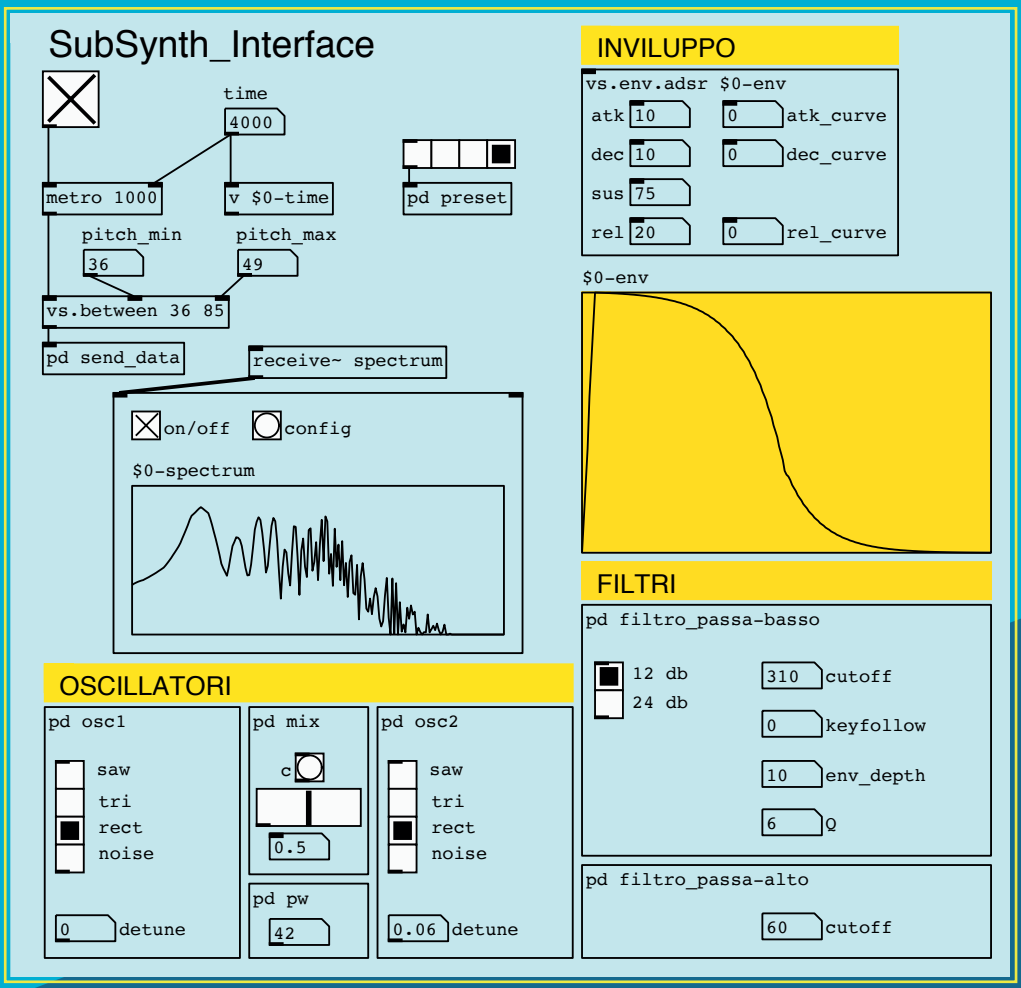
Francesco Bianchi • Alessandro Cipriani • Maurizio Giri



Pure Data: Musica Elettronica e Sound Design

Teoria e Pratica • volume 1

ConTempoNet

Francesco Bianchi • Alessandro Cipriani • Maurizio Giri

Questo è un estratto del libro:

PURE DATA:
MUSICA ELETTRONICA E SOUND DESIGN

Teoria e Pratica - Volume 1

per maggiori informazioni:

www.contemponet.com
www.virtual-sound.com

BIANCHI F. - CIPRIANI A. - GIRI M.
PURE DATA: MUSICA ELETTRONICA E SOUND DESIGN
Teoria e Pratica
Vol. 1
ISBN 978-88-99212-05-6

Prima edizione 2016

Realizzazione figure: Gabriele Cappellani e Maurizio Refice
Realizzazione esempi interattivi: Vincenzo Core
Consulenza glottodidattica: Damiano De Paola

ConTempoNet s.a.s., Roma
e-mail posta@contemponet.com
posta@virtual-sound.com
URL: www.contemponet.com
www.virtual-sound.com

INDICE

INTRODUZIONE

Questo è il primo di una serie di volumi sulla sintesi e l'elaborazione digitale del suono con Pure Data. L'opera è un adattamento della serie "Musica Elettronica e Sound Design - Teoria e Pratica con Max" di A. Cipriani e M. Giri .
Gli argomenti trattati nel presente volume sono: Introduzione alla Sintesi del Suono, Sintesi Additiva, Sintesi Sottrattiva e Filtri, Segnali di Controllo. I successivi volumi tratteranno i seguenti argomenti: Campionamento e Audio Digitale, Processori di Dinamica, Linee di Ritardo, MIDI e tempo reale, l'Arte dell'Organizzazione del Suono, Riverbero e Spazializzazione, Tecniche di Sintesi Non Lineare (come AM, FM, waveshaping e tecniche di distorsione del suono), Sintesi Granulare, Analisi e Risintesi, Modelli Fisici, Filter Design.

LIVELLO RICHIESTO

Tutti i volumi alternano parti teoriche a sezioni di pratica al computer, che vanno studiate in stretta connessione. Questo primo volume può essere utilizzato da utenti di diverso livello di preparazione.
Il livello minimo richiesto per chi inizia a studiare il Vol.1 comprende:

- i primi rudimenti di teoria musicale (note, scale, accordi etc.)
- una competenza di base nell'utilizzo di un computer (saper salvare un file, copiare, cancellare etc.).

Il testo va studiato alternando ogni capitolo di teoria a quello corrispettivo di pratica incluse le attività al computer. La parte teorica non è sostitutiva di testi teorici sulla sintesi. Si tratta, invece, di un indispensabile compendio teorico al lavoro pratico di programmazione e di invenzione di suoni al computer, ed è parte quindi di un sistema didattico organico. Il percorso di questo volume può essere svolto in auto-apprendimento oppure sotto la guida di un insegnante.

I TEMPI DI APPRENDIMENTO

I tempi di apprendimento, come è ovvio, sono diversi da persona a persona. In particolare daremo conto di tempi di mero riferimento nelle due modalità: auto-apprendimento e apprendimento sotto la guida di un docente esperto.

Auto-apprendimento (300 ore globali di studio individuale)

Capitoli	Argomento	Totale ore
1T+1P+IA	Sintesi del suono	100
2T+2A	Sintesi Additiva	60
3T+3P+IB	Sottrattiva e filtri	110
4T+4P	Segnali di Controllo	30

Apprendimento con docente (corso di 60 ore in classe + 120 di studio individuale)

Capitoli	Argomento	Lezioni	Feedback	Studio	Totale ore
1T+1P+IA	Sintesi del suono	16	4	40	60
2T+2P	Sintesi Additiva	10	2	24	36
3T+3P+IB	Sottrattiva e filtri	18	4	44	66
4T+4P	Segnali di controllo	5	1	12	18

GLI ESEMPI INTERATTIVI

Il percorso della parte teorica è accompagnato da molti esempi interattivi reperibili sul sito **www.**********. Utilizzando questi esempi, si può fare esperienza immediata del suono e della sua creazione ed elaborazione senza aver ancora affrontato alcun lavoro pratico di programmazione. In questo modo lo studio della teoria è sempre in connessione con la percezione del suono e delle sue possibili modificazioni. Far interagire percezione e conoscenza nello studio del sound design e della musica elettronica è stato da sempre un nostro obiettivo, e questo criterio guida l'intera opera didattica, comprensiva anche di ulteriori materiali online che verranno man mano aggiornati ed ampliati.

TEORIA E PRATICA

L'impostazione didattica è basata proprio sull'interazione (per noi imprescindibile) fra teoria e pratica. Uno dei problemi nel campo dell'elaborazione del suono, infatti, è quello di avere esperti di teoria che normalmente non si trovano ad affrontare problemi concreti riguardanti la pratica dell'invenzione del suono, e persone (molto più numerose) che amano lavorare al computer con i suoni, ma che spesso hanno una scarsa coscienza tecnico-teorica di cosa stiano facendo, e una scarsa capacità di modificare ciò che i software che utilizzano li "costringono" a fare. Il mercato propone sempre più oggetti tecnologici meravigliosi, ma difficili da personalizzare. Un'informazione spesso approssimativa e poco sistematica, unita alla rapida obsolescenza dei sistemi, contribuisce a mantenere gli utenti in una (apparentemente piacevole) ignoranza, e quindi in una condizione di scarsa libertà, costringendoli, in un certo senso, ad usare le macchine e il software che acquistano in modo superficiale e ad aggiornarle continuamente spesso senza averne compreso la natura profonda. In questo senso intraprendere lo studio di questo libro significa anche iniziare ad acquisire una maggiore consapevolezza dell'uso dei software commerciali di sintesi ed elaborazione del suono.

L'IMPOSTAZIONE DIDATTICA

Sulla base dei concetti appena esposti, abbiamo pensato di colmare il vuoto di informazione riguardante questa materia, avanzando nella direzione intrapresa da Cipriani e Bianchini con il testo "Il Suono Virtuale", dedicato alla sintesi ed elaborazione del suono. La differenza con quel testo è grande, sia per la qualità degli esempi proposti, sia perché l'impostazione didattica è completamente diversa. Esiste pochissima bibliografia sulla metodologia didattica della musica elettronica. A questo scopo abbiamo riflettuto sulla possibilità di approfondire questa tematica e progettare finalmente un sistema didattico organico, mutuando alcune idee e tecniche dalla didattica delle lingue straniere, in modo da sviluppare una concezione più aperta e interattiva dell'insegnamento e dell'apprendimento.

Per questo abbiamo inserito, oltre agli esempi interattivi, anche contratti formativi per ogni capitolo, attività di ascolto e analisi, test, glossari, indicazioni discografiche e introduzioni storiche (online) oltre a tante altre novità contenute nei capitoli di pratica come le attività di sostituzione di parti di algoritmi, correzione, completamento e analisi di algoritmi, costruzione di nuovi algoritmi, compiti di reverse engineering (cioè, a partire dall'ascolto di

un suono, cercare di inventare un algoritmo che possa creare un suono simile a quello ascoltato). Nel corso dei capitoli la presenza di tali attività viene segnalata con l'icona visibile qui a lato.
Il sistema, composto da più volumi e una sezione online è multi-piattaforma, e la teoria è costruita in modo tale da poter fare da base a possibili altri testi di pratica basati su software diversi, utilizzando lo stesso percorso didattico.

PURE DATA

Questo testo si basa sul volume di A. Cipriani e M. Giri "MUSICA ELETTRONICA e SOUND DESIGN – Teoria e Pratica con Max[1]", di cui mantiene intatta la struttura e la vocazione didattica. La parte dedicata alla pratica, in cui si esemplificano le tecniche esposte nei capitoli di teoria, fa riferimento al software Pure Data, un programma creato, al pari di Max, da Miller Puckette, e che con Max condivide l'impostazione generale e molte altre caratteristiche.
Pure Data è un linguaggio di programmazione visuale, in cui oggetti grafici vengono collegati tra loro. Tali oggetti, che possono eseguire calcoli o elaborare segnale audio, vengono connessi al fine di creare entità anche molto complesse come sintetizzatori e processori di segnale, fino ad autentici dispositivi sonori automatici.
Pure Data è un software gratuito e *Open Source*: può essere liberamente scaricato e utilizzato sui più diffusi sistemi operativi; è anche possibile manipolare il codice sorgente al fine di creare versioni personalizzate del programma. Queste caratteristiche hanno stimolato la nascita di una nutrita comunità di musicisti, programmatori e appassionati, cui hanno fatto seguito alcuni progetti interessanti, fra i quali spicca *Pd-extended* (che purtroppo non viene più aggiornato e che, prima di cessare di esistere, è stata la versione di riferimento della comunità).
Questo testo fa uso della versione originale di Pure Data (chiamata talvolta *Pd-vanilla*), che continua ad essere aggiornata e mantenuta dallo stesso Miller Puckette. Il vantaggio della versione *vanilla* è quello di garantire la compatibilità con tutte le piattaforme e soprattutto una maggiore continuità nel tempo rispetto alle versioni alternative.
Il programma, nell'edizione *vanilla*, contiene una libreria nativa di oggetti non molto nutrita, per cui questo testo è corredato da una libreria di *abstractions* (oggetti creati all'interno dell'ambiente, costituiti cioè da oggetti di Pd già esistenti) che consente di potenziare le funzionalità del programma e di facilitare la realizzazione di operazioni altrimenti difficoltose a uno stadio iniziale di apprendimento.

INDICAZIONI PRATICHE

A corredo di questo libro sono stati realizzati molti materiali assolutamente indispensabili per procedere nell'apprendimento: esempi interattivi, *patch* (ovvero programmi scritti in Max), *sound file*, estensioni di libreria e altri materiali di supporto si trovano tutti all'indirizzo **www.**********

[1] (Cipriani e Giri, 2016)

Esempi Interattivi
Durante lo studio della teoria, prima di affrontare la parte pratica, è importante utilizzare gli esempi interattivi. Lavorare con questi esempi sarà di notevole aiuto per affrontare poi la parte pratica relativa all'argomento trattato.

File di esempio
I file di esempio (patch) sono utilizzabili con il software Pure Data, liberamente scaricabile dal sito di Miller Puckette http://msp.ucsd.edu/software.html. Come abbiamo detto, il libro si basa sulla versione *vanilla* (senza estensioni) di Pure Data.

Alternanza di Teoria e Pratica
Nel libro i capitoli di teoria si alternano ai capitoli di pratica. Il lettore si troverà quindi ad affrontare tutto un capitolo di teoria per poi passare al corrispettivo capitolo di pratica (ad esempio tutto il capitolo 1T e poi tutto il capitolo 1P). In alternativa può scegliere di leggere un paragrafo di teoria e subito dopo il paragrafo corrispondente di pratica per poi passare al paragrafo successivo (ad esempio 1.1T e 1.1P, poi 1.2T e 1.2P etc.).

Gli Interludi
Da notare che fra il primo e il secondo capitolo, e fra il terzo e il quarto capitolo ci sono 2 "interludi" tecnici, rispettivamente l'Interludio A e l'Interludio B, dedicati specificamente al linguaggio Pure Data e non legati ai temi trattati nella teoria, ma ugualmente necessari per procedere nel percorso tracciato nel libro. Dopo aver affrontato la teoria e la pratica del primo capitolo, prima di passare al secondo capitolo è fondamentale studiare l'interludio A. Ciò vale, ovviamente anche per l'interludio B, da studiare subito dopo aver completato i capitoli 3T e 3P.

L'apprendimento di Pure Data
L'apprendimento di Pure Data (e in generale della sintesi ed elaborazione del suono) richiede applicazione e concentrazione. Al contrario di molti software commerciali, infatti, Pure Data consente una flessibilità massima nella programmazione, e quindi consente una grande libertà a chi programma gli algoritmi; ma per poter usufruire di questa libertà è fondamentale evitare di saltare i passaggi consigliati nel libro e procedere in modo sistematico. Un apprendimento "intuitivo"o a salti dà scarsi risultati in Pure Data, specialmente all'inizio del percorso di apprendimento. Questo software è un vero e proprio "strumento musicale" e va studiato come si studierebbe uno strumento tradizionale (ad esempio un violino); è necessario cioè utilizzarlo con continuità, partendo dagli esercizi di base e affrontando via via le tecniche più complesse per evitare di dimenticare le conoscenze e di perdere le abilità acquisite. Solo così sarà possibile arrivare a una vera padronanza del programma.

Bibliografia e sitografia
Si è scelto di inserire nel testo soltanto una bibliografia assolutamente essenziale, e i riferimenti bibliografici relativi ai testi citati nel libro. Una bibliografia più completa e una sitografia è disponibile online.

Prima di cominciare
Per lavorare con questo testo è necessario accedere agli Esempi Interattivi che si trovano alla pagina di supporto **www.**********.
Durante la lettura dei capitoli di teoria si farà costante riferimento agli esempi contenuti nel sito. Per affrontare la parte pratica è invece necessario, come abbiamo detto, aver installato il programma Pure Data, reperibile al sito http://msp.ucsd.edu/software.html. Bisogna inoltre scaricare la libreria Virtual Sound Macros dalla pagina di supporto di questo testo (www.******************); nella stessa pagina troverete istruzioni dettagliate sulla procedura da seguire per la corretta installazione della libreria. Sempre a partire dalla pagina di supporto troverete le patch (programmi Pure Data) relative a tutti i capitoli di pratica e i file audio per gli esercizi di reverse engineering.

Commenti e segnalazioni
Correzioni e commenti sono benvenuti. Vi preghiamo di inviarli per e-mail a:
Francesco Bianchi frabianchi72@gmail.com
Alessandro Cipriani a.cipriani@edisonstudio.it
Maurizio Giri maurizio@giri.it

https://francescobianchiblog.wordpress.com/
http://www.edisonstudio.it/alessandro-cipriani/
http://www.giri.it

RINGRAZIAMENTI
Si ringraziano: Gabriele Cappellani e Vincenzo Core per il loro paziente e lungo lavoro; Eugenio Giordani, Giuseppe Emanuele Rapisarda e Fausto Sebastiani per la loro disponibilità.

