetizzata del segnale vocale originario, nel *decodificatore*.

6.10 Codifica percettiva

In questi ultimi anni, con l'avvento di internet, si è avuto un enorme incremento di tecniche di compressione. La maggior parte delle tecniche moderne sono basate su principi di tipo percettivo. Le *codifiche percettive* sono state introdotte per la compressione di audio in

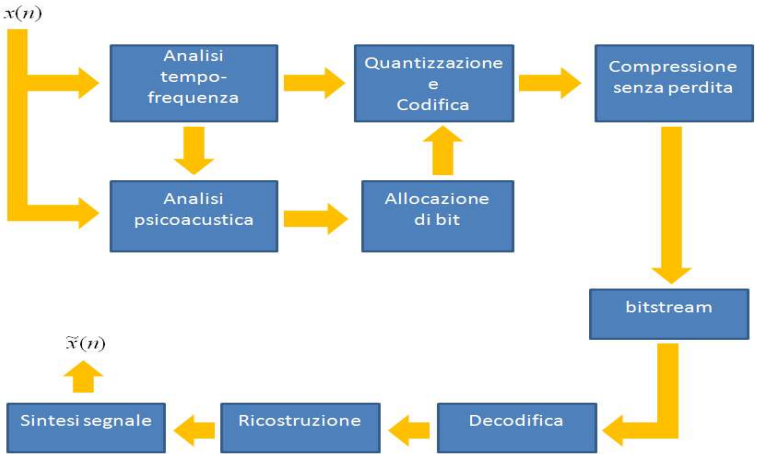


Figura 6.9: Sitema di compressione percettiva.

diversi settori come ad esempio nell'ambito della televisione digitale e la musica. Riprendono principalmente le caratteristiche del modello psicoacustico dell'orecchio umano. Con questo approccio i segmenti dell'audio sono analizzati ma solo le caratteristiche che sono percettibili dall'orecchio sono trasmesse. Sebbene l'orecchio umano è sensibile a segnali con frequenze contenute nell'intervallo 15 Hz - 20 KHz il livello di sensitività di ogni segnale è non lineare. In altre parole l'orecchio è più sensibile ad alcuni segnali che ad altri. Nella Figura 6.9 viene rappresentato lo schema generale di un sistema di compressione audio di tipo percettivo. Un segnale di input $x(n)$ viene segmentato in frame quasi-stazionari (analisi nel tempo - 2 a 50 millisecondi). I singoli blocchi vengono analizzati nel dominio della frequenza. La trasformazione avviene con una trasformata unica o banchi di filtri. Il segnale viene acnhe sottoposto ad un'analisi psicoacustica che usa una FFT a 512 o 1024 punti. L'analisi psicoacustica produce le soglie di mascheramento per filtrare le componenti non udibili durante la fase di ri-quantizzazione. In particolare calcola il rapporto **SMR** (Signal-to Mask Ratio) tra ampiezza del segnale e la soglia di mascheramento per decidere successivamente quanti bit allocare e per ridurre il rumore di quantizzazione. La fase di quantizzazione e codifica usa i bit allocati per stabilire le dimensioni delle regioni di quantizzazione e codificare il segnale. Le ridondandanze ancora presenti vengono rimosse mediante schemi di compressione semplici di tipo *lossy* (DPCM e ADPCM) o *lossless* (codice di Huffman). La codifica elaborata viene formattata per ottenere il *bitstream*. Nella fase di decodifica si "scompatta" il bitstream e i campioni di frequenza vengono

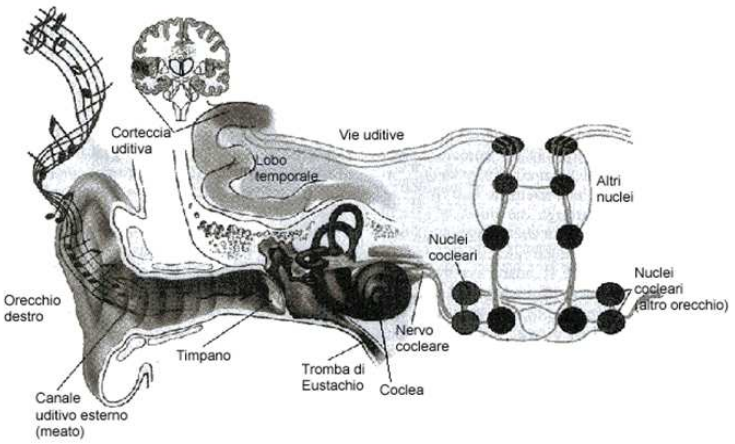


Figura 6.10: Sistema uditivo.

trasformati in un segnale audio nel tempo.
Per capire il funzionamento degli algoritmi di compressione basati sulla codifica percettiva, successivamente approfondiamo la percezione uditiva dell'orecchio umano.

6.10.1 Percezione uditiva

Come abbiamo visto nei Capitoli precedenti i suoni sono vibrazioni in un mezzo che possono essere descritte mediante dei parametri fisici ampiezza, frequenza, forma d'onda. La sensazione uditiva dipende dalla combinazione tra il fenomeno fisico e il nostro sistema percettivo. Ancora oggi, molti aspetti del nostro sistema uditivo, soprattutto le funzioni superiori legate alla parte cognitiva del processo neurale, sono in gran parte sconosciuti. Non è chiaro, infatti, come i segnali provenienti dalle due orecchie vengono miscelati ed elaborati per produrre la sensazione di musica e linguaggio o per la localizzazione precisa degli oggetti.
L'orecchio si può suddividere in tre sezioni principali (vedi Figura 6.10):

- *orecchio esterno* (padiglione, meato, timpano)
- *orecchio medio* (ossicini)
- *orecchio interno* (coclea)

Come possiamo notare, il punto di ingresso di un segnale audio è dato dall'orecchio. La forma del padiglione è molto complessa. Lo scopo è di offrire un'antenna per la cattura

dei segnali ad alta frequenza. Le pieghe del padiglione auricolare "colorano" i suoni ad alta frequenza mediante l'interferenza tra segnali riflessi da pieghe differenti. Successivamente, il segnale in ingresso mette in vibrazione la membrana del timpano che trasmette il movimento meccanico agli *ossicini*. Gli ossicini sono tre con le seguenti caratteristiche

- *Martello*, connesso al timpano
- *Incudine*, propaga il movimento
- *Staffa*, connessa alla coclea

Il meccanismo degli ossicini permette di trasformare una vibrazione su un'ampia superficie in una vibrazione della stessa energia ma distribuita su una piccola superficie. Questo avviene secondo il principio della leva, che si applica in conseguenza della differente lunghezza dei due bracci rispetto al fulcro, la forza che agisce sulla finestra ovale viene amplificata permettendo la percezione anche di segnali deboli.
I due elementi principali dell'orecchio interno sono i *canali semicircolari* (compiti di equilibrio) e la *coclea*.
La coclea è un tubo di 3.5 cm avvolto a forma di chiocciola ed è il cuore della percezione uditiva. La parte fondamentale della coclea è la *membrana basilare*. Essa occupa la parte centrale della coclea per tutta la sua lunghezza. E' stretta e leggera vicino agli ossicini (base) e aumenta il suo spessore verso l'altra estremità (apice) dove si trova l'elicotrema che permette di farla comunicare direttamente con la rampa timpanica. Lo spessore variabile della membrana corrisponde a una differente risonanza, per cui vibra in risposta a frequenze differenti lungo la sua lunghezza. L'analisi spettrale sulla membrana non segue una scala lineare sulle frequenze. E' possibile identificare delle bande di frequenza dette bande critiche. Tra le bande si osserva un cambiamento repentino della percezione uditiva. Il sistema uditivo, quindi, si può descrivere come un banco di filtri passa-banda con circa 25 *bande critiche*. Le frequenze sono distribuite ordinatamente sulla membrana e possiamo associarla ad un analizzatore di Fourier. All'estremità vicina agli ossicini vengono percepite le frequenze alte (stretta, rigida, leggera) invece le frequenze basse sono identificate all'estremità interna (ampia, flessibile, massiccia).
All'arrivo di un suono si ottiene un punto sulla membrana. La frequenza che presenta il picco è detta *frequenza caratteristica*. Dall'ampiezza della vibrazione scaturisce l'intensità dei suoni.

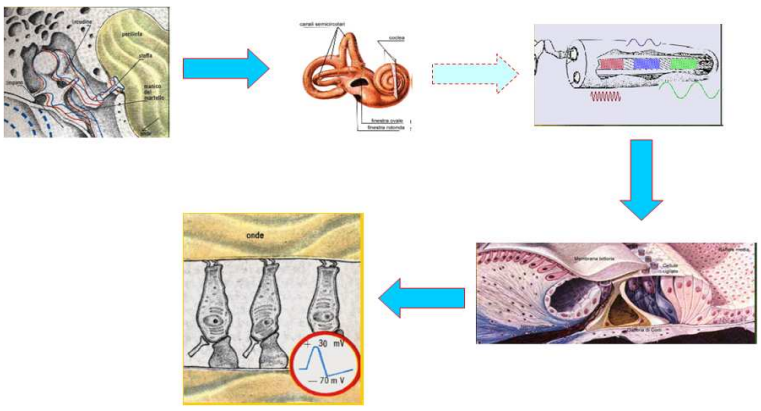


Figura 6.11: Processo di percezione uditiva.

A questo punto, le cellule cigliate dell'*organo dei Corti* trasducono le vibrazioni meccaniche in impulsi nervosi elettrochimici. In dettaglio si ha che le cellule cigliate vengono stimolate dalle vibrazioni della membrana basilare. Quando le ciglia delle cellule cigliate si flettono, i canali degli ioni si aprono, e il voltaggio all'interno varia abbastanza da causare il rilascio di *neurotrasmettitori* alla giunzione tra la cellula cigliata e la *sinapsi* del *nervo uditivo*.

Infine i neurotrasmettitori vengono assorbiti dalle fibre nervose uditive stimolando l'invio di un segnale elettrico lungo il *nervo cocleare*.

6.10.2 Diagramma di Fletcher-Munson

Il diagramma di Fletcher e Munson risale agli anni 40 ed è il risultato di un'indagine empirica condotta su un vasto campione di cittadini americani. Il diagramma è rappresentato in Figura 6.12. Sull'ascissa troviamo le frequenze dell'udibile da 20 a 20.000 Hz e sull'ordinata vengono rappresentati i valori di intensità sonora da 0 a 140 dB. Nel punto (x, y) del diagramma viene quindi rappresentato un tono puro di frequenza x Hz a un'intensità di y dB.

Tutti i suoni che nel diagramma si trovano al di sotto della isofona a 0 foni (soglia dell'udito) non verranno uditi e possono essere eliminati senza introdurre alcuna distorsione del segnale.

La percettività del suono non è costante nel tempo, ma varia in funzione di ciò che as-

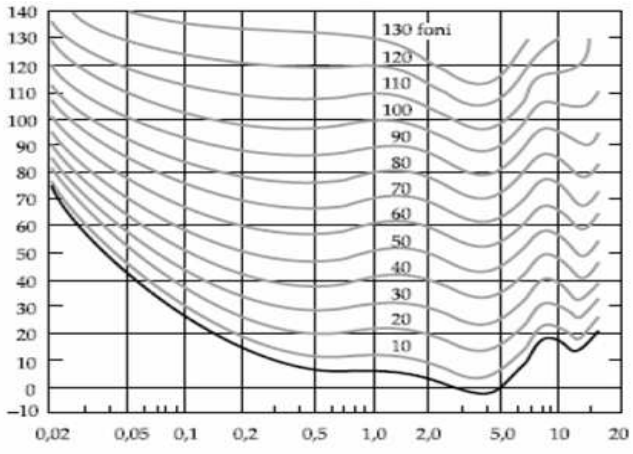


Figura 6.12: Diagramma di Fletcher e Munson.

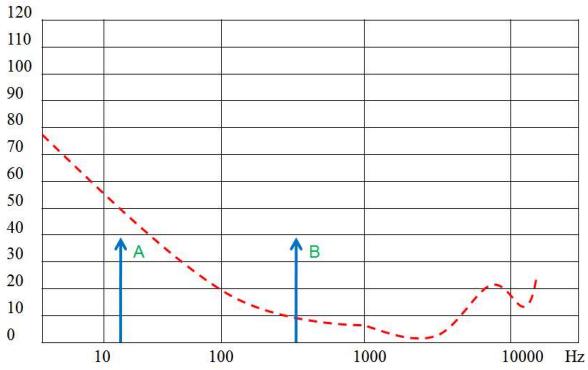


Figura 6.13: Soglia di udibilità e toni.

coltiamo. L'orecchio è molto sensibile ai segnali con frequenza compresa tra 2 e 5 KHz. In Figura 6.13 vengono rappresentati due toni a diversa frequenza ma con uguale intensità. Mentre il tono A è sotto soglia e quindi non è percepito il tono B è udibile essendo sopra la soglia di udibilità.

6.10.3 Mascheramento

Il nostro apparato uditivo si comporta come un analizzatore di Fourier. In altre parole, percepisce le componenti individuali di un suono e le distribuisce lungo la membrana basilare della coclea con un funzionamento tonotopico. Vediamo in dettaglio come funziona la membrana basilare (Figura 6.14). Un fenomeno psicoacustico che riscontriamo all'interno

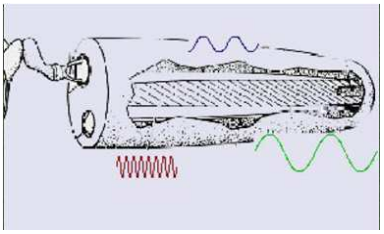


Figura 6.14: Membrana basilare.

Figura 6.15: Bande critiche.

della membrana basilare è quello del *mascheramento*. Questo fenomeno si presenta quando un segnale forte maschera un segnale debole. Vediamo in dettaglio questo fenomeno all'interno della membrana basilare. L'ampiezza di banda con cui lavorano i filtri uditivi viene chiamata *banda critica*. I suoni possono essere discriminati perfettamente quando ricadono in differenti bande critiche. La larghezza di banda tende a rimanere constante fino a 500 Hz (minore di 100 Hz). Oltre i 500 Hz le bande diventano più larghe 20% della frequenza centrale della banda. L'intera gamma di frequenze viene partizionata in 25 bande. L'unità di misura fondamentale è il *bark* (dal nome dello studioso Barkhausen).

Il fenomeno si presenta quando un segnale a debole intensità (mascherato) non viene percepito a causa della presenza simultanea di un segnale di intensità superiore (mascheratore). Questo tipo di mascheramento viene chiamato *mascheramento simultaneo*. Ad esempio in Figura 6.16 il segnale A modifica la soglia di udibilità in modo tale che il segnale B non può essere percepito.