DEDICA
Questo testo è dedicato a Riccardo Bianchini, che avrebbe voluto realizzare anche quest'opera didattica, ma che purtroppo è prematuramente scomparso prima che il lavoro iniziasse. Abbiamo raccolto alcuni suoi materiali (anche inediti), li abbiamo editati e citati in alcuni paragrafi di teoria. Questo è stato un modo, idealmente, per avere Riccardo ancora con noi. Un ringraziamento particolare va ad Ambretta Bianchini, per la grande disponibilità e sensibilità dimostrataci in questi anni di lavoro.

Buona lettura,
Francesco Bianchi, Alessandro Cipriani e Maurizio Giri

LEGENDA DEI SIMBOLI UTILIZZATI

- **ATTIVITÀ ED ESEMPI INTERATTIVI**
- **COMPITI UNITARI**
- **CONCETTI DI BASE**
- **DETTAGLI TECNICI**
- **VERIFICA**

1T
INTRODUZIONE ALLA SINTESI DEL SUONO

CONTRATTO FORMATIVO

PREREQUISITI PER IL CAPITOLO

- Conoscenze di base degli strumenti informatici (operazioni base, gestione delle cartelle, scheda audio etc.)
- Conoscenza minima della teoria musicale (semitoni, ottave, tempi etc.)

OBIETTIVI

Conoscenze

- Conoscere i percorsi mediante i quali si realizza la sintesi e l'elaborazione del suono
- Conoscere i parametri principali del suono e le loro caratteristiche
- Conoscere le codifiche dell'altezza e dell'intensità
- Conoscere i rapporti fra gli intervalli musicali nei diversi sistemi di accordatura
- Conoscere i diversi formati dei file audio

Abilità

- Saper individuare all'ascolto mutamenti di frequenza e d'ampiezza e saperne descrivere le caratteristiche
- Saper individuare le varie fasi dell'inviluppo di un suono o di un suono glissato

CONTENUTI

- Sintesi ed elaborazione del suono al computer
- Timbro, altezza e intensità di un suono: (teoria)
- Glissando e inviluppo d'ampiezza: (teoria)
- Rapporti tra frequenze, altezze e codifiche MIDI
- Uso di suoni campionati (cenni)

TEMPI - Cap.1 (teoria e pratica) + Interludio A

Autodidatti

Per 300 ore globali di studio individuale (vol. I, teoria e pratica):

- ca. 100 ore

Corsi

Per un corso globale di 60 ore in classe + 120 di studio individuale (vol. I, teoria e pratica):

- ca. 16 ore frontali + 4 di feedback
- ca. 40 di studio individuale

ATTIVITÀ

- Esempi interattivi

VERIFICHE

- Test a risposte brevi
- Test con ascolto e analisi

SUSSIDI DIDATTICI

- Concetti di base - Glossario - Un po' di storia

1.1 SINTESI ED ELABORAZIONE DEL SUONO

L'introduzione dell'elettronica e, soprattutto, del computer nella musica ha consentito a compositori e musicisti di gestire e manipolare i suoni con una precisione e una libertà impensabili con i soli mezzi acustici.
Grazie all'uso del computer è infatti possibile modellare il suono in ogni modo immaginabile; si dice spesso che mentre il compositore "tradizionale" compone con i suoni, il compositore elettronico compone i suoni, ovvero entra nel suono, nelle sue componenti elementari, creandole e trasformandole a suo piacimento.
La stessa cosa avviene, per fare un parallelo, nella grafica e nell'animazione: grazie al computer è possibile creare immagini e sequenze filmate estremamente realistiche, che sarebbe difficile produrre con altri mezzi. Attualmente quasi tutti gli effetti speciali al cinema sono realizzati con il computer, e sempre più spesso i personaggi virtuali "recitano" al fianco di attori in carne ed ossa.
Il "segreto" di questa flessibilità sta nel passaggio dal mondo analogico (quello degli oggetti concreti) a quello digitale (ovvero dei numeri): il processo di digitalizzazione consiste appunto nel trasformare un'informazione (un testo, un suono, un'immagine) in numeri.[1] Una volta che un'immagine o un suono sono stati convertiti in una sequenza numerica, possono subire qualunque tipo di trasformazione, perché i numeri, grazie a secoli di sviluppo delle tecniche matematiche, possono essere trasformati e manipolati in qualsiasi modo.

Questo testo si concentrerà essenzialmente su due aspetti: la sintesi e l'elaborazione del suono.

- La **sintesi del suono** (sound synthesis) si riferisce alla generazione elettronica di un suono. In pratica si tratta della possibilità di creare un suono sulla base di alcuni parametri scelti in funzione del risultato sonoro che si vuole ottenere.

- L'**elaborazione del suono**, o del segnale, (signal processing) si riferisce ai processi utilizzati per modificare un suono già esistente, ad esempio un suono di una chitarra che abbiamo precedentemente registrato, un suono generato con una particolare tecnica di sintesi, etc.

SINTESI DIGITALE DEL SUONO

Per ottenere qualsiasi tipo di suono utilizzando un linguaggio di programmazione per la sintesi e l'elaborazione del suono, scriveremo nel computer le informazioni sul tipo di "macchina virtuale" che vogliamo costruire (realizzeremo cioè un **algoritmo**[2]) e le operazioni che questa macchina deve compiere.

---

[1] Approfondiremo questo concetto nel corso del capitolo.

[2] Un algoritmo è un procedimento che comporta una serie ordinata di istruzioni elementari: queste istruzioni, eseguite in successione, permettono di risolvere un problema, di ottenere un risultato. Informalmente possiamo dire che anche una ricetta di cucina è un algoritmo; si tratta infatti di una serie di istruzioni che danno come risultato una pietanza. In informatica un algoritmo è una sequenza di istruzioni scritta in un particolare linguaggio di programmazione che permette al computer di svolgere un compito definito.

Una volta scritte queste istruzioni, chiederemo al programma (Pure Data o altri) di eseguirle e di creare un flusso di dati numerici in cui sono rappresentate digitalmente[3] tutte le caratteristiche del suono o dei suoni che abbiamo richiesto. Tra la generazione di questo flusso di dati digitali e l'ascolto del suono avviene un'altra operazione fondamentale che richiede una **scheda audio**. La scheda legge i dati digitali e li trasforma in segnale elettrico che viene inviato all'amplificatore e poi agli altoparlanti. In questo caso la scheda opera una conversione da digitale ad analogico (D/A), cioè ci consente di ascoltare dei suoni le cui caratteristiche sono scritte in un flusso di dati digitali (fig. 1.1).

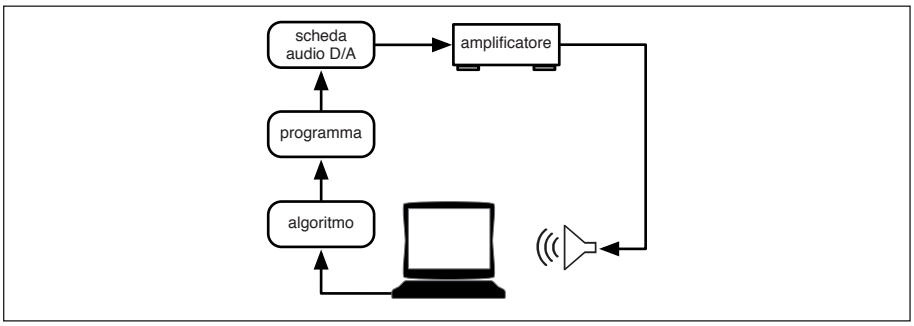


fig. 1.1: sintesi in tempo reale

Questi dati possono anche essere memorizzati in un file audio che verrà salvato nel nostro hard disk, per permettere una riesecuzione dei dati stessi o una loro elaborazione. Quando il flusso dei dati è direttamente inviato alla scheda audio man mano che viene calcolato si ha una **sintesi in tempo reale** (*real time*). Quando invece il processo di calcolo viene prima svolto per intero (e memorizzato in un file audio) e solo successivamente inviato alla scheda audio per l'ascolto si ha una **sintesi in tempo differito** (*non-real time* o *offline*), vedi fig. 1.2.

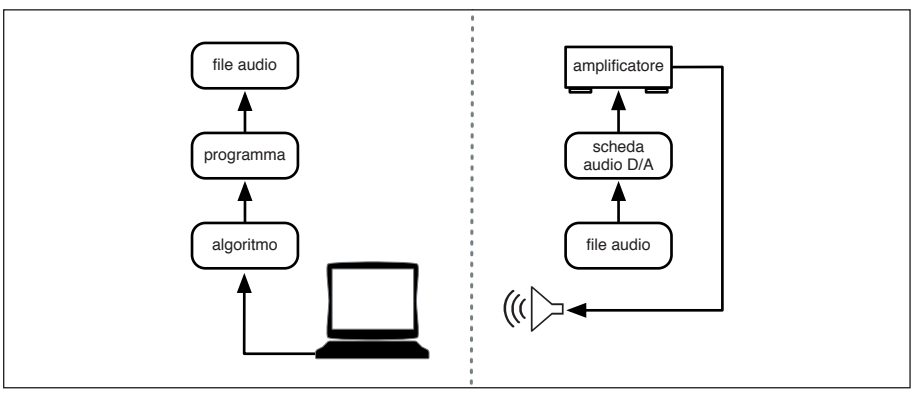


fig. 1.2: sintesi in tempo differito e successivo ascolto

[3] Ovvero sotto forma di numeri.

ELABORAZIONE DEL SUONO

L'elaborazione del suono consiste nella modifica di un suono preesistente, che può provenire sia da una fonte live, sia da un file audio. È possibile operare sia in tempo reale sia in tempo differito, in diversi modi. Vediamo tre possibilità:

1) SUONO PREESISTENTE IN TEMPO DIFFERITO,
ELABORAZIONE IN TEMPO DIFFERITO

Un suono di flauto, ad esempio, può essere registrato (con un microfono collegato alla scheda audio che opererà una conversione analogico-digitale[4]) su un file audio. Possiamo creare un algoritmo in cui specificheremo come quel file audio deve essere modificato, poi il programma eseguirà quei comandi e creerà un nuovo file audio che conterrà un suono di flauto, elaborato dal computer secondo le nostre indicazioni. Infine potremo ascoltare questo nuovo sound file operando una conversione digitale-analogica (fig. 1.3).

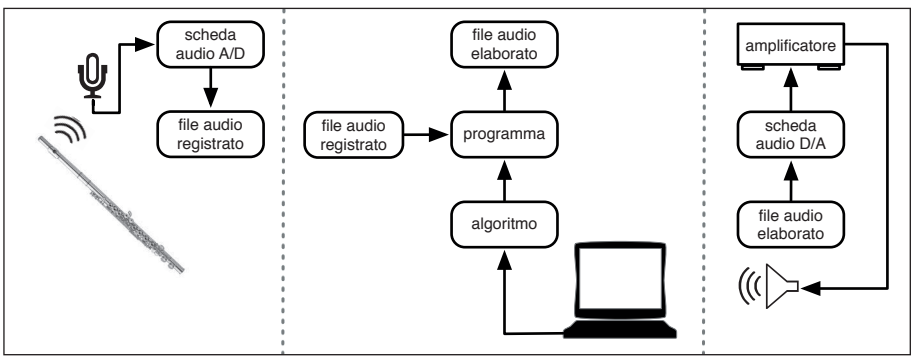


fig. 1.3: esempio di elaborazione in tempo differito

2) SUONO PREESISTENTE IN TEMPO DIFFERITO,
ELABORAZIONE IN TEMPO REALE

Il suono, come nell'esempio 1, proviene da un file audio. Il programma di elaborazione, eseguiti i comandi, invia il flusso di dati contenenti il suono elaborato direttamente alla scheda audio per l'ascolto in tempo reale. Oltre a ciò il programma può registrare, sempre in tempo reale, il risultato dell'elaborazione su un file audio (fig. 1.4).

3) SUONO IN TEMPO REALE,
ELABORAZIONE IN TEMPO REALE

Il suono proviene da una fonte live. Come nell'esempio precedente, il programma di elaborazione, eseguiti i comandi, invia il flusso di dati contenenti il suono elaborato direttamente alla scheda audio.

[4] Ovvero trasformerà un suono reale in una sequenza di numeri.

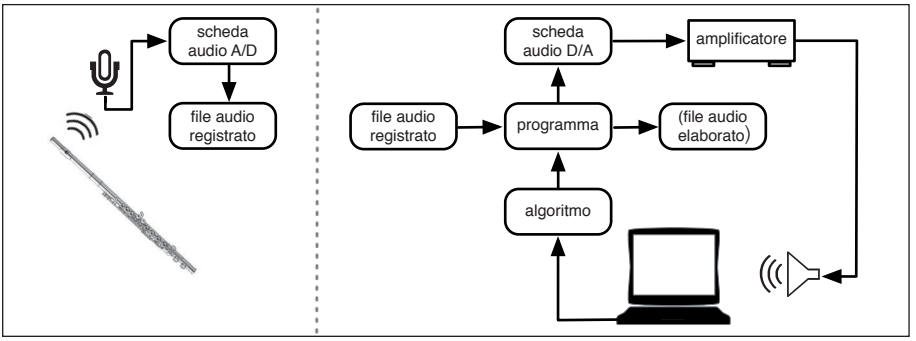


fig. 1.4: esempio di elaborazione in tempo reale da suono preesistente

Naturalmente, anche in questo caso il programma può registrare, sempre in tempo reale, il risultato dell'elaborazione su un file audio (vedi fig. 1.5).

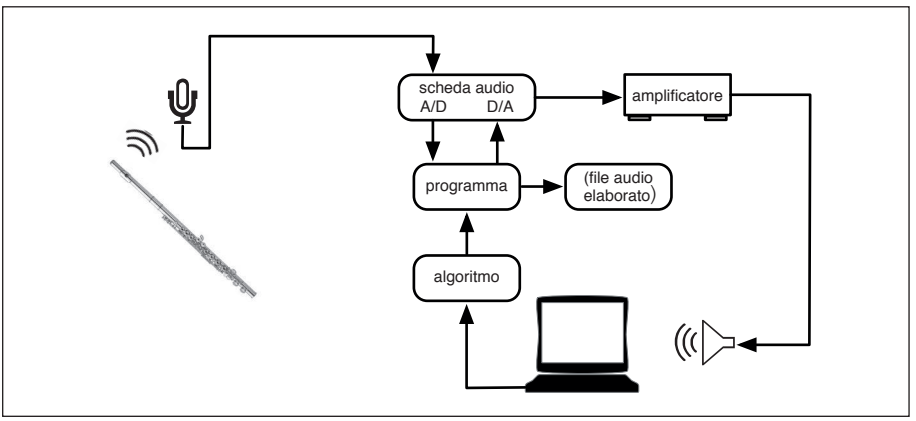


fig. 1.5: esempio di elaborazione in real-time

Definiamo **sistema DSP** l'insieme delle risorse hardware e software (scheda audio, linguaggio di programmazione etc.) che ci permette di elaborare e/o sintetizzare digitalmente un suono (o segnale). Il termine **DSP** è un acronimo che sta per Digital Signal Processing (Elaborazione Digitale del Segnale).

TEMPO REALE - TEMPO DIFFERITO

Abbiamo visto che sia la sintesi sia l'elaborazione del suono possono avvenire in tempo reale o in tempo differito. A prima vista il metodo più vantaggioso appare senz'altro il tempo reale, che ci fornisce un feedback istantaneo e ci consente di valutare immediatamente l'efficacia dell'algoritmo che si sta mettendo a punto e a cui si possono fare le opportune modifiche e migliorie.
A cosa serve quindi il tempo differito?
- Innanzitutto a realizzare degli algoritmi che il computer non è in grado di eseguire in tempo reale: se ad esempio per sintetizzare o elaborare un suono che dura 1 minuto il computer impiega 2 minuti, si dovrà per forza registrare il risultato su disco per poterlo ascoltare una volta che il processo di sintesi o elaborazione sia finito.

Agli albori della computer music tutti i processi di sintesi ed elaborazione del suono erano realizzati in tempo differito, perché un calcolatore non aveva abbastanza potenza per il tempo reale. Con l'aumentare della potenza dei calcolatori, è diventato possibile realizzare alcuni processi direttamente in tempo reale, e nel corso degli anni le capacità di un personal computer di realizzare algoritmi di sintesi ed elaborazione in tempo reale sono aumentate enormemente. Ma naturalmente, per quanto potenti possano diventare i calcolatori, sarà sempre possibile immaginare un processo talmente complesso da richiedere il tempo differito.
- Esiste poi una seconda categoria di processi che sono concettualmente in tempo differito, indipendentemente dalla potenza di calcolo dell'elaboratore: poniamo ad esempio di voler realizzare un algoritmo che, data una sequenza musicale suonata da uno strumento, scomponga tale sequenza in singole note e poi le riordini dalla più grave alla più acuta.
Per realizzare questo algoritmo abbiamo bisogno della sequenza completa, molto probabilmente registrata in un file audio, in modo che il computer la possa analizzare nel suo complesso e individuare la nota più grave e via via le note successive.
Questa analisi può ovviamente avvenire solo in tempo differito, dopo l'esecuzione: l'unico computer che potrebbe realizzare questo algoritmo in tempo reale (cioè mentre lo strumento sta suonando) è un computer in grado di prevedere il futuro!
- Un altro motivo per cui si ricorre al tempo differito è per risparmiare tempo. Contrariamente a quello che si può pensare, il tempo reale non corrisponde alla massima velocità di elaborazione possibile. Immaginiamo ad esempio di dover modificare, con una particolare tecnica di elaborazione, un file di suono della durata di 10 minuti: se la modifica avviene in tempo reale impiegherà ovviamente 10 minuti. Immaginiamo però che il nostro computer sia abbastanza potente da poter realizzare questa elaborazione, in tempo differito, in un minuto. Questo significa che il computer può eseguire i calcoli, per quella particolare tecnica di elaborazione, ad una velocità 10 volte superiore al tempo reale, ed è quindi conveniente ricorrere al tempo differito.

1.2 FREQUENZA, AMPIEZZA E FORMA D'ONDA

Frequenza, ampiezza e forma d'onda sono tre parametri fondamentali del suono. Ognuno di questi parametri influenza nell'ascoltatore la percezione sonora, in particolare:

a) la possibilità di distinguere un suono grave da uno acuto (frequenza)
b) la possibilità di distinguere un suono di forte intensità da uno di intensità minore (ampiezza)
c) la possibilità di distinguere diversi timbri (forma d'onda)[5]

[5] Vedremo più avanti come il parametro del timbro dipenda in realtà da diversi fattori concomitanti.

Vediamo una tabella (tratta da Bianchini, R., 2003) delle corrispondenze fra caratteristiche fisiche del suono, parametri musicali e sensazione sonora.

CARATTERISTICA	PARAMETRO MUSICALE	SENSAZIONE
Frequenza	Altezza	Acuto ↔ Grave
Ampiezza	Intensità	Forte ↔ Piano
Forma d'onda	Timbro	(Chiaro ↔ Scuro Armonico ↔ Inarmonico etc.)

TABELLA A : corrispondenza fra caratteristiche del suono, parametri musicali e sensazione sonora

FREQUENZA

La **frequenza** è il parametro fisico che determina l'altezza di un suono, cioè la caratteristica che consente di distinguere un suono acuto da un suono grave. La gamma delle frequenze udibili dall'uomo si estende da circa 20 a circa 20000 Hertz, cioè da 20 a 20000 cicli al secondo (spiegheremo tra un momento di cosa si tratta): al di sotto della minima frequenza percepibile, sotto i 20 cicli al secondo, si hanno gli infrasuoni, al di sopra di quella massima, sopra i 20000 cicli al secondo, si hanno gli ultrasuoni.[6] Se ci concentriamo sul campo delle frequenze udibili, quindi dei suoni, potremo affermare che maggiore è la frequenza, tanto più acuto sarà il suono.
Ma cosa intendiamo per Hertz o "cicli al secondo"? Per saperlo facciamo riferimento alla definizione di suono data da Riccardo Bianchini:

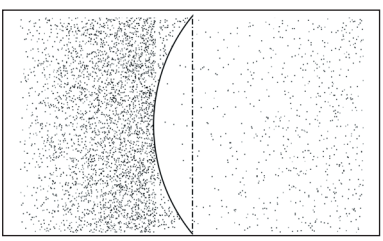


fig. 1.6: vibrazione di una corda

"Per suono si intende quel fenomeno meccanico dato da una perturbazione di un mezzo di trasmissione (in genere l'aria) che abbia caratteristiche tali da essere percepito dall'orecchio umano.[7] La vibrazione viene trasmessa all'aria, per esempio da una corda vibrante (vedi fig. 1.6). La corda si sposta avanti e indietro, e durante questo spostamento comprime le particelle

[6] In realtà la massima frequenza udibile diminuisce con l'età.

[7] Ci sono molte teorie sulla natura del suono: Roberto Casati e Jérôme Dokic sostengono che l'aria è un mezzo attraverso cui il suono si trasmette, ma che il suono in sé è un evento localizzato nel corpo risonante, ovvero nel sistema meccanico che produce la vibrazione. (Casati, R., Dokic, J. 1994). Un altro punto di vista è quello espresso da Frova: "con il termine «suono» si dovrebbe intendere la sensazione, com'essa si manifesta a livello cerebrale, di una perturbazione di natura meccanica, a carattere oscillatorio, che interessa il mezzo interposto tra sorgente e ascoltatore" (Frova, A., 1999, pag.4).

d'aria (molecole) da un lato e le espande dall'altro. Successivamente il moto si inverte, e le molecole che prima erano state compresse si espandono e viceversa. Le compressioni e le espansioni (cioè le perturbazioni dell'aria che inizialmente era in stato di quiete) si propagano poi con una certa velocità attraverso l'aria circostante in tutte le direzioni, dando luogo a onde sferiche. Inizialmente la densità delle molecole d'aria è costante, cioè in ogni unità di volume (per esempio in un cm^3) vi è lo stesso numero di molecole.

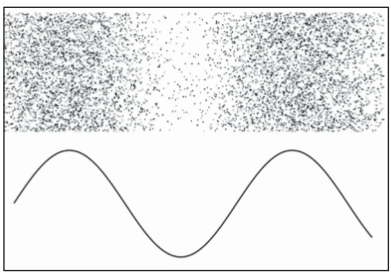


fig.1.7: compressione e rarefazione delle molecole dell'aria

Questa densità può essere espressa da un valore di pressione. Quando l'aria viene perturbata, il valore di pressione non è più costante, ma varia da punto a punto: aumenta dove le molecole sono compresse, diminuisce dove le molecole sono espanse (vedi fig. 1.7).

Il fenomeno può essere studiato sia dal punto di vista dello spazio (osservando il valore della pressione nei vari punti in un determinato istante) sia dal punto di vista del tempo (misurando il valore della pressione in uno stesso punto in funzione del tempo). Ad esempio, se immaginiamo di trovarci in un determinato punto, assisteremo a una successione di compressioni ed espansioni dell'aria (fig. 1.8).

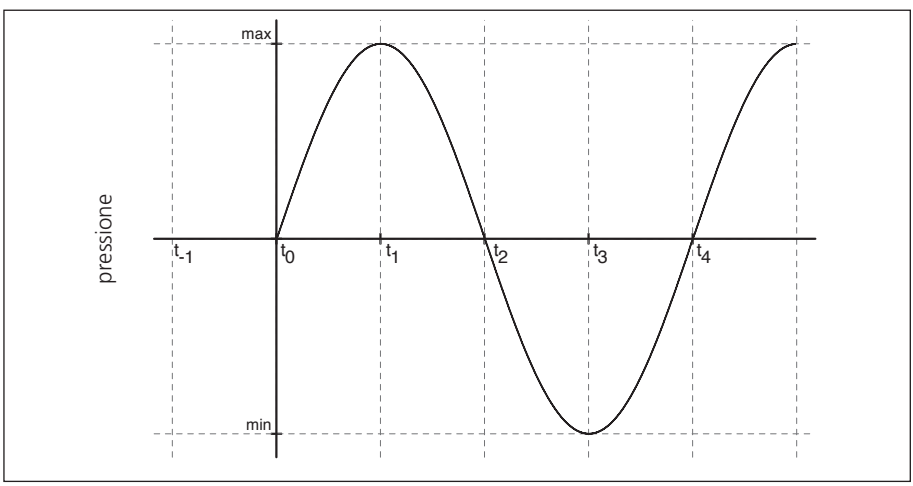


fig.1.8: rappresentazione grafica di compressione e rarefazione

All'istante t_{-1}, ovvero prima dell'istante t_0 la pressione dell'aria è al suo valore normale, dato che la perturbazione non è ancora giunta al nostro punto di osservazione. All'istante t_0 la perturbazione giunge al nostro punto di osservazione, la pressione inizia a crescere, giunge al massimo all'istante t_1, poi decresce fino a tornare al valore normale all'istante t_2, continua a decrescere e giunge al minimo all'istante t_3, per poi risalire fino al valore normale all'istante t_4, e così via.

Si è fin qui descritto un **ciclo** del fenomeno. Se questo si ripete sempre allo stesso modo il fenomeno si dice periodico.[8] Il tempo necessario al completamento di un ciclo si dice **periodo**, si indica con il simbolo T e si misura in secondi (s) o in millisecondi (ms). L'inverso del periodo, cioè il numero di cicli che vengono completati in un secondo, si dice frequenza, e si misura in Hertz (Hz) o cicli per secondo (cps). Se per esempio un'onda sonora ha periodo T=0.01 s (cioè 1/100 di secondo) la sua frequenza sarà di: 1/T = 1/0.01 = 100 Hz (o 100 cicli al secondo)" (ibidem).

Osservando la figura 1.9 ascoltiamo i suoni dell'esempio interattivo[9] numero 1A: possiamo constatare come, all'aumento del numero dei cicli al secondo (Hz), corrispondano suoni sempre più acuti.

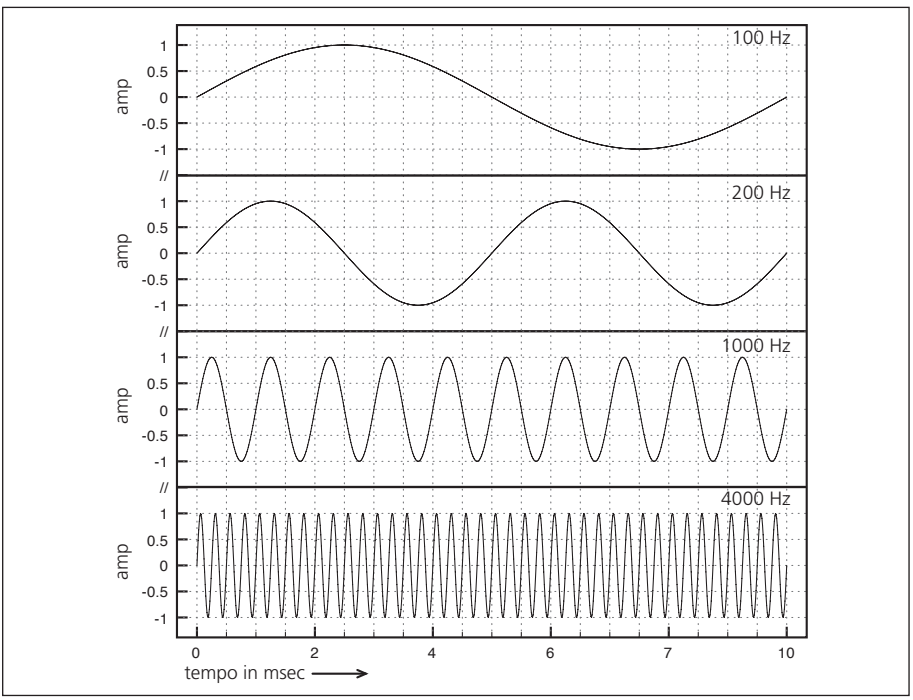


fig.1.9: quattro suoni di frequenza diversa

ESEMPIO INTERATTIVO 1A • FREQUENZA

[8] Matematicamente una forma d'onda si dice periodica se si ripete regolarmente e per un tempo infinito: nella pratica musicale, naturalmente, ci si "accontenta" di durate molto inferiori! In genere un'onda è "musicalmente periodica" quando, ripetendosi con regolarità, persiste per un tempo sufficiente a generare la sensazione di altezza corrispondente al periodo dell'onda. Approfondiremo la questione nel capitolo 2.

[9] Vi ricordiamo che gli esempi interattivi e gli altri materiali di supporto al libro si trovano all'indirizzo http://www.**********.

Dal momento che si propaga nello spazio, un'onda ha una lunghezza che è inversamente proporzionale alla sua frequenza. Chiariamo questo concetto: la velocità del suono nell'aria (cioè la velocità con cui si propagano le onde sonore a partire dalla sorgente) è di circa 344 metri al secondo.[10] Questo significa che un'ipotetica onda di 1 Hz avrebbe una lunghezza di circa 344 metri, perché quando ha completato un ciclo è passato un secondo e in questo tempo si è dispiegata nello spazio per una lunghezza di 344 metri. Un'onda di 10 Hz, invece, in un secondo compie 10 cicli, che si dispongono nello spazio di 344 metri occupando ciascuno 34.4 metri, cioè un decimo dello spazio totale.
Per lo stesso ragionamento un'onda di 100 Hz misura 3.44 metri: come si vede all'aumentare della frequenza diminuisce la lunghezza, e le due grandezze sono quindi, come abbiamo già detto, inversamente proporzionali.

AMPIEZZA

Il secondo parametro fondamentale del suono è l'**ampiezza**, che dà informazioni sulla variazione della pressione sonora, e che permette di distinguere un suono di forte intensità da uno di intensità debole.
La pressione sonora più debole che l'orecchio umano è in grado di percepire si dice **soglia inferiore di udibilità**, mentre la pressione sonora massima che un ascoltatore umano può sopportare si dice **soglia del dolore**, in quanto al di là di questa si ha una vera e propria sensazione di dolore fisico e danni permanenti all'organo dell'udito.
Osservando il fenomeno rappresentato in fig. 1.10, il valore massimo della pressione si dice **ampiezza di picco** dell'onda sonora; il valore della pressione in un punto qualsiasi si dice invece **ampiezza istantanea**.
Quando si indica l'ampiezza di un suono, ci si riferisce al valore dell'ampiezza di picco del suono stesso (vedi fig. 1.10).

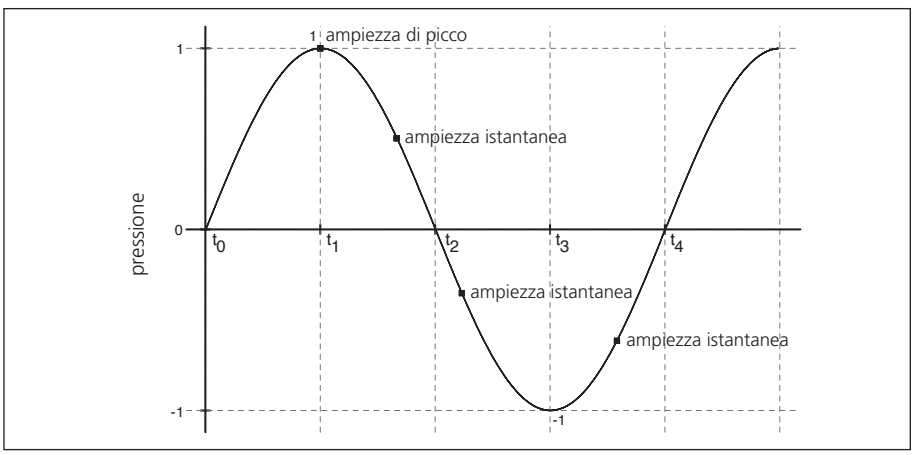


fig.1.10: ampiezza di un suono

[10] Per la precisione questa velocità viene raggiunta quando la temperatura è di 21°. La velocità del suono, infatti, è proporzionale alla temperatura dell'aria.

Ad esempio, se indichiamo un'ampiezza di picco 1, avremo un'onda che parte da un'ampiezza istantanea 0 (all'istante t_0); poi l'ampiezza istantanea inizia a crescere, giunge al massimo all'istante t_1 (valore 1) poi decresce fino a tornare al valore 0 all'istante t_2, continua a decrescere e giunge al minimo all'istante t_3 (-1) per poi risalire fino al valore 0 all'istante t_4, e cosí via. Questa è la rappresentazione dell'ampiezza di un'onda sonora in funzione del tempo. Il processo di digitalizzazione trasforma tale ampiezza in una serie di numeri compresi tra 1 e -1.
I numeri così ottenuti possono essere usati per rappresentare graficamente la forma dell'onda (fig. 1.11). La posizione in cui si trova il ciclo di un'onda in un determinato istante viene chiamata **fase**.
Approfondiremo il concetto di fase nel par. 2.1.

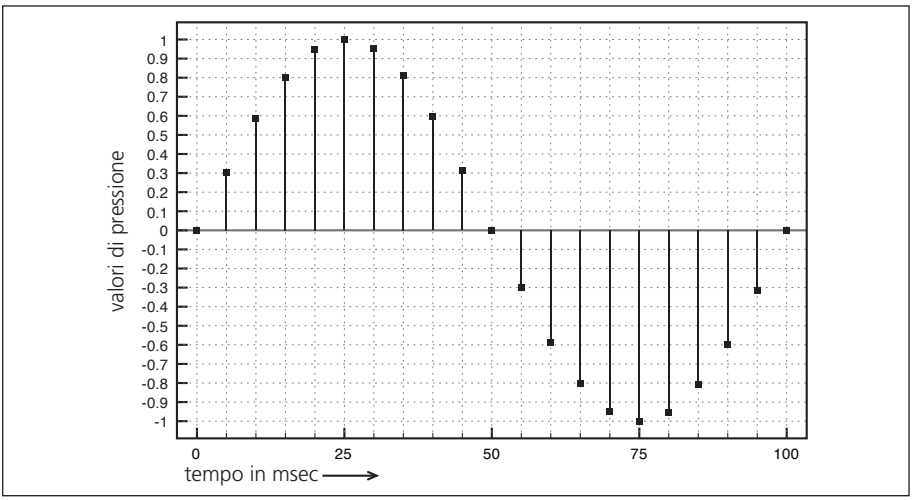


fig. 1.11: rappresentazione digitale di una forma d'onda.

Facendo un confronto con l'onda reale (cioè con la successione di compressioni ed espansioni delle molecole dell'aria), possiamo vedere che la compressione corrisponde ai numeri positivi e l'espansione ai numeri negativi, mentre il valore 0 indica una pressione non perturbata (l'assenza di segnale corrisponde digitalmente a una sequenza di zeri). I valori di ampiezza sono convenzionalmente espressi in numeri con la virgola e variano tra 0 e 1: se indichiamo 1 (cioè il valore massimo) come valore d'ampiezza di picco, avremo oscillazioni fra 1 e -1 (come nell'esempio citato); se impostiamo 0.5 come valore d'ampiezza di picco (cioè metà dell'ampiezza massima), avremo oscillazioni fra 0.5 e -0.5 etc. (vedi fig. 1.12).

(...)