Inoltre è possibile che si verifichi anche un *mascheramento temporale*. L'orecchio, infatti, non ha tempi di reazione nulli e quindi impiega un certo tempo per adattarsi a nuove condizioni e impiega tempo a tornare in quiete dopo una sollecitazione (vedi Figura 6.17). Ad esempio se si suona un tono di 1000 Hz a 60 dB, un tono da 1100 Hz a 40 dB è mascherato per un tempo che va oltre la fine del tono da 1000 Hz. Nel *pre-mascheramento* si verifica che un suono per essere percepito necessita di un certo tempo in cui si mantiene

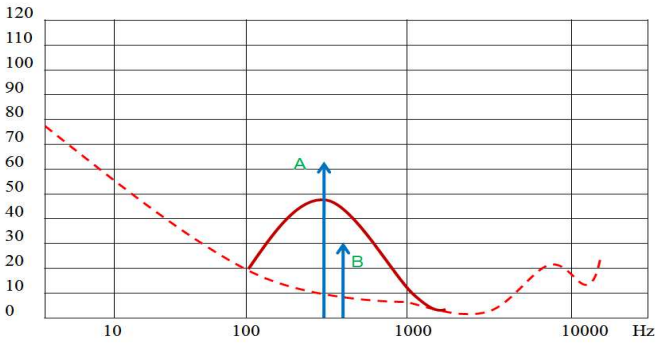


Figura 6.16: Mascheramento simultaneo.

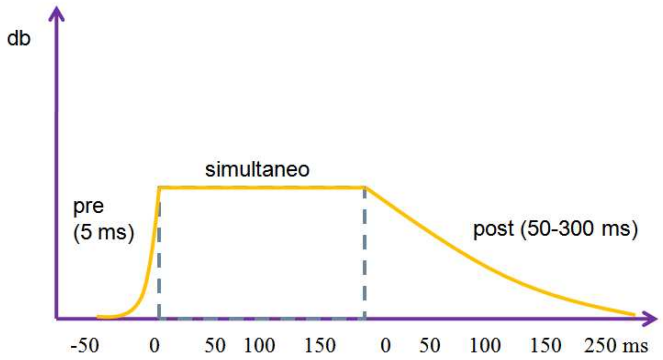


Figura 6.17: Mascheramento temporale.

senza disturbi. Supponiamo, infatti, di avere un suono e prima che l'orecchio lo distingua chiaramente interviene un'altra sollecitazione più forte, allora il primo suono non può essere più rilevato.

6.10.4 Mascheramento e bit

Le nozioni introdotte fino ad ora sulla banda critica e sul mascheramento sono alla base dei moderni algoritmi di compressione audio. In Figura 6.18 viene rappresenta un tono di mascheramento e il relativo rapporto segnale rumore (**SNR**) utilizzando m bit. Il tono di mascheramento genera una certa soglia di mascheramento (linea rossa) di cui il minimo si trova all'estremo sinistro della banda critica. Per lavorare sicuro rispetto all'introduzione di distorsioni occorre basarsi esattamente su tale soglia minima. In figura

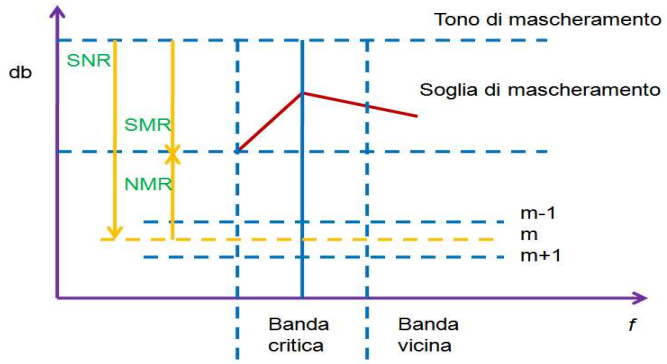


Figura 6.18: Mascheramento simultaneo e numero di bit.

vengono riportati i possibili SNR per $m-1$, m e $m+1$ bit. Come abbiamo visto nel Capitolo 2 ogni bit contribuisce con circa 6 dB al rapporto SNR. Consideriamo inoltre il rapporto segnale/maschera (**SMR** - Signal to mask ratio) che rappresenta la distanza tra il livello del rumore e la minima soglia di mascheramento e il rapporto (**NMR**, Noise to mask-ratio) che rappresenta la distanza tra il livello del rumore e la minima soglia di mascheramento. Il rapporto SNR è dato dalla somma di SMR e NMR. Sarà il rapporto SMR ad essere usato dagli algoritmi di compressione nell'allocazione dei bit necessari per la codifica del segnale.

6.11 Compressione MPEG

Il Motion Pictures Expert Group (MPEG) fu formato dall'ISO (International Organization for Standardization) per studiare standard per applicazioni multimediali che comprende il video con il suono. MPEG 1 fu introdotto nel novembre del 1992 e fu sviluppato per la codifica del segnale audiovisivo combinato a un bit-rate di circa 1,5 Mbps (1,2 Mbps per il video e 0,3 Mbps, cioè 300 Kbps, per l'audio stereofonico) con qualità inferiore alla TV. Lo standard prevede 32, 44.1 e 48 KHz come frequenze di campionamento e 1 o 2 canali. I modi di funzionamento sono mono, dual-mono, stereo, joint-stereo. Furono previsti tre livelli di compressione

- Layer I - più semplice (bit-rate superiore a 128 kbps, poca compressione)
- Layer II - più complesso (bit-rate di esattamente 128 kbps)

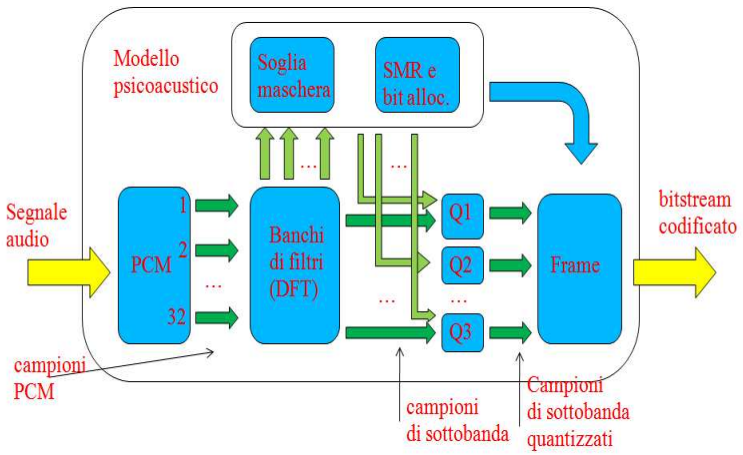


Figura 6.19: Decodificatore MPEG 1 Layer I.

- Layer III - il più complesso (bit-rate di 64 kbps, molta compressione).

Successivamente introdurremo e metteremo a confronto le caratteristiche di questi tre livelli di compressione.

6.11.1 Layer I

Lo schema generale del codificatore è descritto in Figura 6.19. La prima fase prevede il campionato mediante una codifica PCM. La frequenza di campionamento e il numero di bit sono determinati dall'applicazione specifica. La larghezza di banda che è disponibile per la trasmissione è divisa in un numero fissato di sottobande (32) di frequenza mediante un banco di filtri. Ogni sottobanda di frequenza ha la stessa larghezza. Il banco di filtri associa ad ogni pattern di 32 campioni PCM un pattern equivalente di 32 campioni di frequenze. In un codificatore base, vengono definiti dei segmenti che sono composti da 12 successivi pattern di 32 PCM e cioè 12 x 32 = 384 PCM.
Il modello psicoacustico contiene una FFT a 1024 punti e permette di avere la separazione del segnale in tonale e non tonale. Contiene, inoltre, il calcolo delle soglie di mascheramento globale e il calcolo del rapporto SMR per quantificare i dati da comprimere ed è necessario per stabilire il livello massimo di quantizzazione.
L'algoritmo di allocazione dei bit seleziona un passo di quantizzazione che soddisfa sia le richieste di bit-rate che il rapporto SMR (vedi Figura 6.18). L'algoritmo inizia con il calcolo del rapporto maschera/rumore MNR. Considerando tutti i valori MNR calcolati

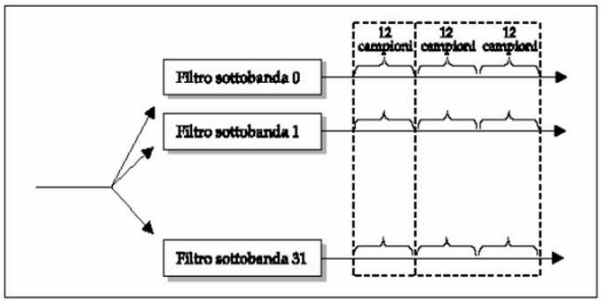


Figura 6.20: Segmenti PCM per sottobande.

per sottobanda, si cerca la banda con il più basso rapporto MNR e si allocano i bit per quella sottobanda. Quando i bit totali del blocco superano i limiti imposti dal bit-rate, l'unità di allocazione dei bit computa una nuova stima del rapporto SNR, e il rapporto MNR per le sottobande viene ricalcolato di conseguenza. Il processo si ripete finchè non si possono allocare più bit.

Nella Figura 6.21 viene rappresentato un esempio di allocazione di bit rispetto ai livelli di intensità sonora nelle sottobande e alla soglia globale di mascheramento. I tre diagrammi mostrano l'intensità sonora per sottobanda, la soglia di mascheramento per sottobanda e i bit allocati. Possiamo notare che a basse frequenze i suoni sono molti intensi e poco mascherati e quindi bisogna allocare alcuni bit per la quantizzazione. Alle alte frequenze, invece, si ha una debole intensità e un forte mascheramento per cui occorre istanziare un numero minore di bit. La fasi di ricostruzione del segnale vengono effettuate nel decodificatore di Figura 6.22. Sostanzialmente dopo che le ampiezze di ogni insieme delle 32 sottobande sono state determinate dal de-quantizzatore, sono passati al banco di filtri di sintetizzazione. Successivamente vengono prodotti i corrispondenti campioni PCM che sono successivamente decodificati per ottenere un segnale analogico.

6.11.2 Layer II

Lo standard MPEG 1 Layer II apporta lievi miglioramenti alle prestazioni del Layer I. Esso usa lo stesso 32 filtri e l'unità psicoacustica usa una FFT con risoluzione a 1024 punti. I blocchi diventano 1152 campioni e cioè tre blocchi consecutivi (384 x 3). Le sotto-bande vengono suddivise in tre regioni di frequenza (bassa, media, alta) e le classi di ri-quantizzazione in ciascuna regione di sotto-bande sono diverse.

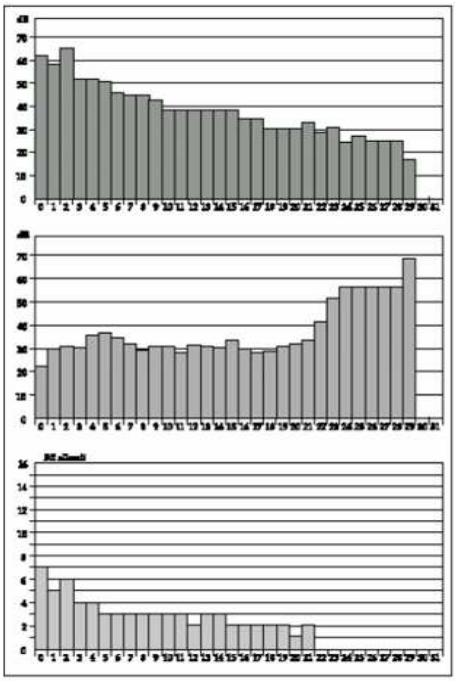


Figura 6.21: Allocazione di bit.

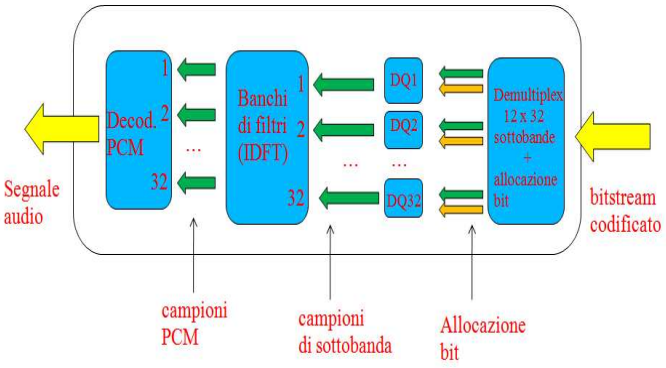


Figura 6.22: Decodificatore MPEG 1 Layer 1

6.11.3 Layer III

Vediamo ora più in dettaglio il codificatore e il decodificatore MPEG 1 Layer III che definisce la classe di codificatori audio più usati (mp3). Facendo riferimento alle Figure 6.23a e 6.23b analizzeremo i moduli principali di questo schema di codifica.

Il banco dei filtri polifase

Il primo passo è il filtraggio del segnale audio attraverso un banco di filtri. Una sequenza di 1152 campioni PCM sono filtrati da una struttura parallela di 32 filtri passa-banda e decimati di un fattore 32 (ogni sottobanda conterrà 36 campioni). Assumendo filtri passa banda perfetti (finestra quadrata), il teorema di Nyquist garantisce che i 1152 campioni PCM possono essere ricostruiti con una interpolazione delle sottobande, alla loro frequenza di campionamento originale, seguita da una somma delle sottobande. Chiaramente, il filtro non essendo perfetto viene prodotto aliasing.

Modulo MDCT

In questo processo le 32 sottobande sono elaborate dalla Modified Discrete Cosine Transform (MDCT). Con questa trasformazione otteniamo un miglioramento della risoluzione spettrale delle 36 linee di frequenza per sottobanda. Prima di trasformare viene effettuata una finestratura delle sottobande. Vengono definite due finestre (start e stop) per un migliore risultato nel caso di transienti. La decisione sul tipo di finestra applicata viene controllata dal modello psicoacustico.

Modulo FFT

Il modulo FFT viene usato per ottenere un'alta risoluzione spettrale. Vengono effettuate due FFT sui 1152 campioni PCM (rispettivamente a 1024 e 256 punti).