Il capitolo prosegue con:

Forma d'onda
La sinusoide
Altre forme d'onda
Onde bipolari e unipolari
Uso dei logaritmi nel calcolo dei decibel

1.3 VARIAZIONI DI FREQUENZA E AMPIEZZA NEL TEMPO: INVILUPPI E GLISSANDI
Inviluppi di strumenti acustici
Inviluppi di suoni sintetici
Glissandi
Curve esponenziali e logaritmiche

1.4 RAPPORTO TRA FREQUENZA E INTERVALLO MUSICALE

1.5 CENNI SULLA GESTIONE DEI SUONI CAMPIONATI
La digitalizzazione del suono

1.6 CENNI SUL PANNING

ATTIVITÀ
• Esempi interattivi

VERIFICHE
• Test a risposte brevi
• Test con ascolto e analisi

SUSSIDI DIDATTICI
• Concetti di base - Glossario - Un po' di storia

1P
SINTESI DEL SUONO CON PURE DATA

CONTRATTO FORMATIVO

PREREQUISITI PER IL CAPITOLO

- Conoscenze di base degli strumenti informatici (operazioni base, gestione delle cartelle, scheda audio, etc.)
- Conoscenza minima della teoria musicale (toni, semitoni, ottave, tempi etc.)
- Contenuti del Cap. 1 della parte di teoria (si consiglia di studiare un capitolo per volta, affrontando prima la teoria e poi la pratica con Pd)

OBIETTIVI

Abilità

- Saper utilizzare tutte le funzioni di base del software Pd
- Saper sintetizzare suoni in sequenza e in sovrapposizione utilizzando oscillatori sinusoidali, ad onda quadra, triangolare o dente di sega
- Saper controllare in modo continuo l'ampiezza, la frequenza e la spazializzazione stereofonica di un suono (uso di spezzate di retta e di esponenziale per glissandi, inviluppi d'ampiezza e movimento del suono nello spazio stereo)
- Saper generare sequenze casuali di suoni sintetizzati
- Saper gestire l'utilizzo elementare dei suoni campionati

Competenze

- Saper realizzare un primo studio sonoro di due minuti basato sulle tecniche acquisite e memorizzarlo su file audio

CONTENUTI

- Sintesi ed elaborazione del suono al computer
- Timbro, altezza e intensità di un suono
- Glissando e inviluppo d'ampiezza
- Rapporti tra frequenze, altezze e codifiche MIDI
- Uso di suoni campionati (cenni)

TEMPI - Cap. 1 (Teoria e Pratica) + Interludio A

Autodidatti

Per 300 ore globali di studio individuale (vol. I, teoria e pratica): ca. 100 ore

Corsi

Per un corso globale di 60 ore in classe + 120 di studio individuale (vol. I, teoria e pratica): ca. 16 ore frontali + 4 di feedback - ca. 40 di studio individuale

ATTIVITÀ

- sostituzione di parti di algoritmi, correzione, completamento e analisi di algoritmi, costruzione di nuovi algoritmi

VERIFICHE

- Compiti unitari di Reverse Engineering

SUSSIDI DIDATTICI

- Lista comandi principali - Lista oggetti nativi di Pd - Lista oggetti della libreria virtual sound - Lista messaggi per oggetti specifici - Glossario

1.1 INSTALLAZIONE E PRIMI PASSI CON PD

Installazione e configurazione del sistema

Prima di procedere alla lettura di questo capitolo è necessario svolgere le seguenti operazioni:

1. scaricare e installare il software Pure Data (Pd);
2. scaricare e installare la libreria *Virtual Sound*;
3. scaricare il materiale di supporto a corredo di questo testo;
4. configurare il proprio sistema audio e MIDI.

Il software Pure Data[1] può essere scaricato dal sito http://msp.ucsd.edu/software.html, dove si trovano i file per i sistemi operativi Windows e Mac OSX (anche a 64 bit). Per installare il software basta eseguire il file scaricato e seguire le istruzioni che compaiono. Su sistemi Linux le cose sono leggermente più complicate, ma solitamente Pd può essere installato tramite l'applicazione *synaptic* senza particolari problemi. Per una trattazione più approfondita in merito all'installazione sui diversi sistemi operativi vi invitiamo a leggere il documento relativo (**MESD_installazione_Pd.pdf**) che trovate nel sito http://www.*****

Questo testo fa riferimento a Pd-0.47-1 (l'ultima versione stabile in circolazione al momento della pubblicazione di questo libro), ma gli esempi dovrebbero funzionare anche con le precedenti versioni, fino alla 0.45 inclusa. Nel caso di alcune installazioni in sistemi Linux, potrebbero non essere disponibili le versioni più recenti del software, quindi bisogna "adeguare" le *patch* affinché funzionino su versioni più vecchie dell'applicazione. Un pdf allegato (**MESD_Pd_old_versions.pdf**), nel sito sopra citato, spiega come fare.

Sullo stesso sito dovete scaricare la libreria *Virtual Sound*, che è assolutamente necessaria per garantire il funzionamento degli esempi del testo. Per installarla correttamente osservate la seguente procedura:

1. Scaricate il file **VirtualSoundLibraryPd.zip** sul vostro computer e scompattatelo: avrete così la cartella *Virtual_Sound_Library_Pd*;
2. Aprite Pd e dal menu *Pd/Preferences* (Mac) o *File/Preferences* (Altri sistemi) aprite la sezione *Path*. Fate clic su *New*: si aprirà una finestra in cui dovete cercare e selezionare la cartella *Virtual_Sound_Library_Pd*. Una volta trovata e selezionata fate clic su *Ok*;
3. Riavviate *Pd*.

[1] Cercando su Internet potreste imbattervi in versioni diverse di Pure Data. In particolare è molto diffusa la versione *Pd-extended*, un'edizione molto ricca di librerie esterne che purtroppo non è più mantenuta e per tale ragione si basa su una versione di Pure Data superata. Questo testo è basato su Pure Data, edizione *vanilla*, che rappresenta la versione originale del programma ed è mantenuta dal suo creatore Miller Puckette. Gli esempi del libro NON funzionano tutti su *Pd-extended.*

Se la procedura è andata a buon fine *Pd* cercherà gli oggetti della libreria *Virtual Sound* in automatico, ogni volta che un oggetto viene richiamato dall'utente[2]. Per maggiori informazioni consultate i documenti relativi presenti nel sito.

Il materiale di supporto è raccolto nel file **MESD_PD_MATERIALE_vol_1.zip**, che potete scaricare e scompattare in una posizione arbitraria del vostro computer. Da lì potete aprire i file a corredo dei capitoli, inclusi quelli che riguardano le attività e i compiti di *reverse engeneering*.

Configurazione dell'audio

Per verificare che l'audio funzioni osservate la seguente procedura:

1. Dal menu *Media/Audio Settings* accertatevi che i campi *Input Device 1* e *Output Device 1* contengano il nome della vostra scheda audio, se così non fosse, scegliete la vostra scheda dal menu a tendina relativo (in entrambi i campi);
2. Dal menu *Media/Test Audio and MIDI* aprite la finestra relativa e, nella sezione a sinistra, etichettata con 'TEST TONES', mettete la spunta su '80' e su 'noise' (fig. 1.1). Accertatevi che il computer non sia in *mute*!

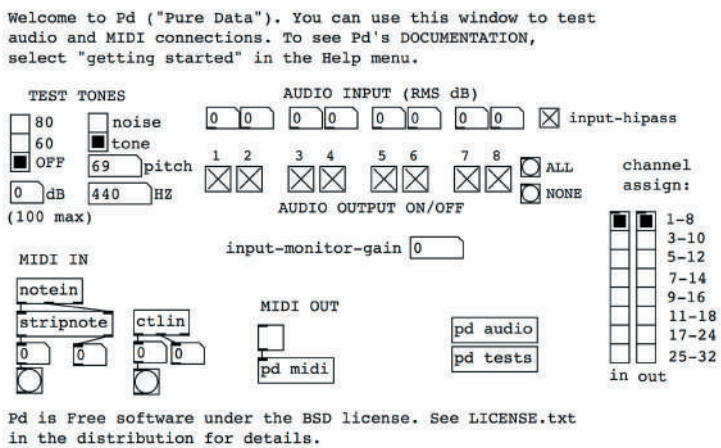


fig. 1.1: la finestra per verificare il funzionamento dell'audio all'interno di Pd.

Per eventuali problemi consultate il documento **MESD_audio_config.pdf** che si trova all'indirizzo http://www.**********.
Per la configurazione del protocollo MIDI, rimandiamo il lettore al capitolo IB.1 ("Cenni sul MIDI") o, eventualmente, al pdf allegato (**MESD_MIDI_config.pdf**).

Ora possiamo cominciare davvero!

[2] Tutti i nomi degli oggetti della libreria *Virtual Sound* iniziano con il prefisso *vs.* seguito dal nome dell'oggetto.

Primi passi con Pd

Lanciamo il programma Pd e selezioniamo dal menù File la voce *New* (oppure digitiamo *<Mac: Command–n> <Win e Linux: Control–n>*)[3]: apparirà una finestra, la **Patcher Window**, detta **Canvas** (*tela*), nella quale possiamo cominciare ad assemblare il nostro primo algoritmo. Un algoritmo si configura come un insieme di "scatole", dette genericamente **text box**, che contengono stringhe di testo e numeri. Un insieme di *text box* collegati fra loro si chiama **patch**, in analogia con i vecchi moduli dei sintetizzatori analogici che venivano programmati con connessioni fisiche effettuate tramite cavi chiamati **patch cords**).

Dopo aver aperto una *patcher window* premete la combinazione di tasti *<C-1>*, o in alternativa aprite il menu *Put* e selezionate la voce *Object*: comparirà un rettangolo con i lati tratteggiati nel quale è possibile scrivere delle sequenze di caratteri (fig. 1.2) grazie alla presenza del cursore lampeggiante.

fig. 1.2: *object box* generico

Questo rettangolo è detto **object box**. Scrivete all'interno la stringa di caratteri 'osc~ 440'[4] e fate clic col mouse all'esterno del rettangolo (fig. 1.3). In questo modo avete creato l'oggetto `[osc~]`, cioè l'oscillatore sinusoidale. In Pd una sequenza di caratteri priva di spazi è detta **atomo** (*atom*) e può essere costituita da un numero, da una stringa letterale o un qualunque simbolo. Un oggetto in Pd è costituito da un atomo iniziale che definisce il nome dell'oggetto e da uno o più atomi che costituiscono gli eventuali argomenti di inizializzazione (**creation arguments**). Con l'operazione appena compiuta abbiamo creato un'oggetto della classe `[osc~]` con un *creation argument* pari a 440.

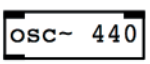


fig. 1.3: l'oggetto `[osc~]`

I rettangolini nella parte alta e bassa dell'oggetto sono rispettivamente gli ingressi (**inlet**) e l'uscita (**outlet**), e vedremo tra poco come si utilizzano. (NB: Se l'oggetto non dovesse avere questo aspetto vuol dire che c'è un problema, leggetevi le FAQ alla fine di questo paragrafo).

[3] Su Mac OS X teniamo premuto il tasto *Command* e digitiamo "n"; su Windows e Linux teniamo premuto il tasto *Control* e digitiamo "n". Da ora in poi i comandi *Command* e *Control* saranno abbreviati in "C". Ad esempio per indicare l'apertura di una nuova *Canvas* il comando sarà *<C-n>*.
[4] Il simbolo "~" (tilde), che segue la stringa "osc", si ottiene usando diverse combinazioni di tasti. Su MAC OS X: <ALT-5>; su Linux: <ALT GR-ì>; su Windows: <ALT-126>; la sequenza "126" va digitata sul tastierino numerico mentre ALT resta premuto. In mancanza del tastierino, come nei portatili, usare la combinazione <ALT-Fn-126>.

fig. 1.4: l'oggetto `[vs.gain~]`

Ora creiamo un altro oggetto, `[vs.gain~]`, che fa parte della libreria *Virtual Sound* e che ha l'aspetto del *fader* di un mixer (vedi fig. 1.4). Come per `[osc~]`, premete <*C-1*> per creare un *object box* generico e al suo interno scrivete la stringa 'vs.gain~', dopo di che fate clic su un punto vuoto della *Patcher Window*[5]. Questo oggetto grafico fa parte della categoria GUI (*graphical user interface*) e permette l'interazione diretta tramite il mouse. Spostate questo oggetto sotto `[osc~]`, e collegate l'uscita di `[osc~]` con l'ingresso di `[vs.gain~]`. Per creare un collegamento è necessario usare dei cavi, detti **patch cords**.

fig. 1.5: collegare gli oggetti

Per questa operazione si usa il mouse seguendo la sequenza di azioni *premi-trascina-rilascia* (*click, drag and drop*); portate il puntatore (che ha la forma di una piccola mano) sull'*outlet* di `[osc~]`; appena il puntatore diventa un circoletto, fate clic col mouse, trascinate il puntatore sopra l'*inlet* di `[vs.gain~]` e appena diventa nuovamente un circoletto rilasciate il pulsante del mouse. Se non sono stati fatti errori il risultato dovrebbe essere simile a quello della figura 1.5.

fig. 1.6: l'oggetto `[dac~]`

Abbiamo collegato l'oscillatore all'oggetto `[vs.gain~]` che serve a regolarne il volume. Adesso dobbiamo inviare il segnale audio all'uscita: creiamo l'oggetto `[dac~]` (fig. 1.6) e spostiamolo sotto a `[vs.gain~]`. Colleghiamo l'uscita di `[vs.gain~]` prima all'*inlet* sinistro di `[dac~]`, poi, ripetendo le operazioni di trascinamento dei *patch cords*, all'*inlet* destro (fig.1.7).

fig. 1.7: la nostra prima *patch*

[5] Se l'oggetto non appare, forse non avete scritto correttamente il nome, oppure non avete installato la libreria *Virtual Sound:* in quest'ultimo caso tornate all'inizio del capitolo e seguite le istruzioni per l'installazione.

Attenzione: verificate di aver usato l'uscita sinistra di `[vs.gain~]` per entrambi i collegamenti. Se vi accorgete che uno dei due cavi esce dell'*outlet* destro di `[vs.gain~]` potete cancellarlo in questo modo: selezionatelo con un clic (il cavo diventerà blu) e premete il tasto di cancellazione (quello che usate quando dovete cancellare del testo); a questo punto ricollegate gli oggetti nel modo corretto.

Probabilmente ora vorrete salvare la *patch* su disco; fatelo pure, ma con un'avvertenza: NON date alla *patch* lo stesso nome di un oggetto Pd! Ad esempio, non chiamate questa *patch* "osc~": è il modo migliore per confondere Pd e avere risultati inaspettati la prossima volta che ricaricherete la *patch*. Dal momento che è impossibile ricordare tutti i nomi degli oggetti di Pd (per evitare di usarli come nome di *patch*), un buon modo per scongiurare il "pericolo" è dare al file un nome composto da più parole, ad esempio "test oscillatore", oppure "test oggetto osc~", o quello che preferite: nessun oggetto di Pd ha un nome composto da più parole. Non trascurate questo consiglio, molti dei malfunzionamenti riscontrati dagli utenti di Pd alle prime armi derivano proprio dal fatto che prima o poi creano un file con lo stesso nome di un oggetto.

Bene, abbiamo realizzato la nostra prima *patch* e siamo pronti per farla funzionare. Manca però ancora un passaggio: finora abbiamo lavorato in **Edit Mode**, cioè la modalità che ci permette di assemblare la *patch* spostando e collegando gli oggetti; ora, per far suonare la nostra *patch* dobbiamo passare in **Run Mode**, premendo la combinazione <*C-e*>, oppure dal menu *Edit* togliendo la spunta alla voce *Edit Mode*.
Adesso con la combinazione <*C-/*> (oppure dal menu *Media/DSP On*) attiviamo il motore audio[6] e alziamo la levetta presente nell'oggetto `[vs.gain~]`[7]: dovremmo udire un suono, per la precisione un La sopra il Do centrale. Con la combinazione <*C-.*> (menu *Media/DSP Off*) possiamo invece "spegnere" la *patch*. Se non avete sentito alcun suono consultate le FAQ alla fine di questo paragrafo.

Analizziamo ora il nostro algoritmo: l'oggetto `[osc~]` è un oscillatore, ovvero un generatore di suono che nel nostro caso genera un'onda sinusoidale, e il numero 440 indica la sua frequenza; questa sinusoide, cioè, si ripete 440 volte al secondo[8]. In altre parole `[osc~]` è il nome dell'oggetto e 440 è il suo **argomento**, vale a dire il valore che l'oggetto in questione utilizza per operare, in questo caso appunto 440 Hz.

[6] In Pd il motore audio non è attivo per impostazione predefinita. Bisogna attivarlo (e disattivarlo) all'occorrenza. Questo perché Pd è un vero e proprio linguaggio di programmazione che, oltre alla sintesi del suono, può svolgere molte operazioni che non richiedono l'uso dell'audio.
[7] Quando si crea un oggetto `[vs.gain~]`, per impostazione predefinita questo avrà la levetta abbassata al minimo.
[8] Tutti questi concetti vengono spiegati nel paragrafo 1.2 della parte di teoria.

Questo oggetto è collegato con l'oggetto `[vs.gain~]` e quindi il segnale che genera viene passato a quest'ultimo, che come abbiamo visto modifica il volume del segnale. Il segnale modificato passa poi a `[dac~]`, la cui funzione è quella di mandare il segnale alla scheda audio del computer. Quest'ultima effettua la conversione digitale-analogica del segnale, cioè trasforma i numeri in segnale audio che possiamo udire attraverso le casse collegate al computer. Il nome "dac" peraltro è un acronimo che sta per *Digital to Analog Converter* (Convertitore Digitale-Analogico).

Cerchiamo di approfondire ulteriormente questa *patch*; oltre a udire il suono, infatti, possiamo "vederlo". Salvate la *patch* che avete appena realizzato in una cartella apposita che potreste chiamare, ad esempio, "le mie patch" (ci servirà nel prossimo paragrafo) e chiudete la *Patcher Window*.

Ora scaricate e scompattate (se non l'avete ancora fatto) il file MESD_PD_MATERIALE_vol_1.zip che si trova all'indirizzo www.********** ***. Poi aprite il file **01_01.pd** (fig. 1.8) che trovate nella cartella "*MESD_PD_MATERIALE/VOL_1/01_patch*".

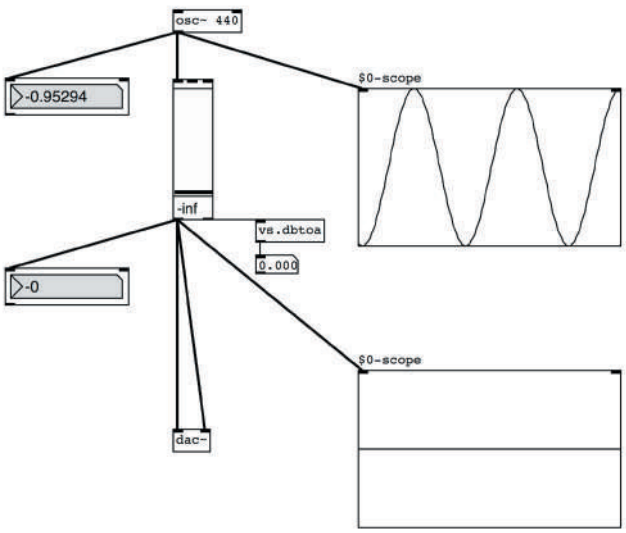


fig. 1.8: file 01_01.pd

Qui abbiamo aggiunto alla *patch* alcuni nuovi oggetti. Gli oggetti sulla sinistra in cui sono visibili dei valori numerici si chiamano `[vs.number~]` e mostrano, sotto forma di numero, il contenuto del segnale che ricevono; gli oggetti quadrati sulla destra si chiamano `[vs.scope~]` e sono degli oscilloscopi che ci fanno vedere il segnale come un'onda che si muove su uno schermo; l'oggetto `[vs.dbtoa]` e l'oggetto collegato (che si chiama *number box*[9]) ci fanno vedere di quanto `[vs.gain~]` amplifica o attenua il segnale che riceve.

[9] Il *number box* si crea con la combinazione <*C-3*> o dal menu *Put/Number*.

Avviamo l'audio con la combinazione <*C-/*> e osserviamo i numeri mostrati dal `[vs.number~]` in alto a sinistra: questi numeri sono prodotti dall'oggetto `[osc~]` e, se li osserviamo per un po', ci renderemo conto che sono valori positivi e negativi compresi fra 1 e -1. Sul lato destro vediamo il `[vs.scope~]` superiore che ci mostra questi stessi numeri sotto forma di grafico: nella metà superiore del riquadro vengono rappresentati i valori positivi, in quella inferiore i negativi. Nel riquadro del `[vs.scope~]` viene mostrato non un singolo numero, ma una sequenza di diverse centinaia di elementi, che vengono visualizzati come punti nel riquadro stesso: questi punti sono molto vicini tra loro e nell'insieme ci appaiono come una linea curva. Questi elementi, questi numeri, nella terminologia della musica digitale si chiamano *campioni*. La linea che oscilla sinuosamente in alto e in basso all'interno dell'oscilloscopio è appunto la forma d'onda sinusoidale prodotta da `[osc~]`.

Se abbassate fino al valore più basso il livello di `[vs.gain~]` vi accorgerete che il `[vs.number~]` e il `[vs.scope~]` in basso mostreranno rispettivamente il numero 0 e una linea piatta. Se alziamo nuovamente il cursore di `[vs.gain~]` vediamo che il `[vs.number~]` ci mostra dei numeri dapprima molto piccoli e poi via via sempre più grandi man mano che aumentiamo il volume. Nello stesso tempo la linea piatta del `[vs.scope~]` in basso comincia a diventare ondulata e ad assomigliare a quella del `[vs.scope~]` in alto: quella che viene modificata è l'ampiezza del segnale; più alziamo il cursore e più l'oscillazione diventa ampia.
Se però alziamo troppo il cursore di `[vs.gain~]` vediamo che i numeri cominciano a superare i limiti di 1 e -1 e che la forma d'onda rappresentata nell'oscilloscopio è diventata troppo ampia e appare tagliata. Inoltre potete osservare che il suono cambia, diventa distorto.

Da tutto ciò possiamo trarre alcune conclusioni:

1. l'oggetto `[osc~]` produce una sequenza di valori digitali che seguono l'andamento di una funzione seno[10];
2. i limiti numerici di questa sinusoide sono 1 e –1. Come si vede nell'immagine che appare nel `[vs.scope~]` superiore, questi sono anche i limiti massimi, superati i quali il suono viene distorto;
3. l'oggetto `[vs.gain~]` modifica l'ampiezza della sinusoide, e fa sì che i campioni in entrata siano diversi dai campioni in uscita. Come fa? Moltiplicando i valori che riceve per una certa quantità che dipende dalla posizione del cursore. Quando il cursore è nella posizione più bassa il segnale viene moltiplicato per 0, e il risultato è una sequenza di zeri, come abbiamo visto, perché qualsiasi numero moltiplicato per 0 dà come risultato 0. Man mano che alziamo il cursore il fattore di moltiplicazione aumenta.

[10] Per la verità, come vedremo successivamente, `[osc~]` segue l'andamento di una funzione coseno.

Se ad esempio lo portiamo a -6, il fattore di moltiplicazione corrispondente sarà circa 0.5 (come si vede dal *number box* collegato all'uscita di `[vs.gain~]`), che significa dimezzare l'ampiezza, perché moltiplicare un numero per 0.5 equivale a dividerlo per 2. Se poi lo alziamo fino a 0[11] (spostando il cursore finché l'indicatore in basso non indichi questo valore) i campioni in entrata non subiscono variazioni in uscita, ma rimangono identici perché moltiplicati per 1 (osservate ancora il *number box* collegato a `[vs.dbtoa]`).

Infine alziamo ulteriormente il cursore (il massimo consentito dall'oggetto è un valore di +6): ora i valori estremi dei campioni superano il limite di 1 e -1, ma questi campioni vengono riportati entro i limiti durante la conversione digitale-analogica: questo fa sì che la forma d'onda non sia più una sinusoide, poiché l'onda appare tagliata (come vediamo nell'oscilloscopio inferiore). In realtà i campioni fuori *range* vengono semplicemente riportati alla massima ampiezza disponibile, e il suono distorto che sentiamo è relativo a questa nuova forma d'onda.

Abbiamo trattato più a fondo i concetti di ampiezza, frequenza e forma d'onda nel par. 1.2 della teoria, riassumiamo alcuni concetti basilari:

- l'*ampiezza* è il parametro fisico da cui dipende l'*intensità* del suono, cioè il parametro che ci fa percepire forte o piano un determinato evento sonoro; i valori assoluti d'ampiezza (cioè indipendenti dal segno) in Pd vanno da un minimo di 0 a un massimo di 1;
- la *frequenza* è il parametro fisico da cui dipende l'*altezza* del suono, cioè il parametro che ci fa percepire un suono come grave o acuto. I valori sono espressi in Hertz (Hz), e quindi dovremo tener conto che i suoni udibili dall'uomo sono fra circa 20 e circa 20000 Hz;
- la *forma d'onda*, che nel caso di `[osc~]` come abbiamo visto è una sinusoide, è un parametro fondamentale che concorre a definire il *timbro* del suono, cioè quella qualità del suono che ci consente di percepire la differenza, ad esempio, fra il Do di una chitarra e quello di un sassofono.

FAQ (Frequently Asked Questions)

FAQ significa "Domande Frequenti" e in questa sezione cercheremo di dare una risposta ad alcuni dei problemi più comuni che si incontrano quando si comincia a lavorare con Pd. Leggetele attentamente anche se non avete incontrato alcun problema: contengono informazioni che vi saranno utili nel seguito della lettura di questo libro.

1) Domanda: Ho creato un oggetto chiamato "osc~440" come c'è scritto in questo capitolo, ma l'oggetto non ha né ingressi né uscite. Perché?

[11] Per ragioni che vi saranno chiare in seguito, `[vs.gain~]` usa la scala dei deciBel (vedi cap. 1.2 della parte teorica), quindi il valore 0 corrisponde all'ampiezza 1, che è quella massima possibile.

Risposta: Controllate di aver messo uno spazio tra "osc~" e "440" perché il primo è il nome dell'oggetto e il secondo è l'argomento, che in questo caso rappresenta la frequenza del suono. Se le due parole sono attaccate Pd cercherà un oggetto inesistente chiamato "osc~440" e non trovandolo non mostrerà un *object box* corretto con ingressi e uscite.

2) D: Va bene. Perché però non mi ha dato un messaggio di errore?

R: Il messaggio d'errore c'è, e si trova nella **Pd window**: una finestra che il programma utilizza per comunicare con l'utente. Potete richiamarla dal menu *Window/Pd window* oppure tramite la combinazione <*C–r*>. Nella finestra, troverete probabilmente questo messaggio:
"osc~440
… couldn't create"

3) D: Io ho messo uno spazio tra "osc~" e "440", però l'oggetto è privo di ingressi e uscite lo stesso!

R: Questo è un errore più sottile, e capita spesso all'inizio con gli oggetti che hanno una tilde (~) alla fine del nome. Probabilmente per scrivere questo carattere avete dovuto usare una combinazione di tasti (ad esempio, per Mac <*alt–5*>), e uno dei tasti della combinazione è rimasto premuto mentre avete digitato lo spazio (avete ad esempio premuto <*alt–spazio*>); la combinazione non è riconosciuta da Pd che quindi non è in grado di separare il nome dell'oggetto dall'argomento. Cancellate lo spazio e riscrivetelo, facendo attenzione a premere solo la barra spaziatrice.

4) D: Non sento alcun suono.

R: Avete attivato l'audio con <*C-/*>? Avete alzato il cursore del *fader* se era abbassato? Siete sicuri che il computer non sia in *mute*, ovvero riuscite a riprodurre dei suoni con altri programmi? Avete controllato che sulla finestra *Audio Settings* (sotto il menù *Media*) sia stata selezionata la scheda audio giusta?

Piccolo "manuale di sopravvivenza" per Pd

In questa sezione daremo alcune informazioni essenziali per muoversi bene nell'ambiente Pd.

COMANDI DA TASTIERA BASILARI
Innanzitutto rivediamo i comandi da tastiera che abbiamo imparato finora:

<*C-n*> serve a creare una nuova *Patcher Window*, lo spazio di lavoro dove possiamo realizzare le *patch*. .

<*C-e*> serve per alternare la modalità *edit* alla modalità *run* nella *Patcher Window*. In *edit* possiamo assemblare le *patch* creando gli oggetti e tutti gli altri

tipi di *text box*; in *run mode* possiamo far funzionare la *patch* ed interagire con gli oggetti grafici di interfaccia, come i *number box* o l'oggetto `[vs.gain~]`.

<C-r> serve per richiamare (qualora non fosse già visibile) la *Pd window* che è una finestra utilizzata dal programma per comunicare con l'utente, e che l'utente può usare per visualizzare brevi messaggi (vedremo più avanti come).

<C-1> serve per creare un *object box* generico, all'interno del quale scriviamo una stringa che rappresenta il nome dell'oggetto specifico, seguito dagli eventuali *creation arguments*. L'*object box* può essere anche creato tramite il menu *Put/Object*.

<C-3> serve per creare un *number box*, che può essere creato anche attraverso il menu *Put/Number*.

SELEZIONARE, CANCELLARE E COPIARE

Per cancellare un cavo o un oggetto bisogna assicurarsi di essere in modalità *edit*[12] e poi selezionarlo con il mouse e premere il tasto di cancellazione, detto anche *Backspace*. Possiamo selezionare più oggetti contemporaneamente facendo clic su un punto vuoto della *Patcher Window* e trascinando il mouse in modo da includere gli oggetti da selezionare nell'area di trascinamento. Gli oggetti selezionati sono colorati in blu; a questo punto se spostiamo uno degli oggetti selezionati spostiamo anche tutti gli altri, oppure se premiamo il tasto di cancellazione li cancelliamo tutti, inclusi i cavi che li collegano.

Per cancellare solo i cavi invece le cose funzionano diversamente: per cancellare un singolo cavo è sufficiente selezionarlo con un clic del mouse e premere il tasto *Backspace*. Cancellare invece un gruppo di cavi è impossibile, perché in Pd non si possono selezionare gruppi di cavi, bisogna selezionarli e cancellarli uno alla volta.

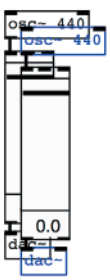


fig. 1.9: duplicare oggetti

Per duplicare un oggetto o un gruppo di oggetti bisogna selezionarlo o selezionarli e premere la combinazione *<C-d>*: gli oggetti duplicati vengono posti molto vicino agli originali (fig. 1.9) e risultano selezionati; in questo modo è possibile spostarli grazie alle frecce della tastiera del computer. Per far sì che lo spostamento sia più rapido si può tenere premuto il tasto SHIFT[13] mentre si usano le frecce: questo meccanismo permette di fare un passo di spostamento più lungo.

[12] Ovvero bisogna accertarsi che muovendo il mouse all'interno della *patcher window* il puntatore abbia la forma di una mano.

[13] SHIFT è il tasto delle maiuscole.

Per copiare un oggetto o un gruppo di oggetti in un'altra *patcher window* è sufficiente selezionare l'oggetto o il gruppo di oggetti poi usare la combinazione <*C-c*> per copiare e <*C-v*> per incollare, come per qualsiasi editor.

Se fate un errore (ad esempio cancellate un oggetto al posto di un altro) potete annullarlo selezionando il comando *undo* dal menù *Edit* (combinazione <*C-z*>) (in italiano il comando si traduce generalmente con *annulla*). Se poi vi accorgete che non era un errore (ad esempio che volevate cancellare proprio quell'oggetto) potrete ripristinare la situazione tramite il comando *redo* (*ripristina*) sempre dal menù *Edit* (<*SHIFT-C-z*>). Attenzione! Pd tiene in memoria solo l'ultima operazione compiuta, quindi il comando *undo* può annullare solo l'ultima operazione. Per *redo* vale lo stesso principio: ripristina solo l'ultima operazione.

LA MODALITÀ AUTOPATCH

Un'interessante modalità di lavoro messa a disposizione da Pd è l'*autopatch mode*, che permette di creare oggetti già collegati fra loro. Vediamo con un esempio pratico come si realizza questa tecnica: aprite una nuova *patcher window* (<*C-n*>) e al suo interno create un oggetto `[osc~ 440]` con <*C-1*>. Selezionate l'oggetto e premete nuovamente <*C-1*>: comparirà un nuovo *object box* generico già collegato ad `[osc~]` tramite un *patch cord* (Fig. 1.10); scrivete `[dac~]` dentro a questo *object box* e poi fate clic in una parte vuota della *patcher window*. In questo modo avete creato due oggetti già collegati. Questo sistema funziona solo nei collegamenti fra gli *outlet* e gli *inlet* di sinistra degli oggetti.

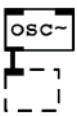


fig. 1.10: creazione di oggetti collegati tramite la modalità *autopatch*

HELP DI PD

Questo libro è autosufficiente: qui troverete tutte le informazioni utili per comprendere e usare le *patch* che via via illustreremo e per utilizzare le diverse tecniche di sintesi ed elaborazione del suono con Pd. Se conoscete l'inglese può essere utile dare anche un'occhiata al sistema di *Help* in linea del programma.
Praticamente ogni oggetto di Pd ha una sua pagina di *help* dedicata. Per accedervi è sufficiente, in *Edit Mode*, fare clic col tasto destro del mouse sull'oggetto e scegliere la voce 'Help' dal menu che compare. In *Run Mode* invece, tenendo premuto il tasto CTRL (o Command per il Mac), fate clic col sinistro del mouse e scegliete la voce *Help*.

Le pagine di *Help* spiegano tutte (o quasi) le funzioni di un oggetto e forniscono semplici esempi di utilizzo (fig. 1.11).

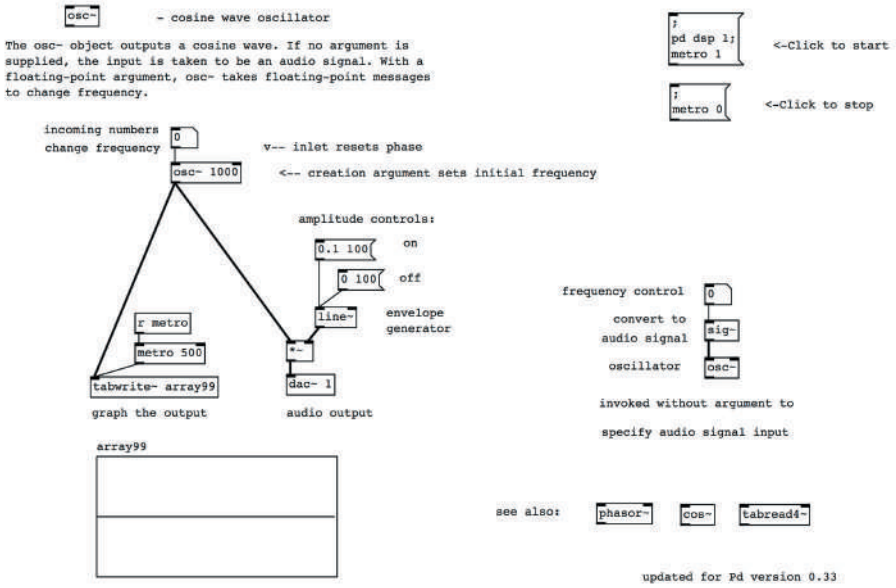


fig. 1.11: pagina di *Help* dell'oggetto `[osc~]`

Sotto il menu *Help* si può anche accedere ad una sorta di *Browser* che permette di navigare all'interno delle pagine della documentazione ufficiale di Pd. Al suo interno si può navigare fra le cartelle che contengono le pagine di *Help* stesse e i Manuali scritti da Miller Puckette, il creatore di Pure Data.