Modello psicoacustico

Il modello psicoacustico, invece, contiene un insieme di algoritmi basati sul modello psicoacustico che abbiamo visto in precedenza. I blocchi elaborati da questo modello sono

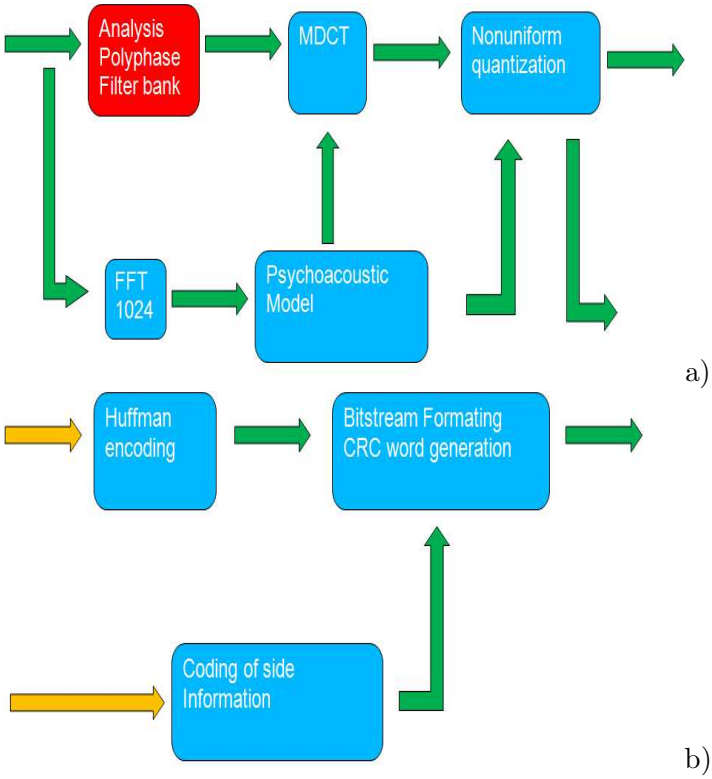


Figura 6.23: Codificatore MPEG 1 Layer III.

usati sia nell'MDCT che nel nonuniform quantization block. Un altro scopo è quello di decidere quale tipo di finestra usare.

Quantizzazione

Nel modello di quatizzazione viene effettuata una quantizzazione non-lineare delle frequenze. Nel primo passo le linee di frequenza vengono quantizzate mediante una potenza di 3 o 4 e successivamente vengono riscalate. Le caratteristiche del Layer III sono che la ri-quantizzazione è non uniforme. A causa della proliferazione di sotto-bande della MDCT si possono raggruppare delle sotto-bande per fattore di scala. Essa usa i codici di Huffman a lunghezza variabile per codificare i campioni quantizzati. Il bitstream è stato progettato in modo da soddisfare le richieste variabili di bit per le varie sotto-bande.

Codifica di Huffman

In questo blocco viene fatta una codifica delle linee precedentemente quantizzate. L'algoritmo usato è quello di Huffman che permette di ottenere una compressione senza perdita ed è basato su una strategia greedy.

Bitstream

La fase successiva è quella di ordinare tutti i parametri usati nel processo di codifica (coding of side information). L'ultima fase è quella della formazione dello stream e la generazione del campo CRC. In questo blocco le linee codificate con Huffman vengono assemblate per formare lo stream. Il bitstream è partizionato in frame di 1152 campioni PCM. Può inoltre essere incluso un campo Cyclic Redundancy Code (CRC) per validare i dati nel codificatore.

Ancillary data

Questo tipo di dato è tipicamente usato per inserire caratteristiche aggiuntive come il nome dell'artista o la categoria musicale.

6.11.4 MP3 in pratica

Il codificatore/decodificatore MPEG lavaora con quattro modalità possibili

- Segnali mono;
- Canali duali (indipendenti, non stereo)
- Segnali stereo
- Join stereo (combinazione dei segnali sinistro e destro)

E' possibile scegliere diversi tassi di campionamento come ad esempio 32, 44.1, 48 KHz per l'MPEG 1 e 16, 22.05, 24 KHz per l'MPEG 2.
La parte fondamentale della codifica risiede nella scelta del bit rate ed è lasciata all'implementatore. Lo standard comunque definisce la gamma di bit rate che va da 32 Kbps a 320 Kbps. La qualità del segnale ottenuto dalla decompressione dipende dal valore di bit rate cioè dalla forza di compressione. In Figura 6.24 viene presentata la compressione "mp3"

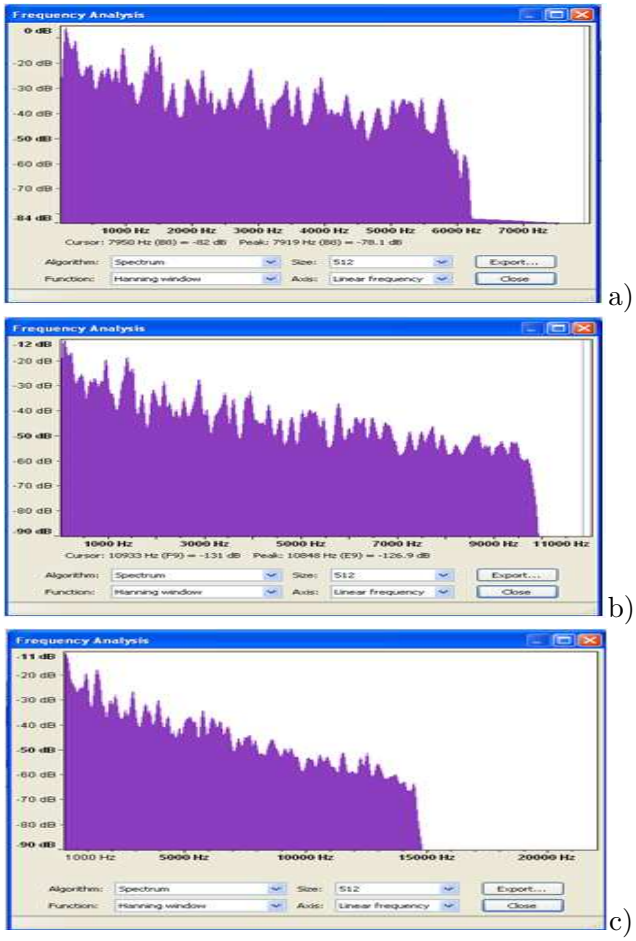


Figura 6.24: Spettro di segnali dopo la compressione mp3: a) 32 Kbps; b) 64 Kbps; c) 128 Kbps.

del segnale introdotto all'inizio del Capitolo. Le figure sono relative allo spettro ottenuto sul segnale usando 32, 64 e 128 Kbps, rispettivamente.

6.12 Codifica Dolby

La codifica Dolby è nata come codifica cinematografica (il primo film che la utilizzò la codifica fu "Batman, il ritorno" nel 1992). E' copyright della Dolby e si è poi diffusa anche nel settore Home Theater come specifica audio di qualità. La codifica è stata pensata per poter essere "contenuta" negli spazi non utilizzati dalle immagini in una pellicola cinematografica, ed inizialmente si utilizzava il marchio Dolby Digital solo per il

cinema, mentre per il mercato consumer si utilizzava il marchio Dolby AC3.
Nella versione a 5.1 canali, si hanno cinque canali effettivi (sinistro, centrale, destro, sinistro surround, destro surround) più il sesto canale detto LFE (Low Frequency Effects, al di sotto dei 120 Hz) su cui si trovano gli effetti a bassa frequenza che sovente si percepiscono come "vibrazioni". Per codificare i LFE basta 1/10 della banda rispetto agli altri cinque canali, da cui la scelta di indicarlo come canale ".1".
La codifica prevede un campionamento PCM da un minimo di 32 KHz ad un massimo di 48 KHz e almeno 20 bit di quantizzazione. I flussi ottenuti vanno da un minimo di 32 Kbps (mono a qualita' minima) fino a 640 Kbps (multicanale alla massima qualità). Ad esempio sui DVD a 5.1 canali si ha una larghezza di banda pari a 448 Kbps. La codifica, come nel caso MPEG 1 Layer 3, è fatta usando una DCT modificata. Inoltre, i numeri in virgola mobile sono spezzati in esponente e mantissa, e per ogni esponente si ricava da un modello psicoacustico quanti bit della mantissa è opportuno conservare (anche nessuno), riducendo il numero di bit necessari (ed introducendo un "rumore" non udibile). Questi esponenti, insieme con altri parametri, sono inviati per la decodifica insieme con il flusso.

6.13 AC3

La codifica AC3 risulta particolarmente efficiente in presenza di più canali. Le caratteristiche fondamentali sono infatti

- il global bit pool - se uno o più canali sono inattivi in un dato istante, vengono trasmessi più bit per i rimanenti canali;
- l'accoppiamento di alte frequenze, che si basa sull'osservazione che alle alte frequenze non viene percepito più il singolo periodo della forma d'onda, ma il suo inviluppo; si codifica quindi tale inviluppo.

Un flusso di bit AC3 è composto da frame indipendenti, ciascuno risultante dalla codifica di 1536 campioni PCM su tutti i canali ed ha lunghezza fissata in funzione della dimensione del campione e del dato codificato.

Capitolo 7

Effetti sonori

Un settore importante nell'ambito dei segnali audio musicali è quello degli effetti sonori. Gli efetti sonori sono tipicamente avvertiti in normali condizioni d'ascolto e sono ad esempio il riverbero, l'eco, la percezione spaziale oppure sono applicati direttamente a strumenti musicali come nel caso della distorsione, il flanging, il chorus, ecc.. In questo Capitolo focalizziamo la nostra attenzione sulla descrizione di alcuni tipi di effetti audio che si ottengono effettuando semplici operazioni sul segnale nel dominio del tempo.

7.1 Filtri

Prima di introdurre gli effetti audio approfondiamo alcuni concetti relativi ai filtri. Molto spesso per la manipolazione dei segnali vengono usate le *linee di ritardo* (Delay Lines, DL) e particolari filtri come quelli *comb* (a pettine) o quelli *all-pass* (passa-tutto) (o filtri comb universali). Vediamo come essi possono essere implementati.

7.1.1 Filtri comb

I filtri comb sono caratterizzati da una risposta che ricorda una forma a pettine e sono caratterizzati da un numero di risonanze e antirisonanze. Possono essere ricorsivi e quindi di tipo IIR o non ricorsivi e quindi di tipo FIR.

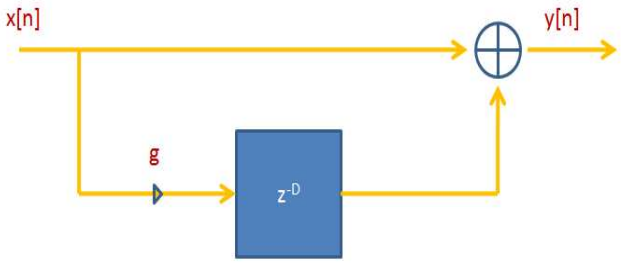


Figura 7.1: Filtro comb non ricorsivo FIR.

Filtri comb FIR

In Figura 7.1 viene rappresentato lo schema del filtro comb non ricorsivo (FIR). La relazione ingresso-uscita è espressa attraverso la seguente equazione alle differenze

$$y[n] = x[n] + gx[n - D] \tag{7.1}$$

mentre la funzione di trasferimento risulta

$$H(z) = 1 + gz^{-D}. \tag{7.2}$$

Filtri comb IIR

La struttura del filtro com IIR è riportata in Figura 7.2. La linea di ritardo può essere inserita nel ramo di feedforward o nel ramo di feedback. Le relazioni ingresso-uscita, sono espresse dalle seguenti equazioni alle differenze

$$y[n] = x[n - D] - gy[n - D] \tag{7.3}$$

e

$$y[n] = x[n] - gy[n - D]. \tag{7.4}$$

Mentre le rispettive funzioni di trasferimento sono

$$H(z) = \frac{z^{-D}}{1 + gz^{-D}} \tag{7.5}$$

e

$$H(z) = \frac{1}{1 + gz^{-D}}. \tag{7.6}$$

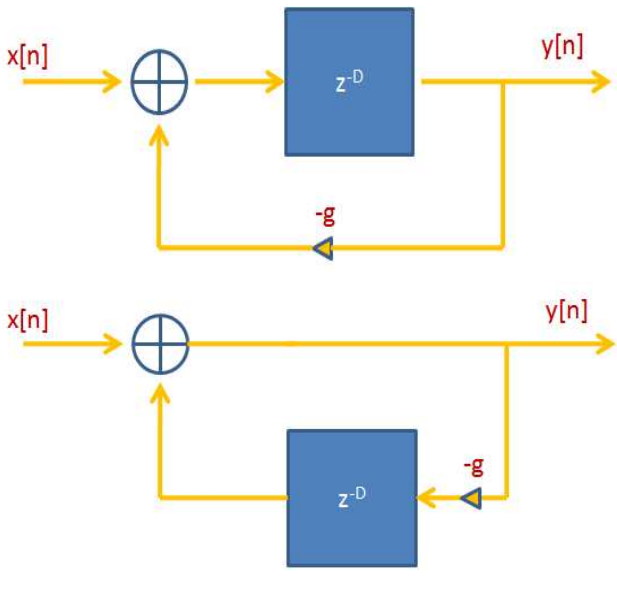


Figura 7.2: Filtro comb ricorsivo IIR: a) DL nel ramo feedforward; b) DL nel ramo feedback.

7.1.2 Filtri all-pass

Un filtro all-pass è caratterizzato da una funzione di trasferimento in cui gli zeri sono reciproci dei poli. La funzione di trasferimento risulta

$$H(z) = \frac{a_N + a_{N-1}z^{-1} + \ldots + a_1 z^{-(N-1)} + z^{-N}}{1 + a_1 z^{-1} + \ldots + a_N z^{-N}}. \tag{7.7}$$

dove al numeratore compaiono gli stessi coefficienti del denominatore ma in versione specchiata. La relazione di ingresso-uscita è

$$y[n] = a_N x[n] + \ldots + x[n-N] - a_1 y[n-1] - \ldots - a_n y[n-N]. \tag{7.8}$$

Nel caso in cui la funzione di trasferimento assume la particolare espressione

$$H(z) = \frac{g + z^{-D}}{1 + gz^{-D}} \tag{7.9}$$

viene anche detto filtro *comb universale*. La relazione ingresso-uscita risulta

$$y[n] = gx[n] + x[n-D] - gy[n-D]. \tag{7.10}$$

7.2 Linee di ritardo

La linea di ritardo o *tapped delay line* (DL) è la struttura fondamentale per l'implementazione dei filtri numerici FIR e IIR. Essa è alla base della realizzazione di numerosi effetti audio quali per esempio il vibrato, il flanger, il chorus, lo slapback, l'echo, ecc. e la simulazione dell'acustica dell'ambiente. La linea di ritardo viene usata come un ritardo puro e cioè

$$y[n] = x[n - D] \tag{7.11}$$

Nelle figure precedenti la linea di ritardo era identificata dal blocco z^{-D}. La linea di ritardo può essere realizzata con un array in cui vengono fatti scorrere i campioni. L'algoritmo per la realizzazione di un linea di ritardo risulta

```
double DL(double *w, int D, double x)
{
 int i;
  for(i=D; i>=1; i--) w[i] = w[i-1];
  w[0] = x;

  return w[D];
}
```

Praticamente il segnale entra dalla prima locazione di memoria e a ogni istante di clock il campione scorre liberando la prima posizione in cui viene fatto contestualmente entrare il nuovo campione del segnale disponibile.
Una diversa implementazione è attraverso un buffer circolare. In questo caso invece di scorrere i campioni viene incrementato un indice per l'inserimento del campione in ingresso. Ad esempio otteniamo

```
double DL(double *w, int D, int p, double x) {
  w[p] = x;
  p = (p+1) % D;
  return w[p];
}
```

oppure

```
double DL(double *w, int D, int p, double x) {
```

```
    double y = w[p];
    w[p++] = x;
     if (p>= D) p = - D;
    return y;
  }
```

7.2.1 Linee con ritardo frazionario

Il ritardo minimo di una linea di ritardo digitale è definita dalla frequenza di campionamento f_c del segnale e risulta essere uguale al periodo di campionamento T_c (ritardo unitario). In molte applicazioni è necessario avere un ritardo che può essere un multiplo del ritardo unitario. In questo caso il ritardo risulta frazionario (fractional delay, FD). Vediamo ora alcune semplici soluzioni a questo problema.