ATTIVITÀ

Create una nuova *Patcher Window* e riproducete la *patch* del file **01_01.pd**. Attenzione a non confondere l'oggetto `[vs.number~]` con il *number box*! Il primo si crea scrivendo la stringa 'vs.number~' all'interno di un *object box* generico, mentre il secondo si crea con la combinazione *<C-3>* o dal menu *Put/ Number*.

(...)

Il capitolo prosegue con:

1.2 FREQUENZA, AMPIEZZA E FORMA D'ONDA
Generatori limitati in banda

1.3 VARIAZIONI DI FREQUENZA E AMPIEZZA NEL TEMPO: INVILUPPI E GLISSANDI
Glissandi
Curve esponenziali e logaritmiche

1.4 RAPPORTO TRA FREQUENZA E INTERVALLO MUSICALE E TRA AMPIEZZA E LIVELLO DI PRESSIONE SONORA
Glissandi "naturali"
Conversione decibel-ampiezza
Scala cromatica

1.5 CENNI SULLA GESTIONE DEI FILE CAMPIONATI
L'oggetto vs.splayer~
Registrazione su file

1.6 CENNI SUL PANNING

1.7 ALTRE CARATTERISTICHE DI PD
Messaggi vs. segnali
L'oggetto cnv

ATTIVITÀ
• Correzione di algoritmi, analisi di algoritmi, completamento di algoritmi, sostituzione di parti di algoritmi

VERIFICHE
• Compiti unitari di reverse engineering

SUSSIDI DIDATTICI
• Lista comandi principali - Lista oggetti nativi Pd - Lista oggetti della libreria Virtual Sound - Lista messaggi per oggetti specifici - Glossario

Interludio A
PROGRAMMAZIONE CON PD

CONTRATTO FORMATIVO

PREREQUISITI PER IL CAPITOLO

- Contenuti del Cap. 1 (teoria e pratica)

OBIETTIVI

Abilità

- Saper utilizzare tutte le funzioni di base del software Pure Data riguardanti i numeri
- Saper generare e controllare sequenze di numeri casuali con possibile uso di metronomo
- Saper costruire algoritmi all'interno di *subpatch* e *abstraction*
- Saper gestire la replicazione di messaggi su varie uscite
- Saper gestire la composizione e scomposizione di liste
- Saper gestire l'utilizzo di argomenti variabili
- Saper gestire la comunicazione fra oggetti senza l'uso di cavi di collegamento virtuali

CONTENUTI

- I numeri in Pd
- Generazione e controllo di numeri casuali con gli oggetti `random`, `vs.drunk` etc.
- Generazione di eventi a ritmo regolare mediante l'oggetto `metro`
- Costruzione di *subpatch* e *abstraction*
- Gestione di liste e argomenti variabili
- Uso degli oggetti `send` e `receive` per comunicazione wireless fra gli oggetti

TEMPI - Cap. 1 (Teoria e Pratica) + Interludio A

Autodidatti

Per 300 ore globali di studio individuale (vol. I, teoria e pratica):

- ca. 100 ore

Corsi

Per un corso globale di 60 ore in classe + 120 di studio individuale (vol. I, teoria e pratica):

- ca. 16 ore frontali + 4 di feedback
- ca. 40 di studio individuale

ATTIVITÀ

Attività al computer:

- Sostituzione di parti di algoritmi, correzione, completamento e analisi di algoritmi, costruzione di nuovi algoritmi

VERIFICHE

- Compiti unitari di Reverse Engineering

SUSSIDI DIDATTICI

- Lista oggetti nativi di Pd – Lista oggetti della libreria Virtual Sound - Lista messaggi per oggetti specifici - Glossario

In questo primo "interludio" approfondiremo alcuni aspetti della programmazione con Pd: si tratta di informazioni essenziali che ci saranno utili per la lettura del resto del libro. Vi consigliamo quindi di non saltarle, a meno che non siate già utenti esperti del programma. È altrettanto importante realizzare tutte le *patch* che vi verranno via via proposte. C'è da fare un piccolo sforzo che vi permetterà di ottenere grandi risultati.

IA.1 OPERATORI BINARI E ORDINE DELLE OPERAZIONI

Installazione e configurazione del sistema

Come ogni linguaggio di programmazione che si rispetti, anche Pd può svolgere diverse operazioni sui numeri: ricreate la semplice *patch* di figura IA.1 (attenzione allo spazio tra '+' e '5').

fig. IA.1: addizione

L'oggetto `[+]` somma il suo argomento (in questo caso 5) a qualunque numero riceva nel suo *inlet*: se mandiamo dei numeri all'oggetto tramite il *number box* superiore (facendo clic sull'oggetto in *run mode* e scrivendo un valore con la tastiera), possiamo vedere il risultato dell'operazione nel *number box* inferiore.

L'ingresso di destra serve a cambiare l'argomento: se inviamo un numero a questo ingresso tramite un nuovo *number box*, questo sostituisce l'argomento dell'oggetto `[+]` nelle somme che si produrranno mandando altri numeri all'ingresso di sinistra.

Aggiungete un *number box* a destra, come in fig. IA.2.

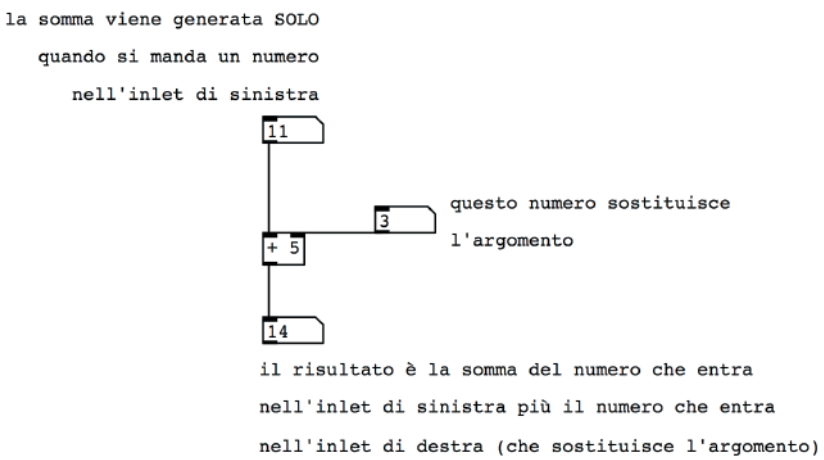


fig. IA.2: addizione con argomento variabile

Se effettuate un po' di prove, noterete che l'operazione viene eseguita solo quando entra un numero nell'ingresso di sinistra, non in quello di destra che serve invece a sostituire l'argomento.

Precisiamo meglio questo concetto: i numeri che entrano nei due ingressi dell'oggetto `[+]` vengono memorizzati in due celle di memoria, che chiameremo *variabili interne*, che si trovano nell'oggetto stesso. Ogni volta che un nuovo numero entra in uno dei due ingressi, la variabile interna corrispondente viene aggiornata: il vecchio numero viene cancellato e sostituito con il nuovo. Oltre a ciò, quando il numero entra nell'ingresso sinistro viene eseguita l'operazione propria dell'oggetto `[+]`, cioè la somma del contenuto delle due variabili interne.
Questa è una caratteristica comune alla grande maggioranza degli oggetti di Pd che svolge la propria funzione solo quando un messaggio entra nell'ingresso di sinistra; i messaggi che entrano dagli altri ingressi servono a modificare gli argomenti, ovvero ad aggiornare le variabili interne, o a modificare il comportamento degli oggetti.

Nel lessico degli oggetti di Pd, viene definito *caldo* l'ingresso di sinistra, che generalmente fa compiere l'operazione all'oggetto stesso, mentre vengono definiti *freddi* tutti gli altri ingressi che aggiornano le variabili interne senza produrre un *output*.
Per riassumere, affinché un'operazione sia eseguita in modo corretto, è essenziale che vengano inviati prima i dati agli ingressi freddi e poi a quello caldo.
Per favorire l'ingresso dei dati in modo che l'ultimo ingresso a ricevere sia quello a sinistra (*caldo*), anche l'uscita dei messaggi dagli oggetti avviene da destra a sinistra. Osserviamo un esempio con l'oggetto `[swap]`, che abbiamo incontrato alla fine del primo capitolo. Questo oggetto inverte l'ordine di due numeri in entrata. Al momento della restituzione verrà inviato prima un numero dall'*outlet* destro, poi dal sinistro (fig. IA.3).

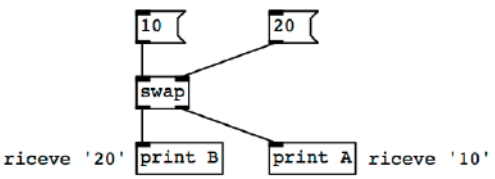


fig. IA.3: ordine di uscita da un oggetto

Per verificare il funzionamento di questa *patch*, dopo averla riprodotta fate clic prima sul *message box* a destra ('20') poi su quello a sinistra ('10'): verranno restituiti prima il valore 10 dall'uscita destra e poi il valore 20 da quella di sinistra.

L'OGGETTO TRIGGER

Entrambi questi comportamenti – *inlet* caldi e freddi e ordine da destra a sinistra – non generano ambiguità. Basta tenere presenti queste 'precedenze' e non si incorre in problemi. Cosa succede invece quando da uno stesso *outlet* escono due *patch cords* come nell'esempio IA.4?

fig. IA.4: ordine di uscita da uno stesso *outlet*

Pd stabilisce che ha la precedenza il *patch cord* che è stato creato per primo. Nell'esempio, che mostra come calcolare il quadrato del numero tre, l'operazione si svolge correttamente solo se è stato creato prima il cavo di collegamento destro. In questo caso, infatti, all'attivazione del *message box* viene inviato prima il messaggio '3' all'*inlet* destro e memorizzato nella corrispondente variabile interna, poi una copia dello stesso messaggio viene inviata all'*inlet* sinistro e viene eseguita l'operazione. Se invece venisse creato prima il cavo sinistro, all'attivazione del *message box* il '3' verrebbe inviato prima all'*inlet* sinistro, e quindi subito eseguito il calcolo: il tre verrebbe moltiplicato per zero perché a destra non sarebbe arrivato alcun valore – e si otterrebbe un risultato inatteso.

Guardando la *patch* è impossibile stabilire se sia stata costruita correttamente ed è quindi assolutamente sconsigliabile usare questo tipo di configurazione. L'oggetto che permette di determinare con certezza le priorità nell'invio dei messaggi è `[trigger]`, che si può usare anche nella sua forma abbreviata `[t]` (fig. IA.5).
L'oggetto accetta un numero arbitrario di *creation arguments* ognuno dei quali indica il *tipo* di messaggio da restituire. Per ognuno degli argomenti viene creato un *outlet*.

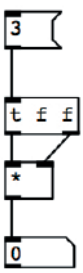


fig. IA.5: oggetto `[trigger]`

Quando riceve un messaggio dal suo unico *inlet*, `[trigger]` invia lo stesso da tutti i suoi *outlet*, partendo da quello di destra. Nell'esempio vengono creati due *outlet* di tipo numerico (*float*); quando viene ricevuto il messaggio '3', questo valore viene immediatamente restituito, prima dall'*outlet* destro, poi dal sinistro. In questo modo l'oggetto `[*]` riceve nel giusto ordine i due membri della moltiplicazione e può effettuare il calcolo nel modo corretto.

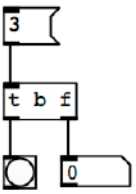


fig. IA.6: conversione di tipo con `[trigger]`

La necessità di *tipizzare* gli *outlet* è motivata dal fatto che `[trigger]` può anche convertire i tipi di atomi che riceve. In figura IA.6 un `[trigger]` contiene due argomenti: il primo, *b*, abbreviazione di *bang*, indica che qualunque numero entri nell'*inlet*, dalla prima uscita verrà sempre restituito un *bang* mentre il secondo argomento, *f*, abbreviazione di *float*, indica che dall'uscita destra uscirà il numero che `[trigger]` riceve dall'*inlet*.

Nell'esempio in figura il *message box* invia a `[trigger]` il numero 3, che viene restituito dall'*outlet* destro, mentre dall'*outlet* sinistro viene restituito un *bang*.

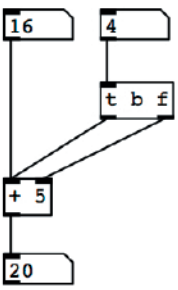


fig. IA.7: rendere *caldo* un ingresso *freddo*

In figura IA.7 usiamo il meccanismo della conversione per realizzare un semplice algoritmo che 'scalda' l'*inlet* freddo di un oggetto `[+]`: quando inviamo l'addendo all'*inlet* sinistro, l'oggetto `[+]` per sua natura esegue l'operazione e restituisce il risultato. Quando invece inviamo un valore dal *number box* in alto a destra questo invia il valore a `[trigger]` che prima fa uscire il valore stesso dal suo *outlet* destro, memorizzandolo nella corrispondente variabile interna, e immediatamente dopo invia un *bang* all'ingresso sinistro di `[+]`, che 'forza' l'oggetto a produrre il risultato addizionando i due valori memorizzati nelle variabili interne.

Per finire ricapitoliamo i diversi argomenti che l'oggetto `[trigger]` accetta, introducendone alcuni che ancora non conosciamo:

f: numero *float*
s: *symbol*
b: *bang*
l: lista (cioè un messaggio composto da più atomi)
a: *anything* (questo argomento lascia passare qualunque cosa senza convertirla)

DEPTH FIRST

L'ultimo criterio che regola l'ordine delle operazioni è quello definito *depth first*. In sostanza questa regola prescrive che un messaggio intraprende un solo viaggio alla volta e finché non giunge ad un punto di terminazione non ne intraprende un altro. In figura IA.8 il *bang* iniziale percorre prima il ramo destro dell'algoritmo, fino a giungere all'oggetto `[print A]`, poi percorre il ramo centrale per giungere a `[print B]`; infine percorre il ramo sinistro fino all'oggetto `[print C]`.

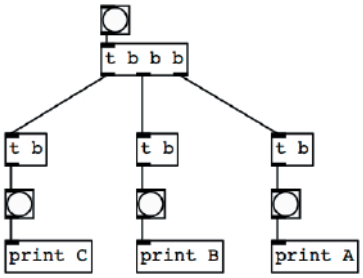


fig. IA.8: il principio *depth first*

Facendo clic sul *bang* in alto vi accorgerete che "a vista d'occhio" i tre *bang* si accendono contemporaneamente perché il viaggio attraverso i tre rami della *patch* avviene in tempi brevissimi. Se osservate la *pd window* però, potete osservare che il primo `[print]` a ricevere il *bang* è quello etichettato con 'A', poi quello con etichetta 'B' e infine l'ultimo a sinistra, con etichetta 'C'.

Tornando all'oggetto `[+]` di Pd, vediamo ora una semplice applicazione musicale: aprite il file **IA_01_trasposizione.pd** (fig. IA.9).

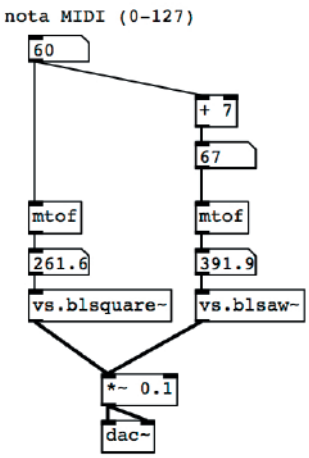


fig. IA.9: file IA_01_trasposizione.pd

In questa *patch*, ogni volta che viene impostato un valore sul *number box* in alto vengono generate due note, la seconda delle quali si trova sette semitoni sopra la prima, ovvero una quinta sopra. I pratica, ogni volta che immettiamo un valore nel *number box* la corrispondente nota MIDI viene inviata all'oggetto `[mtof]` di sinistra che la converte in frequenza. Allo stesso tempo però la stessa nota viene inviata ad un sommatore che aggiunge il valore '7' alla nota, ovvero aggiunge sette semitoni alla nota di partenza. La somma viene quindi inviata ad un secondo oggetto `[mtof]`.

Tramite il sommatore, quindi, è possibile effettuare la trasposizione delle note MIDI ad intervalli arbitrari (cioè a distanze fra due note che possiamo determinare noi). Dopo che i valori MIDI delle due note sono stati convertiti in valori frequenza, vengono inviati a due oscillatori, `[vs.blsquare~]` e `[vs.blsaw~]` che eseguono l'intervallo di quinta. Impostate nella *patch* di fig. IA.9 altri intervalli (terza, cioè quattro semintoni, quarta, ovvero cinque semitoni e così via), aggiungete poi un altro sommatore, collegato ad un nuovo `[mtof]` a sua volta collegato ad un oscillatore e fate in modo che ad ogni valore inserito nel *number box* in alto corrisponda un accordo maggiore di tre note: se ad esempio scrivete nel *number box* il valore 60, si ottiene l'accordo 60, 64, 67.

RAPPRESENTAZIONE DEI NUMERI IN PD E ALTRE OPERAZIONI

Per approfondire l'uso degli operatori binari è importante capire come i numeri sono rappresentati in Pd. A differenza della maggior parte dei linguaggi di programmazione, che distingue fra numeri interi e numeri con la virgola, in Pd i numeri sono sempre rappresentati come numeri con la virgola, in inglese *float numbers*.

Questo può creare dei problemi quando si devono svolgere operazioni che richiedano l'uso dei numeri interi. Nell'esempio a sinistra in figura IA.10 un programmatore si aspetterebbe una divisione fra due numeri interi (l'oggetto `[/]` ovviamente è deputato all'esecuzione della operazione di divisione), che in genere restituisce un risultato intero. In Pd invece i due numeri in entrata sono considerati numeri di tipo *float* e il risultato è esso stesso un *float*. Per effettuare una divisione intera (con un risultato di tipo intero) bisogna convertire in intero il risultato con l'oggetto `[int]`. Ad essere precisi `[int]` elimina la parte frazionaria di un *float*, restituendo nuovamente un *float* (vedi la parte destra in fig. IA.10).
L'oggetto `[int]` e il suo omologo per i numeri con virgola `[float]`, rispetti-

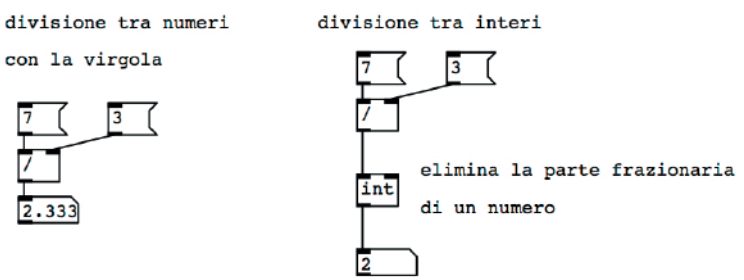


fig. IA.10: divisione

vamente abbreviabili in `[i]` ed `[f]`, hanno anche un'altra funzione importante: mantengono in memoria l'ultimo valore ricevuto dall'*inlet* destro o sinistro. Se ad esempio inviamo ad uno di questi oggetti un valore all'entrata destra, questo può essere restituito dall'*outlet* inviando un *bang* all'*inlet* sinistro (fig. IA.11).

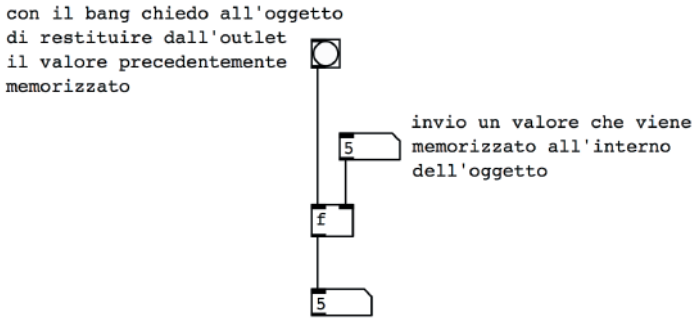


fig IA.11: oggetto `[float]`

Gli operatori binari che abbiamo esaminato (`[+]`,`[*]` e `[/]`) insieme a `[-]` e `[pow]`, costituiscono il gruppo degli operatori binari aritmetici, che avremo occasione di incontrare spessissimo nel corso di questo testo. Aprite una nuova *patcher window*, create alcuni oggetti per le operazioni aritmetiche e verificatene il comportamento.

Per finire, mostriamo un semplice esempio di utilizzo di operatori binari per costruire un meccanismo che, agendo su due oscillatori, usi una frequenza per il primo e la trasposizione alla quinta giusta per l'altro. Aprite il file **IA_02_quinte.pd** (fig. IA.12).

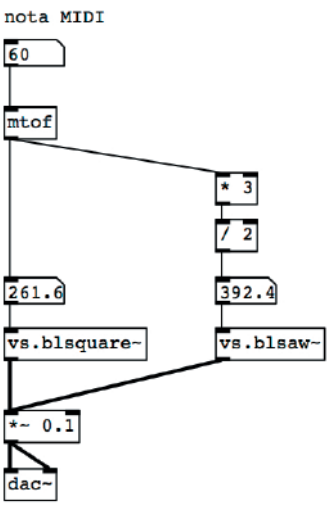


fig. IA.12: file IA_02.quinte.pd

Nella *patch* osserviamo che la nota MIDI inserita tramite il *number box* a sinistra viene trasformata in frequenza tramite `[mtof]` e inviata ad un oscillatore a onde quadre limitato in banda (`[vs.blsquare~]`); questa frequenza viene inoltre moltiplicata per 3/2, in modo da ottenere la frequenza che si trova una quinta sopra, e inviata ad un oscillatore a dente di sega limitato in banda

(`[vs.blsaw~]`). L'uscita dell'oggetto `[mtof]` quindi si sdoppia, un cavo va al *number box* sinistro mentre l'altro cavo va all'oggetto `[*]` con argomento 3.

Se confrontate le figure IA.9 e IA.12 noterete che la frequenza della nota che si trova una quinta sopra è leggermente diversa (e di conseguenza è diverso anche il timbro complessivo).
Tramite i due `[mtof]` infatti abbiamo calcolato un intervallo di quinta *temperata* (quella che si usa normalmente nella musica occidentale) che è esattamente di sette semitoni temperati; tramite il rapporto 3/2 invece calcoliamo una quinta *naturale* che è più larga di quella temperata di circa due *cents* (centesimi di semitono temperato).

Per sapere quali e quanti sono gli operatori binari potete aprire il file **help-intro.pd**, a cui si accede tramite il menu *Help/List of objects*, che mostra l'elenco con tutti gli oggetti di Pd (fig. IA.13): è un buon punto di partenza per conoscere gli oggetti di Pd e la loro funzione.

```
The following is a list of built-in objects in Pd. (Not
included in this list are messages, atoms, graphs, etc.
which aren't typed into object boxes but come straight off
the "add" menu.) Right-click (or control-click on a
Macintosh) on any object to get its "help window".

--------------- GENERAL ----------------

[bang]          - output a bang message
[float]         - store and recall a number
[symbol]        - store and recall a symbol
[int]           - store and recall an integer
[send]          - send a message to a named object
[receive]       - catch "sent" messages
[select]        - test for matching numbers or symbols
[route]         - route messages according to first element
[pack]          - make compound messages
[unpack]        - get elements of compound messages
[trigger]       - sequence and convert messagess
[spigot]        - interruptible message connection
[moses]         - part a numeric stream
[until]         - looping mechanism
[print]         - print out messages
[makefilename]   - format a symbol with a variable field
[change]        - remove repeated numbers from a stream
[swap]          - swap two numbers
[value]         - shared numeric value
[list]          - manipulate lists
------------------------- TIME -------------------------

[delay]         - send a message after a time delay
[metro]         - send a message periodically
```

fig. IA.13: parte del file help-intro.pd

(...)

Il capitolo prosegue con:

IA.2 GENERAZIONE DI NUMERI CASUALI

IA.3 GESTIONE DEL TEMPO: METRO

IA.4 SUBPATCH E ABSTRACTION
Abstraction
Graph-on-parent

IA.5 ALTRI GENERATORI RANDOM

IA.6 LISTE
Gli oggetti pack e unpack
Gli oggetti list append e list prepend
Altri oggetti per manipolare liste

IA.7 IL MESSAGE BOX E GLI ARGOMENTI VARIABILI
Il messaggio di comando "set"
Gli argomenti variabili: il segno del dollaro ($)

IA.9 ARRAY
L'oggetto vs.uzi
L'oggetto tabwrite
Inviluppi
Inviluppi prefabbricati

IA.10 SEND E RECEIVE

ATTIVITÀ
• Analisi di algoritmi, completamento di algoritmi, sostituzione di parti di algoritmi, correzione di algoritmi

VERIFICHE
• Compiti unitari di reverse engineering

SUSSIDI DIDATTICI
• Lista oggetti nativi Pd - Lista oggetti della libreria Virtual Sound - Lista messaggi per oggetti specifici - Glossario

2T
SINTESI ADDITIVA E SINTESI VETTORIALE

CONTRATTO FORMATIVO

PREREQUISITI PER IL CAPITOLO

- Contenuti del cap. 1 (teoria)

OBIETTIVI

Conoscenze

- Conoscere la teoria della somma di onde (fase, interferenza costruttiva, distruttiva etc.)
- Conoscere la teoria e gli utilizzi di base della sintesi additiva a spettro fisso e spettro variabile, a spettro armonico e inarmonico
- Conoscere il rapporto fra fase e battimenti
- Conoscere il modo in cui si utilizzano le tabelle e in cui avviene l'interpolazione
- Conoscere la teoria e gli utilizzi di base della sintesi vettoriale

Abilità

- Saper individuare all'ascolto le caratteristiche di base dei suoni armonici e inarmonici
- Saper riconoscere all'ascolto i battimenti
- Saper individuare le varie fasi dell'inviluppo di un suono e saperne descrivere le caratteristiche

CONTENUTI

- Sintesi additiva a spettro fisso e variabile
- Suoni armonici e inarmonici
- Fase e battimenti
- Interpolazione
- Sintesi vettoriale

TEMPI - Cap. 2 (Teoria e Pratica)

Autodidatti

Per 300 ore globali di studio individuale (vol. I, teoria e pratica):

- ca. 60 ore

Corsi

Per un corso globale di 60 ore in classe + 120 di studio individuale (vol. I, teoria e pratica):

- ca. 10 ore frontali + 2 di feedback
- ca. 24 di studio individuale

ATTIVITÀ

Esempi interattivi

VERIFICHE

Test a risposte brevi
Test con ascolto e analisi

SUSSIDI DIDATTICI

Concetti di base - Glossario - Discografia - Un pò di storia

2.1 SINTESI ADDITIVA A SPETTRO FISSO

Il suono prodotto da uno strumento acustico o il suono in generale è propriamente un'oscillazione complessa esprimibile attraverso un insieme di più vibrazioni elementari che vengono prodotte contemporaneamente dallo strumento stesso: la somma di queste diverse vibrazioni contribuisce in modo determinante al timbro percepito e ne determina quindi la forma d'onda. Ogni suono può quindi essere scomposto in sinusoidi che, come abbiamo detto nel capitolo precedente, costituiscono il mattone fondamentale con cui è possibile costruire ogni altra forma d'onda. Ognuna di queste sinusoidi, o componenti del suono, ha una propria frequenza, ampiezza e fase. L'insieme delle frequenze, ampiezze e fasi delle sinusoidi che formano un determinato suono è definito **spettro sonoro**: vedremo più avanti come può essere rappresentato graficamente.
Non solo i suoni naturali, ma anche i suoni sintetici possono essere scomposti in un insieme di sinusoidi: le forme d'onda che abbiamo descritto nel paragrafo 1.2, quindi, hanno ciascuna un diverso spettro sonoro, ognuna contiene cioè una diversa mistura (combinazione) di sinusoidi (ad eccezione ovviamente della forma d'onda sinusoidale che contiene soltanto sé stessa!).

SPETTRO E FORMA D'ONDA

Spettro e forma d'onda sono due modi differenti per descrivere un suono. La forma d'onda è la rappresentazione grafica dell'ampiezza in funzione del tempo.[1] Nel grafico in figura 2.1 possiamo osservare la forma d'onda di un **suono complesso** in cui sull'asse delle *x* abbiamo il tempo, sull'asse delle *y* l'ampiezza; notiamo che la forma d'onda di questo suono è *bipolare*, cioè i suoi valori d'ampiezza oscillano al di sopra e al di sotto dello zero. Questa è una rappresentazione grafica nel **dominio del tempo** (*time domain*), vengono cioè tracciate tutte le ampiezze istantanee che vanno a formare, istante dopo istante la forma d'onda del suono complesso.

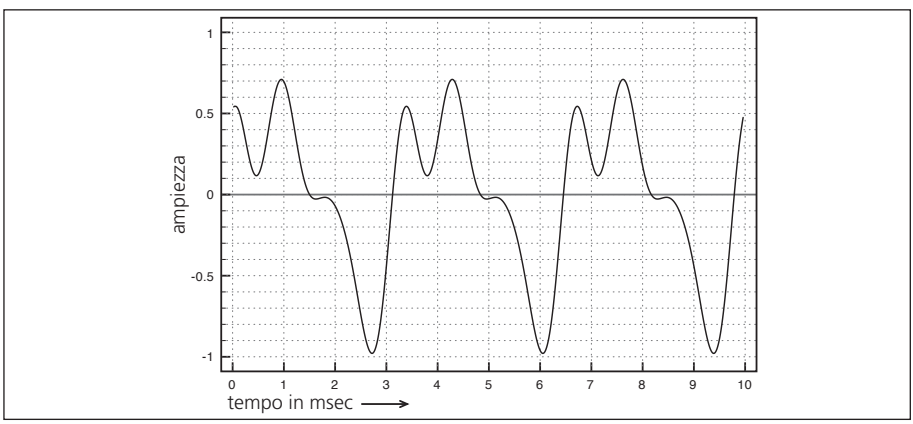


fig. 2.1: forma d'onda di un suono complesso

[1] Nel caso di suoni periodici la forma d'onda può essere rappresentata da un singolo ciclo.

Nella fig. 2.2a vediamo invece come si può scomporre in sinusoidi il suono complesso appena osservato: abbiamo infatti quattro sinusoidi con frequenza e ampiezza diverse, che sommate costituiscono il suono complesso rappresentato nel grafico precedente.

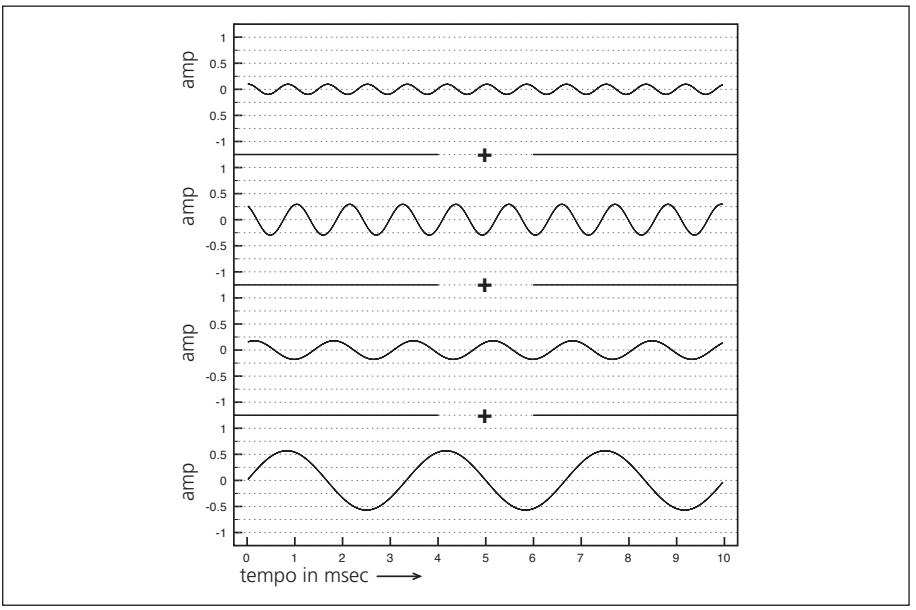


fig. 2.2a: scomposizione in sinusoidi di un suono complesso

Se vogliamo "fotografare" in modo più chiaro le varie frequenze di questo suono e la loro ampiezza, possiamo affidarci ad un grafico dello spettro sonoro, cioè la rappresentazione dell'ampiezza delle componenti in funzione della frequenza (ovvero nel **dominio della frequenza** o *frequency domain*). In questo caso sull'asse delle *x* abbiamo i valori di frequenza, sull'asse delle *y* i valori d'ampiezza. La figura 2.2b mostra le ampiezze di picco di ciascuna componente presente nel segnale.

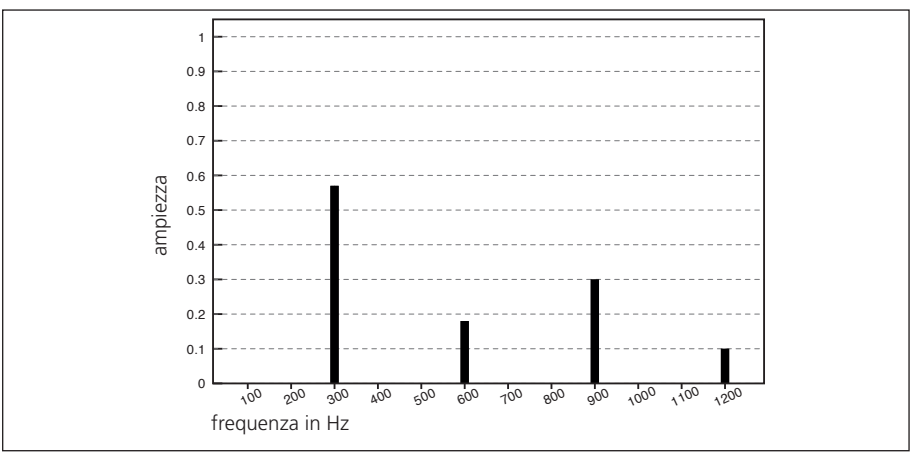


fig. 2.2b: spettro sonoro

Per vedere l'andamento nel tempo delle componenti di un suono possiamo usare anche il **sonogramma**, che ci mostra sull'asse delle *y* le frequenze e in quello delle *x* il tempo (fig. 2.2c). Le linee corrispondenti alle frequenze delle componenti sono più o meno marcate in funzione della loro ampiezza. In questo caso abbiamo 4 linee rette, perché si tratta di uno spettro fisso.