Interpolazione lineare

Il modo più semplice ed intuitivo per la determinazione di un ritardo frazionario è quello di considerare un'interpolazione lineare tra due campioni successivi del segnale. Questa si può ottenere ad esempio nei seguenti due modi

$$y[n] = x[n-1] + \alpha(x[n] - x[n-1]) \tag{7.12}$$

oppure

$$y[n] = (1-\alpha)x[n-1] + \alpha x[n]. \tag{7.13}$$

Filtro all-pass

Un'altra tecnica molto usata, che presenta una minore complessità computazionale, usa i filtri all-pass. In questo caso la funzione di trasferimento risulta

$$H(z) = \frac{a + z^{-1}}{1 + az^{-1}} \tag{7.14}$$

dove $|a| < 1$ per la stabilità. La relazione ingress-uscita risulta

$$y[n] = ax[n] + x[n-1] - ay[n-1]. \tag{7.15}$$

7.2.2 Linee di ritardo tempo varianti

Molti effetti audio sono basati sull'utilizzo di DL con lunghezza variabile nel tempo

$$y[n] = x[n - D[n]] \tag{7.16}$$

Può essere conveniente esprimere il ritardo variabile nel tempo come

$$D[n] = D_0 + D_1 f_D[n] = D_0(1 + m_D f_D[n]) \tag{7.17}$$

dove D_0 rappresenta la lunghezza della DL, la funzione $f_D[n]$ rappresenta la legge di variazione e la costante $m_D \in [0, 1]$ rappresenta l'indice di modulazione. Il tipo di effetto che è possibile ottenere dipende dalla legge di variazione di $f_D[n]$ e dalla sua profondità di modulazione $D_1 = m_D D_0$ (vedi Figura 7.3). In generale il valore $D[n]$ non è intero e deve essere interpolato con tecniche di interpoalzione introdotte precedentemente.
Una possibile implementazione in C++ di una linea di ritardo con lunghezza variabile e ritardo frazionario provvista di interpolazione lineare è la seguente

```
static double A[N];
static double *rptr = A;
static double *wptr = A;
double setdelay(int M){
   rptr = wptr - M;
   while (rptr < A) { rptr += N; }
  double delayline(double x){
  double y;
  A[wptr++] = x;
  long rpi = (long) floor(rptr);
  double a = rptr - (double) rpi;
  y = a * A[rpi] + (1 -a) * A[rpi + 1];
  rptr +=1;
  if((wptr - A) >= N) { wptr -= N; }
  if((rptr - A) >= N) { rptr -= N; }
  return y;
}
```

La funzione $D[n]$ generalmente è continua e quindi descrivibile da una parte intera e una frazionaria. Se usiamo un oscillatore a bassa frequenza (Low Frequency Oscillator, LFO) possiamo avere un segnale modulante. Lo schema è descritto in Figura 7.4.

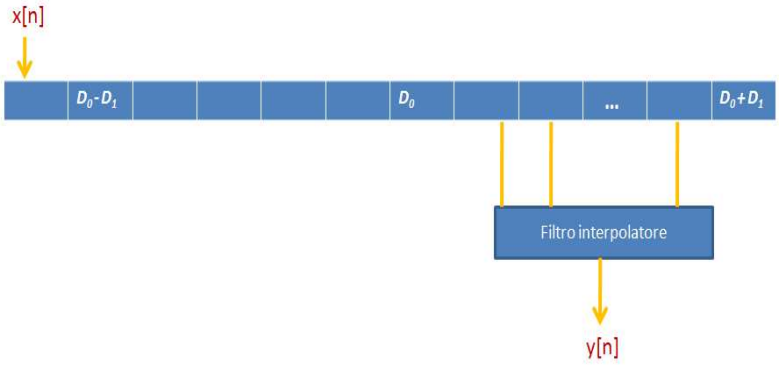


Figura 7.3: Linea con ritardo con lunghezza variabile e filtro interpolatore.

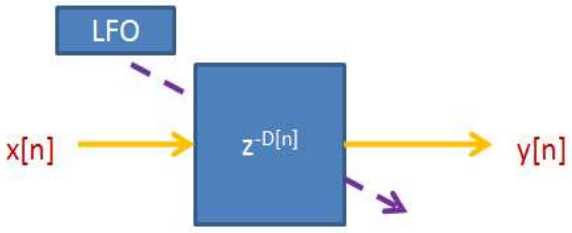


Figura 7.4: Modulatore del tipo LFO.

Cambio di pitch

Un effetto che si può costruire con le linee di ritardo a tempo varianti è quello relativo al cambio di pitch. Consideriamo

$$D[n] = (1 - p)n \tag{7.18}$$

dove p (pitch change ratio) è il rapporto di variazione del pitch tra il segnale processato dalla linea di ritardo modulata e il segnale originale. Per $p = 2$, ad esempio, si ha un raddoppio delle altezze l'uscita dalla linea di ritardo modulata vale

$$x[n - D(n)] = x[pn]. \tag{7.19}$$

Vibrato

Un altro effetto che si può ottenere dalle linee di ritardo a tempo varianti è quello del vibrato. Consideriamo in questo caso

$$D[n] = \frac{M}{2}\sin(2\pi f_0 n) + \frac{M}{2} \tag{7.20}$$

otteniamo

$$x[n - D[n]] = x[n - \frac{M}{2} - \frac{M}{2}\sin(2\pi f_0 n)]. \tag{7.21}$$

Anche in questo caso siccome il valore $D[n]$ non è intero si usano le tecniche di interpoalzione introdotte precedentemente.

7.2.3 Distorsione

Applicando i due diversi interpolatori su un modello per l'alterazione costante del pitch si può ottenere l'effetto di distorsione.

7.3 Effetti

Le linee di ritardo tempo varianti frazionarie sono l'elemento principale per la realizzazione di effetti audio e algoritmi di sintesi sonora. Gli effetti possono essere: il vibrato, il flanging, il chorus, il phasing, ecc.
Lo schema generale di un effetto tradizionale digitale è visualizzato in Figura 7.5.
In questo schema riconosciamo

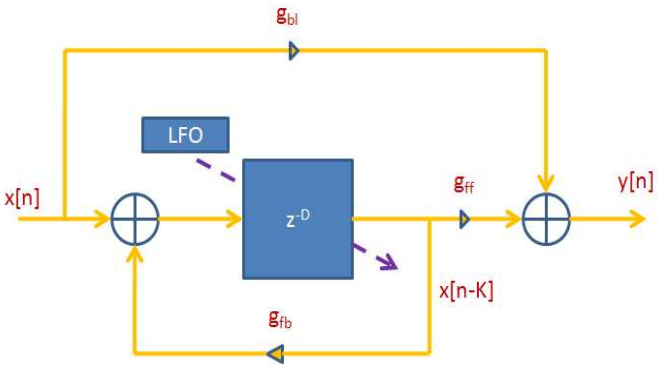


Figura 7.5: Schema con linea di ritardo a lunghezza variabile frazionaria.

- un guadagno di feedback, controllato dal coefficiente g_{fb}. E' prelevato a un ritardo fisso di lunghezza $K = D_0$. K identifica il delay nominale.
- l'uscita modulata, pesata dal coefficiente di feedforward g_{ff};
- un segnale non modulato, che viene miscelato a quello modulato ed è controllato mediante il coefficiente di blend g_{bl}.

La funzione di trasferimento dello schema risulta

$$H(z) = \frac{g_{bl} + g_{ff} z^{-D[n]}}{1 + g_{fb} z^{-K}} \tag{7.22}$$

Per alti valori di g_{fb} si ottiene un suono metallico e intenso. Bisogna anche controllare la stabilità del sistema in modo da prevenire errori di *overflow* e *clipping*. La scelta dei parametri messi a disposizione dallo schema permette di ottenere un discreto numero di effetti diversi.

7.3.1 Vibrato

Eliminando il feedback e il blending, avendo quindi sull'uscita solo il segnale modulato, si ha un effetto vibrato. La linea di ritardo sarà dimensionata per gestire un ritardo approssimativamente inferiore a 5 ms. Un ritardo minimo, anche inferiore a 1 ms, dà i migliori risultati. La lunghezza nominale D_1 della linea di ritardo risulta pari a

$$D_1 = \frac{f_c}{2f_0} m \tag{7.23}$$

Ad esempio con una frequenza di campionamento $f_C = 44.1KHz$, $f_0 = 20$ Hz la profondità di modulazione risulta $D_1 = 127.45$. In questo caso la lunghezza nominale della linea D_0 potrebbe essere $D_0 = 128$ ed in termini di occupazione di memoria di 256 campioni.

7.3.2 Flanging

Lo scopo del flanger è quello di sovrapporre al segnale d'origine un segnale ritardato dinamicamente. Esso è ottenuto disattivando il feedback e il blend. Una linea di ritardo generalmente va da 1 ms a 10 ms. L'equazione dell'uscita del flanger risulta

$$y[n] = x[n] + g_{ff}x[n - D[n]]; \tag{7.24}$$

Nel caso di modulazione sinusoidale abbiamo

$$D[n] = D_0 + D_1 \sin(2\pi f_{FL} n). \tag{7.25}$$

La frequenza f_{FL} definisce la velocità della variazione spettrale e ha frequenza di 1 Hz o minore. D_0 rappresenta la lunghezza media della linea di ritardo. Il parametro D_1 definisce la massima escursione della linea.

Flanging invertito

Nel caso in cui $g_{ff} < 0$ la risposta in frequenza presenta delle risonanze poichè è invertita. In questo caso l'effetto è del tipo "pass-alto".

Flanging stereo

Nel caso in cui il segnale audio considerato sia già stereo si applica l'effetto su ogni singolo canale. Nel caso in cui la sorgente risulta del tipo monofonico e si vuole ottenere un effetto stereofonico si possono modulare due flanger indipendenti attraverso lo stesso segnale LFO ma con segnali tra loro sfasati di 90 gradi.

7.3.3 Chorus standard

L'effetto chorus consiste nella combinazione di almeno due voci indipendenti. Ognuna delle voci ha delle variazioni casuali (ritardo, pitch, ampiezza, ecc.) in modo da rendere il suono più ricco e suggestivo. La struttura base per la realizzazione dell'effetto si basa sulla

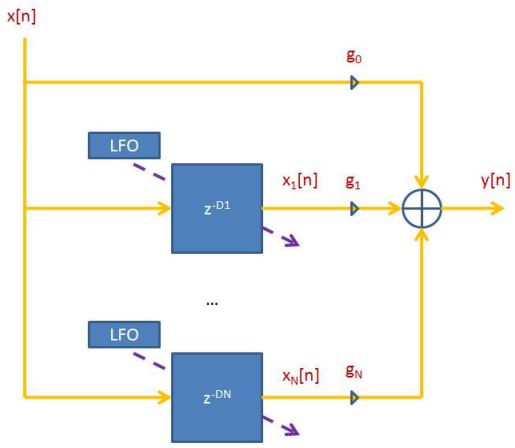


Figura 7.6: Schema dell'effetto chorus.

ripetizione dello schema flanger per ogni voce. Lo schema generale è descritto in Figura 7.6.

7.3.4 Doubling

Un effetto usato soprattutto dai cantanti è quello di raddoppiare la traccia del cantato, rieseguendo la parte che va a sovrapporsi quella già esistente. Il doubling deriva dallo schema generale di Figura 7.5 in cui non è attivo il feedback ($g_{fb} = 0$) e con $D_0 = 20$ ms. Qui il margine di ritardo può variare discretamente (20 ms tipicamente vanno bene) ma una modulazione abbastanza casuale, come quella causata dal doppio cantato, è auspicabile.

7.3.5 Echo

L'eco si distingue per la lunghezza della linea di ritardo, che dovrebbe assicurare un ritardo di almeno 80 ms, tempo al di sotto del quale non è assicurata la percezione distinta di due suoni identici. I coefficienti vengono perlopiù tarati in base alla timbrica richiesta per il tipo di eco.

7.3.6 Parametri degli effetti

Riassumiamo nella Tabella 7.1 i tipici parametri per gli effetti appena introdotti. Nella tabella 0.707 si riferisce all'approssimazione dell'operazione $\sqrt{1/2}$.

	g_{bl}	g_{ff}	g_{fb}	Onset	Depth	Modulazione
Vibrato	0.0	1.0	0.0	0 ms	0-5ms	0.1-5 Hz sinusoidale
Flanger	0.707	0.707	-0.707	0ms	1-10 ms	0.1- 1 Hz sinusoidale
Chorus	1.0	0.707	0.0	1-30 ms	5-30 ms	Lowpass noise
White chorus	0.707	1.0	0.707	1-30 ms	5-30 ms	Lowpass noise
Doubling	0.707	0.707	0.0	10-100 ms	1-100 ms	Lowpass noise
Eco	1.0	≤ 1.0	< 0	50-∞	80-∞	-

Tabella 7.1: Parametri relativi agli effetti sonori.

7.4 Riverbero

Il riverbero è quell'effetto naturale scaturito da un particolare ambiente. Esso è dato dall'insieme delle risonanze e degli echi che accompagnano il suono puro ed influiscono in modo fondamentale sulla nostra percezione del timbro e dell'ambiente che ci circonda (vedi Figura 7.7). In molte circostanze l'effetto risulta desiderabile in altre può risultare sgradevole.

Ad esempio una registrazione condotta all'interno di un ambiente riverberante risulta pessima nella maggioranza dei casi. Infatti può capitare che se i musicisti sono più di uno, i fenomeni di cross-talk tra uno strumento e l'altro si mescolano alle tracce registrate. Questo obbliga a rendere una sala di registrazione la più anecoica possibile e a minimizzare l'influenza sulla singola traccia di qualunque segnale esterno.

La fase più difficile nel processo di messa a punto di un riverbero è costituita dalla scelta di un modello adeguato. Accenneremo a due approcci, quello fisico e quello percettivo.

7.4.1 Approccio fisico

L'approccio fisico si basa sulla messa a punto di modelli dotati di parametri fisici accessibili, come dimensioni e forma dell'ambiente, riflessività delle pareti e attenuazione ambientale, posizioni della sorgente e del punto di ascolto ecc.. La soluzione più semplice consiste nel memorizzare a bordo del riverbero tutta l'informazione relativa a un insieme di N risposte impulsive ambientali campionate $h_1, \ldots, h_N$, e di calcolare il segnale riverberato y come convoluzione discreta tra il segnale musicale x e una risposta ambientale h_i scelta

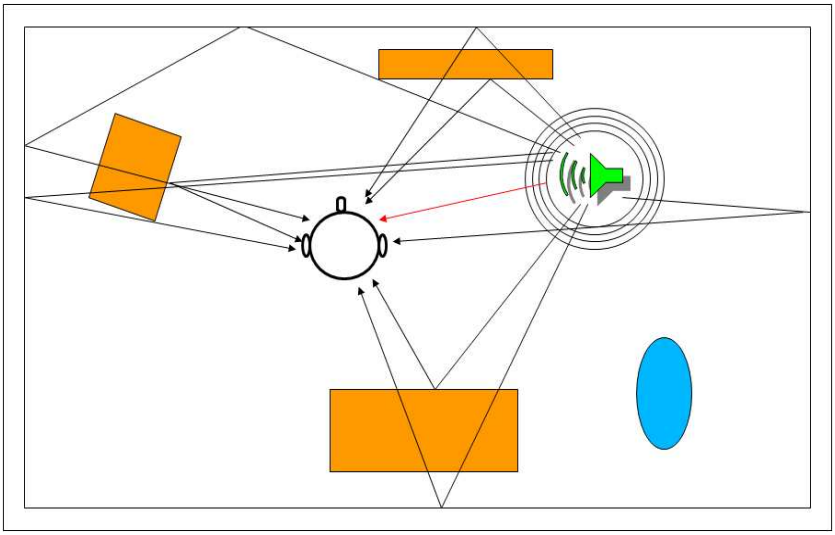


Figura 7.7: Effetto di riverbero.

all'interno di un database

$$y(n) = h_i \otimes x(n) \tag{7.26}$$

Negli anni sono stati proposti diversi modelli di propagazione del segnale all'interno di un ambiente, con l'intento di semplificarne la descrizione e in conseguenza avere un guadagno in termini di efficienza. Ad esempio ricordiamo il *metodo delle immagini*, che deduce la direzione e il tempo di arrivo delle prime riflessioni in un punto dell'ambiente, nota la sua topologia. Inoltre abbiamo la tecnica del *ray tracing* che permette di modellare le riflessioni basandosi sul concetto di campo diffuso e i metodi statistici che si basano sulla modellazione degli eco che formano il tappeto acustico.

7.4.2 Approccio percettivo

Nell'approccio percettivo il controllo dei parametri percettivi possono essere calibrati al gusto dell'ascoltatore. E' possibile inoltre ottenere un sistema più semplificato essendo basato sullo studio dell'informazione presente nella risposta impulsiva ambientale. Le difficoltà nascono quando si cercano di riconoscere dei parametri indipendenti (o fattori) capaci di caratterizzare la sensazione del riverbero. L'obiettivo dei modelli percettivi è la realizzazione di un algoritmo di riverbero piacevole all'ascolto e molto naturale. La strada

per raggiungere questo obiettivo è quella di riprodurre la timbrica del suono lavorando entro la costante di integrazione dell'orecchio, e di generare un valido tappeto acustico.

Capitolo 8

Sintesi sonora

In questi ultimi anni un crescente interesse, nel campo dell'elaborazione dei segnali, è stato rivolto alla *computer music*. Diversi approcci sono stati proposti per la composizione automatica e assistita, per l'automatizzazione di tecniche compositive e assistenza alla composizione (sequencers). Inoltre la ricerca intorno all'Intelligenza Artificiale ha permesso lo sviluppo di applicazioni per la simulazione dello stile compositivo o di grammatiche come modelli di rappresentazione. All'interno della computer music un ruolo fondamentale è ricoperto dalle tecniche di sintesi dei suoni. Prevalentemente, dato uno strumento musicale, la sintesi cerca di creare suoni sintetici quanto più vicini a quelli reali. La sintesi è usata anche per simulare la voce umana o per realizzare algoritmi di compressione. In questo Capitolo verranno descritte le tecniche principali di sintesi sonora.

8.1 Classi di algoritmi

In uno strumento musicale tradizionale il suono è prodotto dalla vibrazione di parti meccaniche. Negli strumenti musicali sintetici, invece, la vibrazione è descritta da segnali realistici in funzione nel tempo. In uno strumento musicale sintetico lo scopo diventa quello di implementare una rappresentazione semplificata ed astratta del modello reale. Nella descrizione del modello si ricorre a relazioni matematiche per prevedere il comportamento dello strumento che includono i parametri variabili nel tempo, e lo stato iniziale. Nella musica occidentale i parametri che il musicista gestisce sono l'altezza del suono, l'intensità, la durata metrica, il timbro e la localizzazione spaziale.
Lo spazio e soprattutto il timbro sono i parametri che caratterizzano i suoni sintetici. E'

possibile fare una classificazione degli algoritmi di sintesi in base sull'analisi della loro struttura. Si possono individuare le seguenti classi di algoritmi:

- *generazione diretta* - di questa classe fanno parte campionamento, sintesi additiva, granulare;
- *feed-forward* - sottrattiva, modulazioni, distorsione non lineare;
- *feed-back* - sintesi per modelli fisici.

8.2 Sintesi

L'algoritmo più semplice di sintesi è quello che si basa sulla generazione di un segnale periodico come ad esempio il tono puro che abbiamo incontrato nel Capitolo 1. Ad esempio è possibile definiere un segnale periodico $y = f(x)$ o un tono puro come $y(t) = A\sin(2\pi ft)$. Se si vuole generare un segnale mono della durata di 1 secondo in qualità CD abbiamo bisogno di un array di 44100 campioni. Come possiamo subito verificare questo algoritmo è poco efficiente dal punto di vista computazionale siccome dobbiamo calcolare ogni volta la funzione in base alla frequenza di campionamento scelta. La lettura in memoria è più veloce del calcolo del valore di una funzione e quindi l'idea è quella di costruire un oscillatore digitale. Come vedremo successivamente.

8.3 Table Look-UP Synthesis

In questo caso possiamo costruire un oscillatore digitale osservando che un suono sinusoidale si ripete uguale ad ogni periodo. Possiamo costruire una tabella (*wavetable*) di n punti (vedi Figura 8.2) relativa ad un suono sinusoidale composto da un solo periodo (vedi Figura 8.1). Le operazioni principali che possiamo effettuare sono

- lettura dei valori della tabella
- arrivato all'ultimo indirizzo riparti dal primo.

L'ultima operazione è chiamata *wrapping around*. Sostanzialmente la tabella costituisce un modello della forma d'onda e sta all'utente decidere ampiezza e frequenza del segnale sintetizzato dall'oscillatore. In Figura 8.3 è presentato lo schema del sintetizzatore.

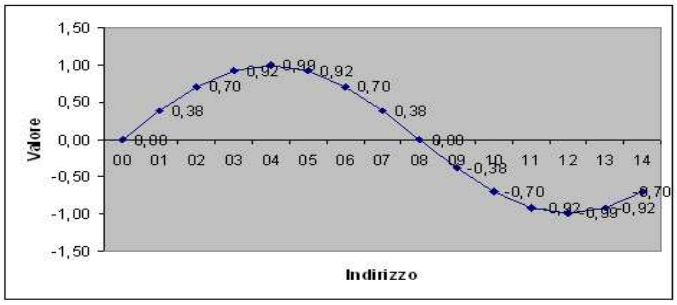


Figura 8.1: Periodo di un segnale sinusoidale.

Indirzzo	Valore
00	0,00
01	0,38
02	0,70
03	0,92
04	0,99
05	0,92
06	0,70
07	0,38
08	0,00
09	-0,38
10	-0,70
11	-0,92
12	-0,99
13	-0,92
14	-0,70
15	-0,38

Figura 8.2: Wavetable.

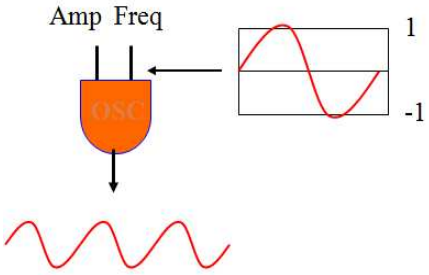


Figura 8.3: Schema del sintetizzatore.

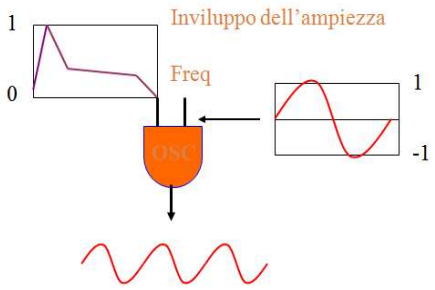


Figura 8.4: Schema per la modulazione con inviluppo.

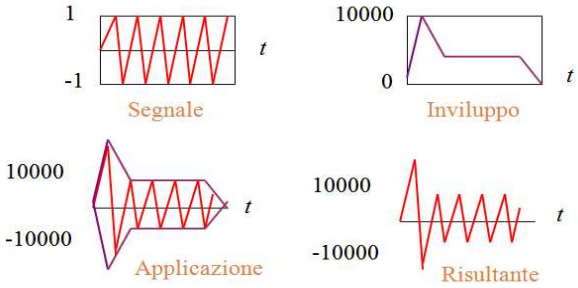


Figura 8.5: Modulazione con inviluppo.

8.4 Segnali di controllo

Dobbiamo comunque osservare che un tono puro manca totalmente delle caratteristiche di dinamicità dei suoni naturali. Un metodo semplice per rendere più complesso il segnale consiste nel controllarlo con un altro segnale che viene chiamato modulatore.

8.4.1 Inviluppo

In questo caso possiamo considerare che un segnale reale ha un forma temporale caratteristica. La forma d'onda è caratterizzata da un *attacco-decadimento-sostegno* e rilascio (attack/decay/sustain/release). Possiamo pensare quindi di generare un segnale di controllo in forma di linea spezzata che descrive un inviluppo di ampiezza. Lo schema del sintetizzatore con il relativo inviluppo è presentato in Figura 8.4. In Figura 8.5 viene presentato un esempio di segnale ottenuto dopo la modulazione tramite inviluppo. Possiamo notare come il segnale risultante può risultare più realistico.

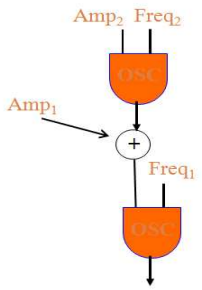


Figura 8.6: Schema per il tremolo.

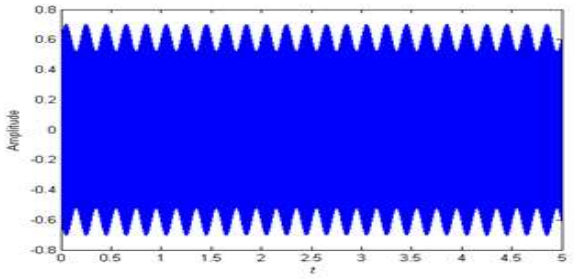


Figura 8.7: Segnale con tremolo.

8.4.2 Tremolo

Per generare l'effetto del tremolo possiamo pensare di controllare l'ampiezza dell'oscillatore attraverso un altro oscillatore. E' sufficiente sommare all'ampiezza del primo oscillatore il segnale prodotto da un oscillatore di controllo (vedi Figura 8.6). Il contributo dell'oscillatore di controllo è una variazione periodica della dinamica percepita e cioè il tremolo (caratteristica dei fiati). Un esempio di segnale con tremolo è visualizzato in Figura 8.7.

8.4.3 Vibrato

In questo caso potremmo pensare di controllare, invece che l'ampiezza, la frequenza dell'oscillatore attraverso un altro oscillatore (vedi Figura 8.8). Il risultato musicale di una simile variazione periodica dell'altezza di una nota viene definito vibrato. In Figura 8.9 viene visualizzato un esempio di segnale con vibrato.

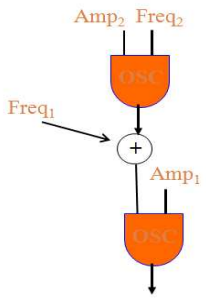


Figura 8.8: Schema per il vibrato.

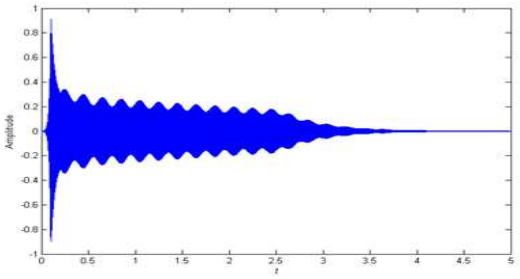


Figura 8.9: Segnale con vibrato.

8.5 Metodi di sintesi

Nella letteratura scientifica negli anni sono stati introdotti diversi metodi di sintesi per l'audio digitale. Successivamente introdurremo alcuni di questi metodi come il metodo del campionamento, della generazione diretta, della trasformazione e dei modelli fisici.

8.5.1 Campionamento

Questa tecnica semplicemente si basa sull'estrazione di campioni da una sequenza digitale. Inizialmente si ottiene un segnale di breve durata mediante la registrazione diretta o attraverso l'editing di un file audio. La lettura avviene da una tabella. Rispetto all'oscillatore semplice comunque in questo caso abbiamo a disposizione una forma d'onda di un segnale complesso dotata di un proprio inviluppo. I limiti computazionali dipendono dall'hardware a disposizione.

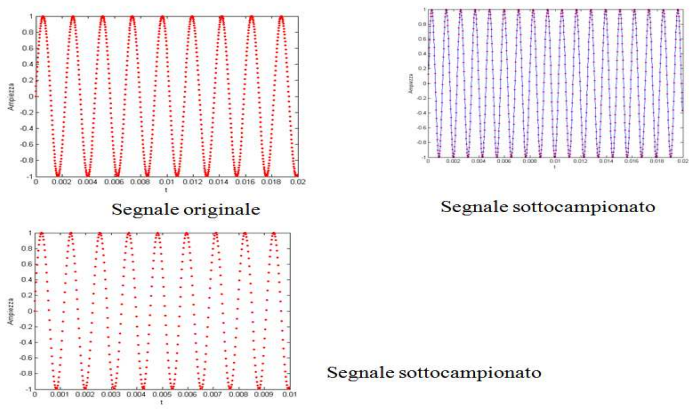


Figura 8.10: Effetto di downsampling e upsampling.

Resampling

Nell'utilizzo della tecnica di campionamento si presenta l'inconveniente che dobbiamo memorizzare le sequenze di una nota a tutte le tonalità. Con un semplice accorgimento possiamo "intonare" il suono campionato. In altre parole è possibile effettuare la variazione della frequenza leggendo tutti i punti dalla wavetable e cambiando la frequenza di campionamento. Infatti, se dalla sequenza originale prendiamo un punto ogni due la frequenza che otteniamo è doppia (*downsampling*) e quindi abbiamo la trasposizione all'ottava superiore e cioè il nuovo periodo T_2 è la metà del periodo originale T_1, $T_2 = T_1/2$ e e il rapporto tra le frequenze di campionamento diventa $R = f_2/f_1 = 2$. Se invece prendiamo due punti ogni punto la frequenza viene dimezzata (*upsampling*, interpolazione). In Figura 8.10 viene presentata una sinusoide e le sequenze ottenute con downsampling e upsampling.
Per trasportare il campione originale non solo all'ottava ma a tutte le frequenze bisogna considerare anche rapporti numerici non interi. Infatti il rapporto tra due frequenze a distanza di semitono (*do* e *do#*) è $\sqrt[12]{2} \approx 1.06$. In questo caso è necessario effettuare un'interpolazione o una decimazione.

Looping

Supponiamo ora di voler simulare la durata di una nota ad esempio in base a quanto un musicista preme un tasto. La tecnica adottata denominata *loop* semplicemente permette

di reiterare il segmento.

8.6 Generazione diretta

Nella tecnica di sintesi basata sulla *generazione diretta* un segnale viene semplicemente generato attraverso il calcolo del valore delle formule matematiche che lo descrivono.

8.6.1 Sintesi additiva

La *sintesi additiva* è una delle prime introdotte nel settore e si basa direttamente sull'idea del Teorema di Fourier. Infatti esso afferma che un segnale periodico qualsiasi è dato dalla sovrapposizione di onde sinusoidali semplici, ciascuna con la sua ampiezza e fase, e le cui frequenze sono armoniche della frequenza fondamentale del segnale. L'idea della sintesi additiva è quella che un segnale complesso può essere costruito a partire dalla somma di segnali sinusoidali ognuno dei quali dotato di un inviluppo d'ampiezza.

8.7 Altri metodi

Esistono altri tipi di sintesi che successivamente accenneremo brevemente. La *sintesi granulare* si basa sul concetto che un suono può essere pensato non solo in termini ondulatori ma anche in termini corpuscolari. Questi vengono chiamati grani sonori. Un modello compositivo per la sintesi granulare può utilizzare dei grafi per produrre sequenze di grani.
La *sintesi sottrattiva*, invece, fa parte dei metodi di trasformazione. I metodi di trasformazione permettono di ottenere un nuovo segnale attraverso la modificazione di un altro segnale. In questo tipo di sintesi il segnale di input è un segnale complesso che viene filtrato attenuando le frequenze indesiderate ed enfatizzando solo alcune regioni. La sintesi sottrattiva è il metodo alla base della tecnica standard di simulazione artificiale della voce e codifica a predizione lineare (Linear Predictive Coding - LPC).
Nelle tecniche di sintesi basate su *analisi e risintesi* sono proprio i dati di controllo che vengono derivati dall'analisi di un segnale preesistente. Il processo viene usualmente scomposto in tre fasi:

- creazione di un file contenente i dati ricavati dal'analisi

- modifica del file d'analisi
- risintesi a partire dal file d'analisi modificato.

Molte sono le possibilità di realizzazione. Tra le tecniche più rilevanti basate rispettivamente sulla sintesi additiva e sottrattiva sono il Phase Vocoder (basato sulla Short Time Fourier Transform) e Linear Predictive Coding.
Infine alla base della sintesi per *modelli fisici* c'è l'idea che il segnale di uno strumento musicale può essere espresso da equazioni che descrivono il comportamento fisico (acustico e meccanico) delle parti che interagiscono nella produzione del suono.

Capitolo 9

Introduzione al C++

La programmazione orientata agli oggetti è una tecnica di programmazione introdotta agli inizi degli anni 80 ed è una metodologia per costruire prodotti software di grosse dimensioni affidabili e facilmente modificabili. Tra i principali linguaggi che usano questo schema di programmazione ci sono il linguaggio C++ e il linguaggio Java. Il linguaggio di programmazione orientato agli oggetti, oggi, è utilizzato in tantissime tecniche di progettazione: database, interfacce grafiche, protocolli di rete, applicazioni Web, etc. In questo Capitolo si intende fare una breve introduzione del linguaggio C++ per agevolare lo studente nella comprensione degli algoritmi introdotti nei Capitoli precedenti.

9.1 Nascita di C++

Il linguaggio C++ fu introdotto da B. Stroustrup nel 1984. Successivamente, all' AT&T tra il 1998 e il 1999 fu definito lo standard ANSI del linguaggio e della libreria. Inizialmente il linguaggio era definito come linguaggio C più la definizione delle classi. Successivamente è stato esteso per supportare la programmazione basata sugli oggetti, la programmazione orientata agli oggetti e la programmazione generica. Il C++ consente di controllare la macchina a basso livello e supporta l'astrazione dei dati e la Programmazione Orientata agli Oggetti (Object Oriented Programming, OOP). Supportare uno stile di programmazione significa fornire strumenti che rendono quello stile semplice ed efficiente e cioè operatori, controlli in compilazione ed esecuzione.

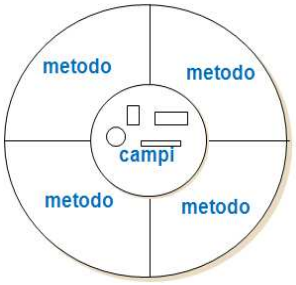


Figura 9.1: Tipo di dati astratti: **classe**.

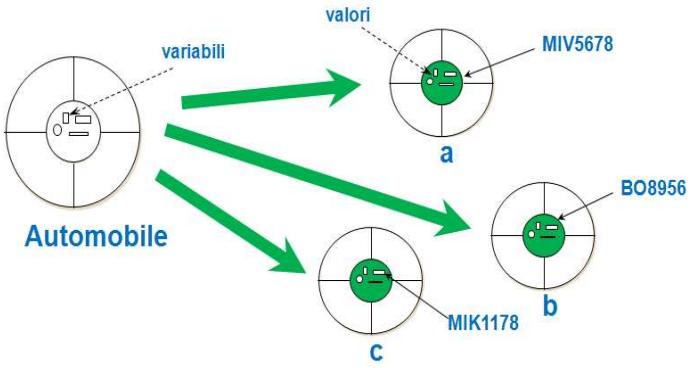


Figura 9.2: Istanze della classe **Automobile**.

9.2 Programmazione ad oggetti

Un progetto è costituito da più oggetti che operano indipendentemente e interagiscono secondo modalità prefissate. Ogni oggetto rappresenta un elemento del dominio del problema, che può rispondere a determinati stimoli provenienti dal mondo esterno. Ai fini della corretta interazione tra gli oggetti non è importante sapere come queste risposte vengono determinate. Ogni oggetto viene implementato separatamente e indipendentemente dagli altri. L'elemento fondamentale per la programmazione ad oggetti è la **classe**. Una classe ha un nome, e contiene due tipi di membri: **campi** e **metodi**. In Figura 9.1 viene presentato un esempio di classe. Un **oggetto** è una *istanza* (*esemplare*) di una classe, che viene creato (*istanziato*) dinamicamente. In Figura 9.2 vengono istanziati tre oggetti della classe **Automobile**. Due esemplari della stessa classe sono distinguibili soltanto per il loro stato (il valore dei loro campi), mentre il comportamento è sempre identico.

9.3 Programmazione in C++

Per descrivere le caratteristiche del linguaggio C++ iniziamo con un esempio di codice (C-like) che descrive un programma per la gestione di dischi

```
#include <iostream>
using namespace std;
void acquista_CD {cout << ‘‘sto acquistando’’; }
void controlla_CD {cout << ‘‘sto controllando’’; }
void conta_CD {cout << ‘‘sto contando’’; }
int main() {
    acquista_CD();
    controlla_CD();
    conta_CD();
 return 0;
}
```

Dal precedente codice possiamo identificare diverse parti che ci aiuteranno a capire il formato di un programma in C++. Possiamo identificare **#include <iostream>** come una direttiva al preprocessore che permette di include il contenuto del file **iostream.h** nel file del programma prima della compilazione. Se il file da includere non ha estensione fa parte della libreria standard. Le direttive al preprocessore sono istruzioni eseguite prima della compilazione **#include**, **#define**, **#ifdef**, **#ifnotdef**, **#else**, **#endif**, **#pragma** sono tutte direttive al preprocessore. Il simbolo # deve essere il primo carattere della linea non c'è **;** al termine dell'istruzione.
Inoltre, ogni variabile è definita all'interno di un **namespace**. Il nome della variabile è formato dal nome del namespace e dall'identificatore della variabile. L'istruzione **using namespace std** specifica il namespace di default. Se non specificato l'identificatore della variabile viene ricercato all'interno del namespace di default. Come nel caso del lionguaggio C il programma principale prevede l'utilizzo del **main** e le parti dei costrutti sono identificati dalle parentesi graffe, aperte e chiuse.

9.4 Compilazione linux

Per compilare un programma dalla linea di comando Unix/Linux-like bisogna digitare
g++ (-g) (-o out) (-c) file1.cpp file2.cpp file3.cpp (-llib)

Tipo	Dimensione	Utilizzo
char	1 byte	caratteri o piccoli interi
int	1 word	interi
short	dipende dalla macchina	
long	dipende dalla macchina	
float	1 word	reali in singola prescione
double	2 word	reali in doppia precisione
long double	3-4 word	reali in multipla precisione

Tabella 9.1: Tipi di dati definiti in C++.

dove le opzioni hanno il seguente significato

- -g debugger
- -o eseguibile in out (per default in a.out)
- -c compila ma non linka i file.o
- -l link libreria lib
- -I modifica path di ricerca nel file system

Il compilatore opera in due passi: compilazione e linkaggio. Il compilatore, inoltre, esegue controlli lessicali e sintattici sul codice del programma.

9.5 Tipi di dati definiti

I tipi di dati definiti nel linguaggio C++ sono gli stessi del linguaggio C: **char**, **int**, **short**, **long**, **float**, **double** e **long double**. Nella Tabella 9.5 vengono descritte le caratteristiche di questo tipo di dati.

9.6 Variabili e putatori

Le **variabili simboliche** introducono il nome ed il tipo di una variabile. Il compilatore alloca memoria sufficiente a contenere il dato ed assegna l'indirizzo di questa memoria all'identificatore della variabile. Un esempio di dichiarazione può essere ad esempio **char ch**. Se usiamo la parola **extern** consideriamo il nome di una variabile che è stata definita da qualche altra parte nel programma. In questo caso il compilatore non alloca memoria. Una variabile di tipo **puntatore** contiene un indirizzo. Tramite una variabile puntatore si può referenziare indirettamente una variabile. Un puntatore può contenere l'indirizzo di un oggetto o di una funzione. Esempi di dichiarazione di puntatori sono

```
-------------------------------------------------------------------------
 int *ip_1, *ip_2;           //ip_1 e ip_2 puntano ad int
 unsigned char *u_c_p;       //u_c_p punta ad un char senza segno
 float fp, *fp_1;            //fp è un float e fp_1 punta ad un float
 char* cp, cp_1;             // cp punta a char, cp_1 è un char
-------------------------------------------------------------------------
```

Inoltre esempi di utilizzo di puntatori sono

```
-------------------------------------------------------------------------
 int i = 1024;
 int *p = &i;
 int *ip = p;
 char *cp = 0; //inizializza a NIL

-------------------------------------------------------------------------
 int *ip = &i;
 int k = *ip;  // k = 1024
 k = ip;   ip = i;  //ERRORE

-------------------------------------------------------------------------
 int *ip = &i;
 *ip = k;   // i = k
 *ip = abs(*ip)  // i = abs(i)
 *ip = *ip + 1 ; // i = i + 1

-------------------------------------------------------------------------
 int *ip = &i;
 *ip = *ip + 2 // i = i + 2
```

```
ip = ip + 2;  // ip punta 8 byte più avanti di i
int ip2 = ip  4; //ip punta 26 byte dietro i
int k = ip  - ip2; // k = 4
-------------------------------------------------------------------------
```

9.7 Tipi di dati derivati

Un tipo di **dato derivato** è ottenuto a partire da tipi di dato predefiniti. Un dato può essere derivato attraverso l'utilizzo degli operatori $*$, &, [], definendo enumerazioni o definendo strutture complesse (i.e. **struct**).

9.7.1 Array

Un **array** (o **vettore**) è una struttura dati complessa, classificato come un costruttore di tipo e cioè consente di definire nuovi tipi di dati a partire da (come aggregati di valori di) tipi preesistenti. La sintassi per la dichiarazione di un array in C++ è quella derivante dal C. Alcuni esempi di inizializzazione di array mono e bidimensionali sono

```
-------------------------------------------------------------------------
int i_a[3] = {0, 1, 2};
int i_a[]  = {0, 1, 2};
int i_a[3] = {1};    // i_a[0] = 1, gli altri elementi sono 0
-------------------------------------------------------------------------
char a[3][4];    //array bidimensionale (matrice) 3 x 4
int i_a[3][4] = {{0, 1, 2, 3}, {4, 5, 6, 7}, {8, 9, 10, 11}};
int i_a[3][4] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11};
-------------------------------------------------------------------------
int i = i_a[2][3];
i = i_a[2,3];      // ERRORE
-------------------------------------------------------------------------
```

Gli array possono essere anche descritti tramite puntatori. Ad esempio

```
-------------------------------------------------------------------------
i_a[4] = 3;     // implementa *(i_a + 4) = 3
M[1][2]= 0;     // implelenta *(*(M + 1) + 2 ) = 0
-------------------------------------------------------------------------
char buffer[256];
char *p = buffer; // p indirizza il primo elemento di buffer
```

9.7.2 Struct

Il tipo strutturato record viene realizzato in C++, come in C, mediante il meccanismo delle strutture. Il meccanismo delle strutture (uso delle struct) consente di dichiarare specifici tipi derivati aggregati di variabili di tipo non omogeneo. Ad esempio una struct che descrive una data

```
struct data {
    int giorno;
    char mese[10];
    int anno;
 };
```

9.7.3 Espressioni e operatori

Un'**espressione** è formata da una o più operazioni. Gli operatori aritmetici sono: $+$, $-$, $*$, /, %. Il tipo del risultato di un'operazione aritmetica è uguale al tipo dell'operando più grande. Gli operatori **logico-relazionali** sono $>$, $<$, $>=$, $<=$, $==$, $!=$, &&, ||, !. Gli operatori di incremento e decremento sono $++$, $--$.
Ad esempio possiamo avere

```
i++ = \rightarrow  i = i + 1
i-- = \rightarrow  i = i - 1
```

Esistono due versioni dello stesso operatore

- $++i$ prima incrementa il valore di e poi lo legge
- $i++$ prima legge il valore di i e poi lo incrementa

Gli operatori ai bit sono %, |, $\sim$, $\wedge$, $<<$, $>>$.

9.8 Allocazione e deallocazione della memoria

Ogni programma è fornito di un certo spazio di memoria utilizzabile durante l'esecuzione che viene chiamato chiamato *heap* del programma. L'allocazione della memoria durante l'esecuzione è detta allocazione dinamica della memoria. Essa si ottiene applicando l'espressione **new** ad uno specificatore di tipo. L'espressione new restituisce un puntatore

all'oggetto appena allocato.

Come esempio consideriamo

```
int *pi = new int(10);      //alloca un int e lo inizializza
                            //con 10
int *pia = new int[4];      // allora un array di interi con 4
                            // elementi
```

Quando si finisce di usare l'oggetto applicando l'espressione **delate** possiamo liberare la memoria utilizzata. Ad esempio

```
-------------------------------------------------------------------------
delete pi;      //cancella la memoria allocata a pi
delete [] pia; //cancella il vettore allocato a pia

-------------------------------------------------------------------------
char *p = new char;  // p indirizza un char sulla free store
delete p;            // dealloca la memoria referenziata da p
-------------------------------------------------------------------------
// allocazione e deallocazione di array
int *pi = new int[size];
int (*pm)[4] = new int[3][4];
delete [] pi;
-------------------------------------------------------------------------
```

9.9 Istruzioni condizionali e cicli

L'esecuzione sequenziale di un programma non è sufficiente per la soluzione di problemi complessi. Speciali istruzioni di controllo di flusso consentono l'esecuzione condizionale o ripetuta di un'istruzione semplice o composta sulla base della valutazione di un'espressione logica. Nel linguaggio C++ queste epressioni sono le stesse del linguaggio C, per questo motivo ci limitiamo solo ad elencarle.

Il costrutto condizionale **if** ha la seguente forma

```
if (expr)
   ist_1;
else ist_2;
```

dove *expr* risulta l'espressione logica da valutare e ist_1 e ist_2 le istruzione da eseguire in base alla valutazione dell'espressione logica. In molti casi è convenienete usare l'operatore

```
switch(expr){
     case i:
          ist_i;
          break;
     case j:
          ist_j;
          break;
     default:
          ist;
}
```

Spesso si rende necessario ripetere delle istruzioni più volte. In questo caso si ricorre ai cicli. Come nel caso del linguaggio C, i costrutti sono il **while**

```
while (expr) ist
```

il ciclo **for**

```
for (ist_iniz; expr_1; expr_2 )
   ist;
```

e quello **do-while**

```
do
 istr;
while (expr);
```

9.10 Funzioni

Anche la dichiarazione e l'utilizzo di funzioni vengono effettuate, nel linguaggio C++, come nel linguaggio C. Mostriamo come esempio le seguenti funzioni

```
-------------------------------------------------------------------
// minimo
inline int min(int a, int b) {
        return (a < b) ? a : b;
}
-------------------------------------------------------------------
//valore assoluto
int abs(int i) {return (i > 0) ? i : -i;}
```

```
-----------------------------------------------------------------------
// MCD tra due valori
int mcd(int x, int y){
  int temp;
  while(y) {
      temp =  y;
      y = x % y;
      x = temp;
    }
  return x;
}
-----------------------------------------------------------------------
```

Una funzione **inline** viene espansa in tempo di compilazione. Il passaggio dei parametri ad una funzione può avvenire per **valore**, come ad esempio

```
int i = 3, j = 8;
cout << i << , << , << mcd(i,j);
// i = 3, j = 8, mcd(i,j) = 1
```

Il passaggio inoltre può avvenire come **puntatore**

```
int i = 3, j = 8;
swap(&i, &j);   // scambia i valori cout << i << ‘‘,’’ << j; // i = 8, j = 3
```

dove la funzione *swap* è definita come

```
void swap(int *v1, int *v2) {
  int tmp = *v2;
  *v2 = *v1; *v1 = tmp;
  }
```

In C++, inoltre, il passaggio dei parametri può avvenire per **riferimento**

```
void swap(int *v1, int *v2) {int tmp = *v2;
  *v2 = *v1; *v1 = tmp;}
int i = 3, j = 8;
swap(i, j);   // scambia i valori cout << i << ,
<< j; // i = 8, j = 3
```

dove in questo caso la funzione *swap* è definita come

```
void swap(int &v1, int &v2) {int tmp = v2;
  v2 = v1; v1 = tmp;}
```

Alcuni esempi per il passaggio di array a funzioni sono

```
-----------------------------------------------------------------------
void inizializza(int*); void inizializza(int[]);
void inizializza(int[10]);
-----------------------------------------------------------------------
void inizializza(int[][10]); // array 2D
```

9.11 Nomi globali

Un **namespace** definisce uno scope. Tutti gli identificatori definiti all'interno del namespace sono distinti dagli identificatori definiti in altri namespace. Un esempio di namespace è il seguente

```
namespace c_plus_plus {
           class matrix { . . . };
           void inversa(matrix&);
}
```

Un riferimento in altri namespace risulta

```
c_plus_plus::matrix m;

c_plus_plus::inversa(m);

::i   //varibile globale i
```

Namespace annidati sono

```
void c_plus_plus::Matrix_Lib::inversa(matrix &m);
```

Tutti i componenti della libreria standard del C++ sono definiti all'interno del **namesape std**. Per accedere agli elementi della libreria bisogna quantificarli con **std::**. Conviene, quindi, usare una direttiva **using**. Alcuni compilatori consentono di referenziare gli elementi della libreria standard senza specificare il nome del namespace.

9.12 Overloading

La sovrapposizione (**overloading**) è un meccanismo che consente di assegnare ad un nome significati differenti. Quando il nome viene utilizzato in una frase il significato corretto viene specificato dal contesto. Un esempio di overload della funzione *max* è il seguente

```
int max(int, int);    // massimo tra due interi
int max(int*, int);   // massimo in un vettore di interi
int max(Lista& );     // massimo in una lista di interi
```

9.13 Classe e oggetti

La **classe** è un meccanismo che consente di definire nuovi tipi di dato a partire da quelli esistenti. La classe può essere considerata come una generalizzazione del concetto di **struct**. La definizione di una classe è formata da due parti

- Testa della classe
  parola chiave class seguita dal nome della classe
- Corpo della classe
  dichiarazione di tutti i membri della classe, racchiusi tra ;

Il corpo della classe definisce uno scope e le dichiarazioni dei membri introducono gli identificatori nel namespace della classe. Un esempio è il seguente

```
class strumento_musicale {

       string nome_strumento;

...

};

strumento_musicale violino;
```

Ogni classe è caratterizzata da

- un insieme di zero o più oggetti (dati membro o **attributi**) che definiscono la rappresentazione in memoria della classe

- un insieme di zero o più funzioni (**metodi**) definiscono le modalità di utilizzo della classe

- un livello di accesso per ogni membro della classe definiscono chi può utlizzare ogni membro

- un nome della classe, che può essere utilizzato come specificatore di tipo.

Consideriamo come esempio la seguente classe **suono**

```
class suono {

        // Attributi
        short _lunghezza;
        short _freq_max;
        short _freq_min;
        int _posizione;
        string tipo_suono;
        double *_suono;



        // Metodi
        void origine() {_posizione = 0;}  //inline
        void forward();
        void forward(int);
        void backward();
        void backward(int);
        double play() {return suono[_posizione];}  // inline

};
```

Il linguaggio C++ definisce tre diversi livelli di accesso ai membri di una classe

- **private**
  accessibile solo dai metodi della classe stessa e dai suoi amici (classi friend)

- **public**
  accessibile da tutti

- **protected**
  accessibile solo dai metodi della classe, dagli amici della classe e dai metodi delle classi derivate

Nel corpo della classe bisogna specificare per ogni membro (sia attributo che metodo) il livello di accesso. Per default un membro ha livello d'accesso privato. Un esempio di mascheramento dell'informazione risulta

```
class suono {

  public:
       void origine() {_posizione = 0;}
       void forward();
       void forward(int);
       void backward();
       void backward(int);
       double play() {return suono[_posizione];}

 private:
       void verifica_dimensione();
       short _lunghezza;
       short _freq_max;
       short _freq_min;
       int _posizione;
       string tipo_suono;
       double *_suono;
};
```

Il progettista della classe può rendere i membri privati della classe visibili anche a funzioni esterne alla classe queste funzioni sono dette amiche (**friend**) della classe

```
class suono {

  friend void filtro (const suono&);

...

};  // filtro potrebbe accede agli attributi di suono
```

Dopo aver definito una classe possiamo istanziare e usare degli oggetti come nel seguente esempio

```
suono s, *ps;


s.origine();


s->forward();
```

Per l'inizializzazione di ogni membro di a con il corrispondente metodo di s possiamo considerare

```
suono s;
suono a = s;
```

L'allocazione dinamica della memosria per un oggetto risulta la seguente

```
suono *ps = new suono;
```

Un metodo può essere definito sia all'interno della definizione della classe che al suo esterno (anche in un altro file). Quando il compilatore legge la definizione del metodo deve vedere la definizione della classe. Ogni metodo deve essere dichiarato nella classe. Un metodo definito all'esterno deve essere qualificato con l'operatore di scope **::**

```
suono::verifica_dimensione(){}
```

I metodi definiti nella definizione della classe sono considerati **inline**. Lo scopo è quello di definire all'interno della classe solo funzioni molto semplici. I metodi definiti all'esterno possono essere qualificati inline usando la parola inline (nella definizione)

```
inline suono::forward(){}
```

Un ruolo fondamentale risulta quello del **costruttore** della classe. Esso permette di costruire ed inizializzare un oggetto allocando memoria. Il costruttore deve avere lo stesso nome della classe. Un esempio di costruttore è il seguente

```
suono::suono (int l, int f_min, int f_max) {


        _dimensione = l;
        _freq_min = f_min;
        _freq_max = f_max;
```

```
        _posizione = 0;

        double _suono = new double[_lunghezza];
}
```

Per deallocare la memoria posseduta da un oggetto, in C++ e non in Java, deve essere usato il **Distruttore**

```
suono:: ~suono () {
      delete [] _suono;
}
```

Un esempio di metodo della classe **suono** può essere il seguente

```
void suono::forward() {

   if (_posizione <= _dimensione)
   _posizione++;
  else cout << BELL << endl;
}
```

Un metodo di una classe accede ai dati membro della classe attraverso il puntatore implicito **this**. Il puntatore this può essere utilizzato per concatenare più operazioni da eseguire in sequenza sullo stesso oggetto

```
suono S(1000,20,20000);
S.forward(10);
S.play();
S.forward(10).play();
```

Un metodo che usa this deve essere implementato nel seguente modo

```
suono& suono::forward() {
   if (_posizione <= _dimensione)
   _posizione++;
  else _posizione = 0;
 return *this;
}
```

9.13.1 Costruttore e Overloading

I meotodi e i costruttori di una classe possono essere sovraccaricati. Consideriamo ad esempio la seguente classe **canzone**

```
class canzone {
 public:
      canzone(const char*, double, int);
 private:
      char *_titolo;
      unsigned int _codice;
      double costo;
};
-----------------------------------------------------------------------
canzone mia_canzone(blue room, 2.50, 10);
-----------------------------------------------------------------------
```

Anche un costruttore può essere sovraccaricato. Consideriamo infatti il seguente costruttore

```
class canzone {

friend class vendita_mp3;

 public:

      canzone(const char*, double = 0);

 private:

      canzone(const char*, double = 0, int);

      char *_titolo;

      unsigned int _codice;

      double costo;

};
-----------------------------------------------------------------------
canzone mia_canzone(blue room);
-----------------------------------------------------------------------
```

9.13.2 Utilizzo di una classe

Un esempio di utilizzo di un costruttore è il seguente

```
------------------------------------------------------------------------
canzone mia_canzone(‘‘Blue Room’’);
------------------------------------------------------------------------
canzone mia_canzone = canzone(‘‘Blue Room’’);
------------------------------------------------------------------------
canzone mia_canzone = ‘‘Blue Room’’;  // solo con
------------------------------------------------------------------------
cotruttore con singolo argomento
------------------------------------------------------------------------
canzone *p_mia_canzone = new canzone(Blue Room);
------------------------------------------------------------------------
```

Un distruttore, invece, può essere definito come

```
canzone:: \sim canzone(){
   delete [] _titolo;
   restituisci_codice(_codice);
}
```

Una delle operazioni più usate è quella di assegnamento. Un esempio di codice relativo all'assegnamento risulta

```
canzone& canzone::operator=(const canzone& c)?


    if(this != c) {
    _titolo = c._titolo;
    _costo  = c._costo;
    _codice = _genera_codice();
   }


  return *this;
}
```

9.13.3 Static

Un **attributo static** opera come variabile globale per tutti gli oggetti della classe. Un **metodo static** viene utilizzato per accedere ai dati static della classe. Un metodo static

può essere invocato senza utilizzare l'operatore di selezione. E' possibile accedere ai dati static anche se nessun oggetto della classe è stato creato. Un esempio di campo static è il seguente

```
-------------------------------------------------------------------
class canzone {


 public:
      static int vendute;


}C_1,C_2;
-------------------------------------------------------------------
C_1.vendute = 1;
Cout << C_2.vendute << endl; // stampa 1
-------------------------------------------------------------------
```

Un metodo static invece risulta

```
class canzone {
 public:
      static double iva() {
         return _iva; }
 private:
      static double _iva;
};
canzone::iva();
```

9.13.4 Classi annidate

Le classi possono contenere altre classi (**annidate**). Un esempio è il seguente

```
class canzone {public: unsigned int _codice};


class lista_canzoni {
  public:
    class canzone {
            canzone *next;
           public:
            canzone() {next = 0;}
          };
```

```
        canzone *lista;
}L;
-------------------------------------------------------------------------
cout << L.(lista->_codice) << endl; // ERRORE ?
-------------------------------------------------------------------------
```

9.13.5 Template di funzione

Il **template di funzione** granatisce una funzione parametrizzata. Un esempio di dichiarazione ed utilizzo di una funzione template è il seguente

```
-------------------------------------------------------------------------
template <class T>
T max(T* a, int dim){
    T tmp = a[0];
    for ( int i = 1; i < dim; i++ )
        if ( a[i] > tmp )
            tmp = a[i];
    return tmp;
};

-------------------------------------------------------------------------
template <class T> T max (T*, int);

int ai[4] = {12, 8, 73, 45};

int size;

. . .

max(ai, size)
-------------------------------------------------------------------------
```

Anche una classe può essere parametrizzata. Ad esempio consideriamo la seguete classe **coda**

```
template <class T>
class coda {

    T *vett;
```

```
  . . .

};

-------------------------------------------------------------------------
coda <int> qi;
-------------------------------------------------------------------------
coda <string> qs;
-------------------------------------------------------------------------
```

9.14 Programmazione ad oggetti

La programmazione ad oggetti si basa su due principi fondamentali: **ereditarietà** e **polimorfismo**. Queste due proprietà consentono di definire nuovi tipi di dato in funzione di tipi già esistenti. In questo modo si specifica soltanto le parti in cui la nuova classe differisce da quelle note (programmazione per differenze) e consente di riciclare tutto il codice presente nelle librerie.

9.14.1 Ereditarietà

Il meccanismo dell'ereditarietà permette di definire una classe in termini di una classe definita in precedenza. La nuova classe eredita la rappresentazione in memoria e l'interfaccia della sua classe base. Il programmatore non deve riscrivere tutto il codice che è in comune tra la classe base e la nuova classe e può modificare l'implementazione di alcuni dei metodi ereditati. L'ereditarietà consente di implementare relazioni di specializzazione tra tipi di dato. Un esempio di generalizzazione ed ereditarietà è mostrato in Figura 9.3.

9.14.2 Polimorfismo

Il polimorfismo consente di programmare senza tener conto dei dettagli relativi all'implementazione di una gerarchia di classi. L'utente opera sull'interfaccia pubblica della classe base. Un oggetto di qualunque classe della gerarchia contiene l'interfaccia della classe base. L'implementazione dei metodi può essere differente. Quando il programma invoca

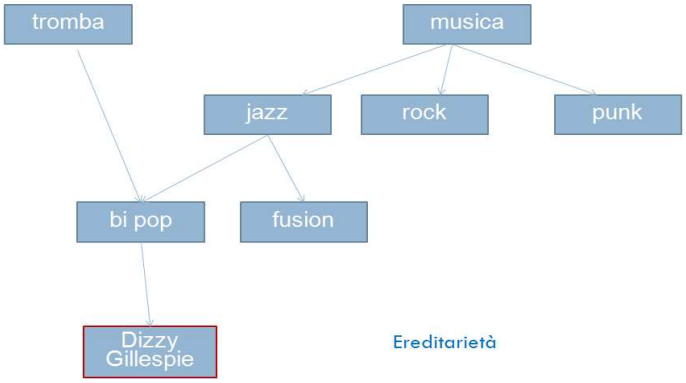


Figura 9.3: Esempio di ereditarietà.

un metodo della classe base attraverso un oggetto viene eseguita l'implementazione del metodo contenuta nella classe a cui appartiene l'oggetto.

9.14.3 Derivazioni di classi

Il meccanismo della derivazione di classi implementa l'operazione di ereditarietà della programmazione ad oggetti La classe derivata può utilizzare tutti i dati ed i metodi non privati della classe base come propri elementi non c'è bisogno di riscrivere i metodi ereditati dalla classe base, a meno che non debbano essere modificati Il livello di accesso di un membro della classe può essere privato, pubblico e protetto I membri protetti non sono visibili dall'esterno della classe ma sono ereditabili.

```
class canzone: public suono {

 public:

       void forward(unisgned short, int, int); // ridefinzione
       double dammi_la_codifica();  // nuovo metodo

 ...

 protected:
      char *_titolo;
      unsigned int _codice;
      double costo;
      unsigned short tipo_codifica;
```

```
};
```

Ogni classe può specificare una o più classi base da cui ereditare dati e metodi. L'insieme delle classi e le relazioni di derivazione formano un *grafo di derivazione*. Se ogni classe ha una sola classe base il grafo è un albero. Se, invece, le classi hanno più classi base il grafo è un Grafo Diretto Aciclico (DAG). Inoltre il grafo di derivazione non può mai contenere cicli. Ogni classe del grafo contiene tutti gli elementi che sono comuni a tutte le sue classi derivate. Consideriamo come esempio la seguente classe che descrive gli strumenti a corda

```
class strumento_a_corde{

 ...


 protected:

      int _num_corde;
      int _num_tasti;
      vector<corda> _note;
      _larghezza(const corda&, const corda&);

};

class strumento_a_corde{


 public:
    strumento_a_corde();
    strumento_a_corde(int N, vector<corda>&);
    virtual double freq_max() const;
    double numero_corde();

. . .

};
```

Se l'implementazione dipende dalla classe derivata la funzione è dichiarata virtuale (**virtual**). Esempi di classi derivate dalla classe **strumento a corde** possono essere

```
class chitarra: public strumento_a_corde{

 public:
    . . .

    void suona_nota(corda&);
    double freq_max();
};

class mandolino: public strumento_a_corde{

public:
    . . .
    void suona_nota(corda&);
    double freq_max();
};
```

L'accesso ai membri ereditati può venire nel seguente modo

```
chitarra *C;

C->freq_max();  //chitarra::freq_max()

C->numero_corde(); // strumento_a_corde::numero_corde;
```

Nel caso di ereditarietà multipola possiamo avere il seguente caso

```
class chitarra: public strumento_a_corde, public strumento_acustico{
    . . .
};
```

Per quanto riguarda il costruttore della classe base derivata possiamo avere il seguente caso

```
chitarra::chitarra(vector<corda>  c):strumento_a_corde(6, c){};
```

Il costruttore della classe base, invece,

```
-------------------------------------------------------------------
class strumento_a_corde{

 public:
```

```
    strumento_a_corde();

 protected:
    strumento_a_corde(double c, double t);

};
-------------------------------------------------------------------------
strumento_a_corde::strumento_a_corde:_num_corde(0), num_tasti(0) {}
-------------------------------------------------------------------------
strumento_a_corde::strumento_a_corde(double c, double t):_num_corde(c),
-------------------------------------------------------------------------
num_tasti(t) {}
-------------------------------------------------------------------------
```

Infine abbiamo che una funzione virtuale viene specificata premettendo al suo prototipo la parola virtual all'interno della definizione della classe

```
class strumento_a_corde{

private:

  virtual double freq_max()

};
```

Le classi derivate possono ridefinire la funzione virtuale, modificandone l'implementazione. La definizione della funzione virtuale nella classe derivata deve avere la stessa firma della funzione ereditata

```
class chitarra: public strumento_a_corde{

public:

   double freq_max();

};
```

Nel caso in cui la classe contiene più sotto-oggetti l'eridarietà virtuale permette di ereditare un solo sotto-oggetto. Ad esempio **iostream** contiene due sotto-oggetti **ios**

```
class iostream: public istream, public ostream {

 . . .

};
```

Una classe base virtuale può essere dichiarata nel seguente modo

```
class chitarra: virtual public strumento_a_corde{

 . . .

};
```

9.15 Le eccezioni

Le **eccezioni** sono organizzate in gerarchie di classi ed è un'istanza di una classe. Può essere intercettata usando le clausole **try** e **catch**.

```
try {

 . . .

}

. . .

catch( Excp ) {

 . . .

}
```

9.16 I/O

Il linguaggio C++ non prevede istruzioni specifiche per gestire le operazioni di input ed output di un programma. L'input e l'output può essere gestito attraverso librerie progettate in C++. La libreria iostream è stata identificata come la libreria standard del

C++ è presente in tutti i compilatori. Volendo, l'utente può utilizzare librerie differenti (per esempio **stdio**). Le funzionalità di I/O sono fornite attraverso la libreria **iostream**. Per usare la libreria bisogna includere il file header iostream. Le operazioni di input sono fornite dalla classe istream e quelle di output dalla classe **ostream**.
L'otput è ottenuto principalmente mediante l'operatore << e l'input mediante l'operatore >>.

```
# include <iostream>
# include <string>

int main(){
string int_string;

cout <<  ‘‘Dammi il tuo nome’’;
cin >> in_string;

return 0;
}
```

9.16.1 I/O su file

Un utente che voglia usare un file per leggere e scrivere i dati deve includere il file header **fstream**

```
#include<fstream>
```

Per aprire un file soltanto per l'output si definisce un oggetto della classe **ofstream**.

```
oftream outfile(‘‘copy.out’’, ios_base::out); // out modalità output o app per append
```

Ad esempio per scrivere caratteri usiamo

```
outfile.put(char);
```

e per chiudere il file

```
outfile.close();
```

Per inserire informazioni nel file usiamo

```
outfile << ‘‘inf’’ << endl;
```

Infine per inserire informazioni sulla precisione richiesta per la registrazione di una cariabile possiamo usare

```
outfile.precision(#);
```

9.16.2 Classe dei numeri complessi

La classe **complex** fornita dalla libreria standard è un esempio di astrazione basata sugli oggetti. Mediante il sovraccaricamento degli operatori, oggetti della complex possono essere usati quasi con la stessa facilità dei più semplici tipi aritmetici predefiniti. Sono supportati gli operatori aritmetici comuni: addizione, sottrazione, moltiplicazione e divisione. Sono supportati anche oggetti complex con oggetti dei tipi aritmetici predefiniti. Un uso della classe è

```
#include<complex>
            complex<double> a;
            complex<double> b;
            complex<double> c= a * b + a / b;
```

Il numero complesso z = a + jb può essere inizializzato come

```
complex<double> z(a,b);
```

Dato un numero complex z si può ottenere la parte reale nel seguente modo:

```
z.real();
z.imag();
```

Bisogna notare che nella libreria ESA-DSP viene fornita una diversa versione della classe ma CHE HA le stesse caratteristiche. La classe è descritta nella Tabella e la sua implementazione risulta la seguente

```
// Classe per la definizione di numeri complessi
class complex {


public:


  // costruttori
  complex() {z.a = 0.0; z.b = 0.0;};
  complex(double a) {z.a = a; z.b = 0.0;};
```

complex				
#	z	:	struct {double a, b;}	numero complesso
+	complex ()			costruttore
+	complex (double)			costruttore
+	complex (double, double)			costruttore
+	∼complex()	:		distruttore
+	real()	:	double	parte reale
+	imag()	:	double	parte immaginaria
+	cabs()	:	double	valore assoluto
+	conj()	:	void	coniugato
+	add()	:	void	addizione
+	sub()	:	void	sottrazione
+	rmul()	:	void	divisione scalare
+	div()	:	void	divisione
+	rdiv()	:	void	divisione scalare
+	div()	:	void	divisione
+	cexp()	:	void	esponenziale
+	cexp(double, double)	:	complex&	esponenziale
+	cexpc(double)	:	complex&	esponenziale
+	put(double, double)	:	complex&	inizializzazione
+	operator*(complex)	:	complex&	operatore
+	operator+=(complex)	:	complex&	operatore
+	operator-(complex)	:	complex&	operatore
+	operator=(complex)	:	complex&	operatore
+	plus(complex,complex)	:	complex&	addizione
+	minus(complex,complex)	:	complex&	addizione
+	prod(complex,complex)	:	complex&	addizione

Tabella 9.2: Classe **complex**.

```
complex(double a, double b) {z.a = a; z.b = b;};

//distruttore
~complex(){};


// parte reale
double real()
   {return z.a;}

// parte immaginaria
double imag()
   {return z.b;}

// valore assoluto
double cabs()
{
     return sqrt(z.a * z.a + z.b * z.b);
 }

 // complesso coniugato  di z = x + jy
 void conj()
  {
    z.b = - z.b;
  }

 // addizione complessa
 void add(complex x)
  {
     z.a += x.real(); z.b += x.imag();
  }


 // sottrazione complessa
void sub(complex x)
{
     z.a -= x.real(); z.b -= x.imag();
}
```

```
// moltiplicazione scalare
void r_mul(double c) {
     z.a *= c; z.b *= c;
}

 // divisione complessa
void div(complex x)
{
  double D = x.real() * x.real()+ x.imag() * x.imag();
  z.a = (z.a * x.real() + z.b * x.imag()) /D;
  z.b = (z.b * x.real() - z.a * x.imag()) /D;
  }

 // divisione per uno scalare
void r_div(double c)
  {
     z.a /= c; z.b /= c;
  }

  // esponenziale complesso

  void c_exp()
  {
     double R = exp(z.a);
     z.a = R * cos(z.b); z.b = R * sin(z.b);
  }

  complex& c_exp(double a, double b)
  {
      z.a = a * cos(b); z.b = a * sin(b);
  return *this;
  }


  complex& c_exp_c(double A)
  {
     z.a =  A*cos(z.a); z.b = A*sin(z.b);
  return *this;
```

```
  }


  // inizializzazione della parte reale ed immaginaria
  complex& put(double x, double y)
   {z.a = x; z.b = y; return *this;}


  // OPERATORI
  complex& operator*(complex x)
  {
    z.a = z.a * x.real() - z.b * x.imag(); z.b = x.imag() * z.a + x.real() * z.b;
return *this;
  }

  complex& operator+=(complex x)
  {
    z.a = x.real() + z.a; z.b = x.imag() + z.b;
return *this;
  }

  complex& operator-(complex x)
  {
    z.a = z.a - x.real() ; z.b = z.b - x.imag();
return *this;
  }

  complex& operator+(complex x)
  {
    z.a = z.a + x.real() ; z.b = z.b + x.imag();
return *this;
  }

  complex& operator=(complex x)
  {
    z.a = x.real() ; z.b = x.imag();
return *this;
  }
```

```
    // ADDIZIONE
    complex& plus(complex x, complex y)
    {
  z.a = x.real() + y.real(); z.b = x.imag() + y.imag();
      return *this;
    }

    // SOTTRAZIONE
    complex& minus(complex x, complex y)
    {
  z.a = x.real() - y.real(); z.b = x.imag() - y.imag();
      return *this;
    }



    // PRODOTTO
    complex& prod(complex x, complex y)
    {
      z.a = x.real() * y.real() - x.imag() * y.imag(); z.b = y.imag() * x.real() + y.real() * x.imag();
  return *this;
    }

private:
struct z_c {double a, b;} z;
};
```

Bibliografia

[1] G. Giunta, *Elaborazione numerica dei segnali*, Università di Roma "La Sapienza", (http://www.comlab.uniroma3.it/ens.htm)

[2] V. Lombardo, A. Valle, *Audio e Multimedia*, Apogeo, II edizione 2005

[3] A. V. Oppenheim, R.W. Schafer, *Elaborazione numerica dei segnali*, Franco Angeli Editore, 1996

[4] D. Rocchesso, *Introduction to Sound Processing*, Mondo Estremo 2003. (http://www.mondoestremo.com/publications/publications/publications/public.html)

[5] S. W. Smith, *Guide to DSP*, (http://www.dspguide.com)

[6] Aurelio Uncini, *Audio Digitale*, McGraw-Hill, 2006

[7] *Audio Multimediale*, (http://audiosonica.blog.excite.it/)