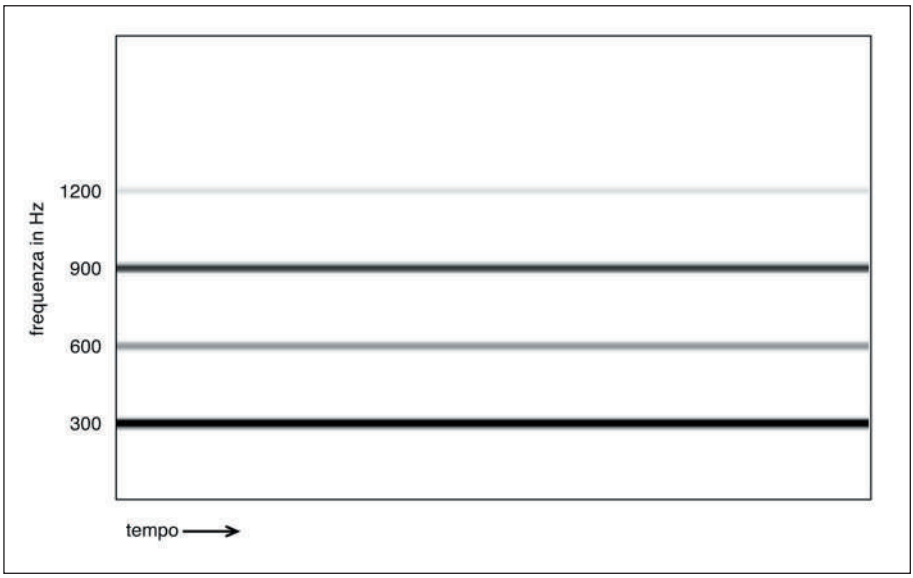


fig. 2.2c: sonogramma (o spettrogramma)

Adesso consideriamo il processo opposto, quello cioè in cui, invece di scomporre un suono complesso in sinusoidi, partiamo dalle singole sinusoidi e creiamo un suono complesso. La tecnica che ci permette di creare qualunque forma d'onda partendo dalla somma di sinusoidi si chiama **sintesi additiva** (fig. 2.3).

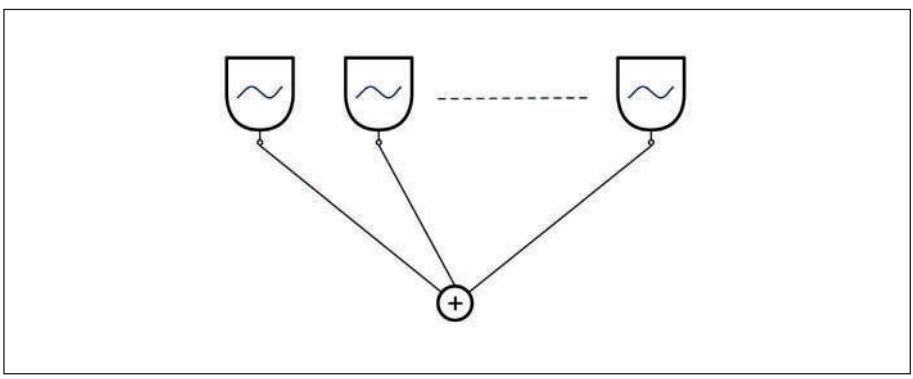


fig. 2.3: somma di segnali in uscita da oscillatori sinusoidali

In fig. 2.4 sono mostrate due onde sonore, A e B, e la loro somma C.

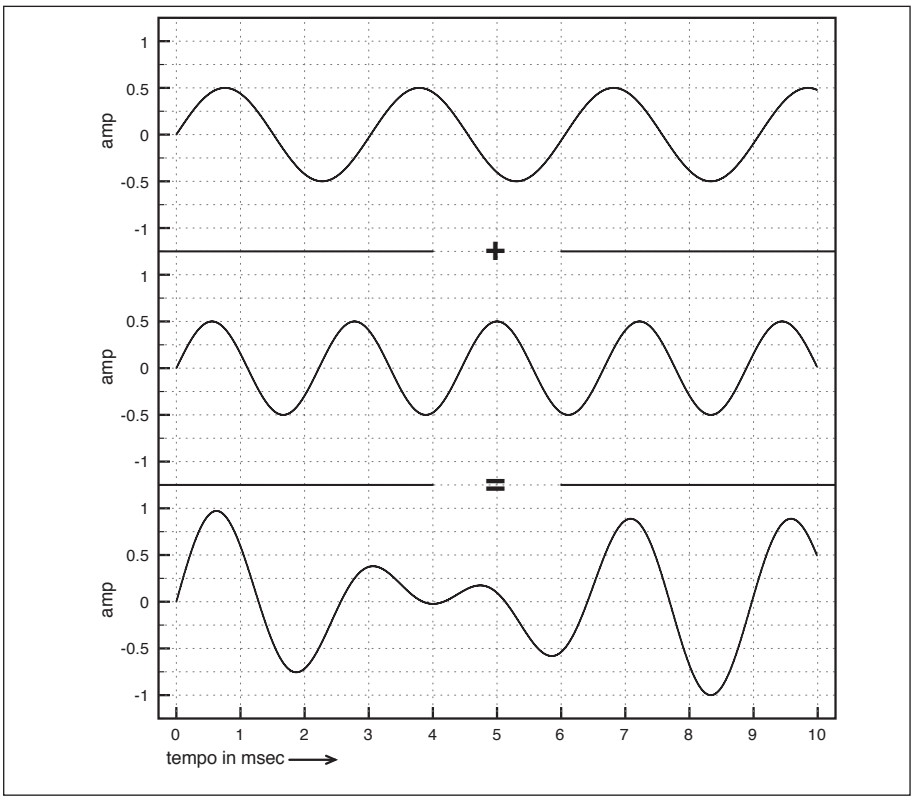


fig. 2.4: rappresentazione grafica di una somma di sinusoidi

Come si può facilmente verificare, le ampiezze istantanee dell'onda C sono ottenute sommando punto per punto le ampiezze istantanee dell'onda A a quelle dell'onda B. Istante per istante i valori dell'ampiezza delle diverse onde si sommano *algebricamente*, cioè con il loro segno, positivo o negativo. Nel caso in cui le ampiezze di A e B, in un dato istante, siano entrambe positive o entrambe negative, il valore assoluto di ampiezza di C risulterà maggiore di ciascuna delle singole componenti, si otterrà cioè un'**interferenza costruttiva**, ad es.

A = -0.3
B = -0.2
C = -0.5

Nel caso in cui invece le ampiezze di A e B, in un dato istante, siano una positiva e l'altra negativa, il valore assoluto della loro somma algebrica C sarà minore di almeno una delle due componenti, e si avrà perciò un'**interferenza distruttiva**, ad es.

A = 0.3
B = -0.1
C = 0.2

"La maggior parte, anzi la quasi totalità, dei suoni che udiamo nel mondo reale non è composta da suoni *puri*, ma da **suoni complessi**, cioè suoni scomponibili in una maggiore o minore quantità di suoni puri; questi vengono detti *componenti* del suono complesso. Per meglio comprendere questo fenomeno, stabiliamo un'analogia con l'ottica. È noto come alcuni colori siano *puri*, cioè non ulteriormente scomponibili (rosso, arancio, giallo, fino al violetto). A ciascuno di essi corrisponde una certa *lunghezza d'onda* del raggio luminoso. Nel caso in cui sia presente soltanto uno dei colori puri, il prisma, che scompone la luce bianca nei sette colori dello spettro luminoso, mostrerà solamente quella componente. La medesima cosa avviene per il suono. A una certa **lunghezza d'onda**[2] del suono corrisponde una certa altezza percepita. Se non è presente contemporaneamente nessun'altra frequenza, il suono sarà *puro*. Un suono puro, come sappiamo, ha forma d'onda *sinusoidale*."
(Bianchini, R., Cipriani, A., 2001, pp. 69-70)

Le componenti di un suono complesso possono avere frequenze che sono multipli interi della frequenza della componente più grave. In questo caso la componente più grave si chiama **fondamentale** e le altre si chiamano **componenti armoniche** (ad esempio, fondamentale = 100 Hz, altre componenti = 200 Hz, 300 Hz, 400 Hz etc.) La componente che ha una frequenza doppia rispetto a quella della fondamentale si chiama *seconda armonica*; la componente di frequenza tripla della fondamentale viene detta *terza armonica* etc. Quando, come in questo caso, le componenti del suono sono multipli interi della fondamentale il suono si dice *armonico*. Notiamo che in un suono armonico la frequenza della fondamentale rappresenta il *massimo comun divisore* delle frequenze di tutte le altre componenti: è cioè il massimo numero che divide esattamente (senza resto) tutte le frequenze.

ESEMPIO INTERATTIVO 2A • *COMPONENTI ARMONICHE*

Se i suoni puri che compongono un suono complesso non sono multipli interi della componente più grave, abbiamo un suono inarmonico e le componenti prendono il nome di...

(...)

[2] "La lunghezza di un ciclo si dice **lunghezza d'onda** e si misura in metri [m] o in centimetri [cm]. Questo è lo spazio che un ciclo occupa fisicamente nell'aria, e se il suono fosse visibile potrebbe facilmente essere misurato, per esempio con un metro." (Bianchini, R. 2003)

Il capitolo prosegue con:

La fase
Spettri armonici e inarmonici
Periodico/aperiodico, armonico/inarmonico
Interpolazione nella lettura di tabelle

2.2 BATTIMENTI

2.3 DISSOLVENZA INCROCIATA DI TABELLE: SINTESI VETTORIALE

2.4 SINTESI ADDITIVA A SPETTRO VARIABILE

ATTIVITÀ
• Esempi interattivi

VERIFICHE
• Test a risposte brevi
• Test con ascolto e analisi

SUSSIDI DIDATTICI
• Concetti di base - Glossario - Discografia - Un po' di storia

2P
SINTESI ADDITIVA E SINTESI VETTORIALE

CONTRATTO FORMATIVO

PREREQUISITI PER IL CAPITOLO

- Cap. 1 (teoria e pratica), Cap.2 (teoria), Interludio A

OBIETTIVI

Abilità

- Saper sintetizzare un suono complesso partendo da semplici sinusoidi
- Saper sintetizzare suoni armonici e inarmonici in sintesi additiva e tabellare e trasformare gli uni negli altri e viceversa tramite controlli d'ampiezza e frequenza
- Saper realizzare forme d'onda triangolari, quadre o a dente di sega approssimate mediante addizione di componenti armoniche sinusoidali
- Saper controllare i battimenti fra due suoni sinusoidali o armonici
- Saper sintetizzare suoni in sintesi vettoriale

Competenze

- Saper realizzare un breve studio sonoro basato sulle tecniche di sintesi additiva e memorizzarlo su file audio.

CONTENUTI

- Sintesi additiva a spettro fisso e variabile
- Suoni armonici e inarmonici
- Fase e battimenti
- Interpolazione
- Sintesi vettoriale

TEMPI - Cap. 2 (Teoria e Pratica)

Autodidatti

Per 300 ore globali di studio individuale (vol. I, teoria e pratica):

- ca. 60 ore

Corsi

Per un corso globale di 60 ore in classe + 120 di studio individuale (vol. I, teoria e pratica):

- ca. 10 ore frontali + 2 di feedback
- ca. 24 di studio individuale

ATTIVITÀ

- Attività al computer: sostituzione di parti di algoritmi, correzione, completamento e analisi di algoritmi, costruzione di nuovi algoritmi

VERIFICHE

- Realizzazione di uno studio breve
- Compiti unitari di Reverse Engineering

SUSSIDI DIDATTICI

- Lista oggetti nativi di Pd – Lista oggetti della libreria Virtual Sound - Lista messaggi per oggetti specifici

2.1 SINTESI ADDITIVA A SPETTRO FISSO

La prima *patch* di questo capitolo realizza un algoritmo di sintesi additiva a *spettro armonico*. Come sappiamo si dice *armonico* uno spettro in cui le frequenze sono multipli interi della frequenza fondamentale. La *patch* fa riferimento alla figura 2.12 del paragrafo 2.1 della teoria, in cui 10 oscillatori vengono sommati tramite un miscelatore.
Ogni oscillatore riceve un valore che imposta la frequenza e un altro che ne definisce l'ampiezza. Per avere uno spettro armonico sarà sufficiente scegliere una frequenza fondamentale e moltiplicarla per una serie di numeri interi successivi. In questo modo il primo oscillatore avrà la frequenza fondamentale, il secondo avrà frequenza doppia, il terzo tripla e così via.
La figura 2.1 mostra la *patch*, contenuta nel file **02_01_spettro_armonico.pd**.

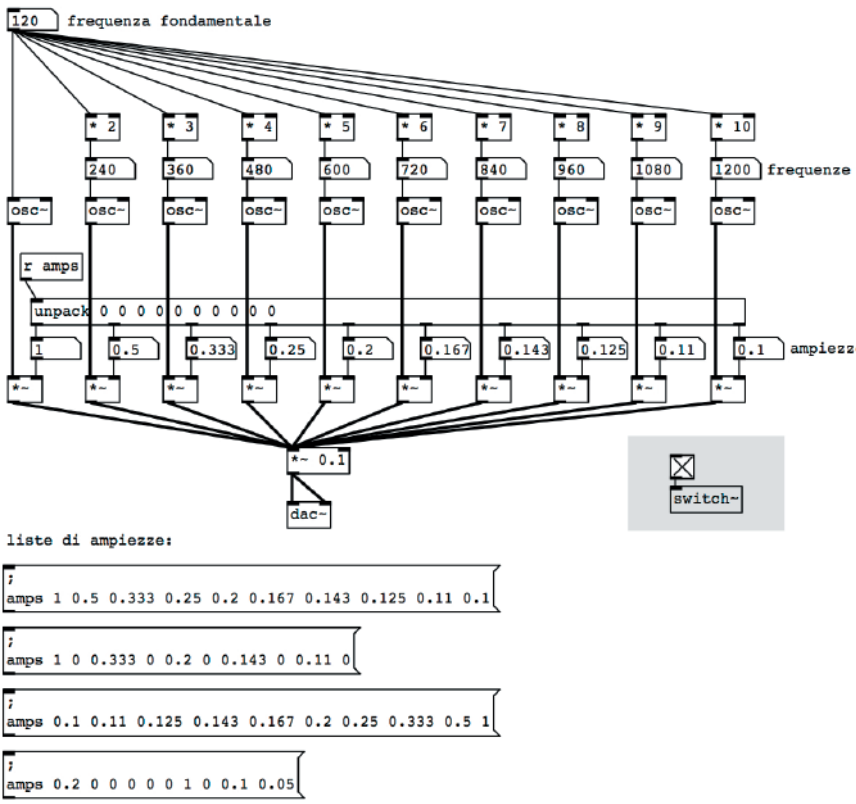


fig. 2.1: file 02_01_spettro_armonico.pd

Il *number box* in alto a sinistra permette di impostare la frequenza fondamentale, che entra direttamente nel primo `[osc~]`. Tutti gli altri oscillatori ricevono la stessa frequenza moltiplicata per un intero tramite l'oggetto `[*]`. Il segnale in uscita da ogni `[osc~]` è collegato a un `[*~]`: in questo modo il segnale può essere riscalato in ampiezza: ogni `[*~]` riceve infatti il fattore di riscalamento da

un *number box*. Infine, i 10 segnali vengono inviati all'*inlet* sinistro di un `[*~]` con argomento 0.1, quindi la somma dei segnali viene moltiplicata per 0.1, in modo che il segnale risultante non superi la soglia massima[1].

Questo tipo di algoritmo definisce uno spettro fisso, perché le frequenze variano al variare della fondamentale, ma mantengono sempre lo stesso rapporto fra loro. Le ampiezze invece possono essere cambiate a piacimento, in modo da creare timbri diversi. I *message box* in basso contengono liste di ampiezze prestabilite, che vengono inviate all'oggetto `[receive]` (`[r]`) con argomento *amps* e poi "spacchettate" tramite `[unpack]`.

Nulla impedisce di impostare manualmente le ampiezze agendo con il mouse sui singoli *number box*. L'unica accortezza è quella di agire col mouse tenendo premuto il tasto SHIFT, per generare valori decimali e non eccedere 1. Ascoltate i diversi timbri facendo clic sui *message box* dopo aver attivato il motore audio.

In basso a destra della *patch* trovate un oggetto `[switch~]` collegato ad un *toggle*: si usa per attivare o disattivare l'audio solo nella *patch* in cui risiede e in tutte le *subpatch* e le *abstraction* che a quella *patch* fanno riferimento[2]. In sostanza con `[switch~]` potete decidere quali *patch* far "suonare" e quali lasciare mute, evitando così fastidiose sovrapposizioni dell'audio. Quando il *toggle* invia il messaggio '1', l'audio viene attivato nella *patch* in cui si trova; al contrario, con il messaggio '0', lo disattiva, lasciando attive le altre. Naturalmente, affinché `[switch~]` possa svolgere il suo compito, il motore audio generale deve essere attivo[3].

Per evitare ridondanze grafiche, l'oggetto `[switch~]` non compare nelle successive figure di questo testo, ma le *patch* allegate ne sono provviste. Può essere utile, quando copiate gli algoritmi di questo libro, corredarli con l'oggetto `[switch~]` collegato ad un *toggle*, in modo da poter gestire comodamente l'audio distribuito fra le *patch* aperte. L'unica accortezza è che l'oggetto `[switch~]` sia creato prima dell'ultimo oggetto con la tilde della *patch*. In caso contrario, `[switch~]` non funzionerà, a meno di non salvare con un nome la *patch* in un file.

---

[1] Ogni oscillatore restituisce un segnale che può avere al massimo ampiezza pari a 1. Se tutti i segnali escono alla massima ampiezza, la somma degli oscillatori sarà un segnale di ampiezza 10. Moltiplicare questo segnale per 0.1 garantisce che la somma possa dare al massimo un segnale di ampiezza pari a 1.

[2] In realtà l'oggetto [switch~] svolge anche altre funzioni, ma queste saranno esaminate successivamente.

[3] Per attivare il motore audio, come sempre, si usa la combinazione da tastiera <C-/>.

ATTIVITÀ

1. Usando la *patch* **02_01_spettro_armonico_fisso.pd** create due nuove liste per le ampiezze in due nuovi *message box*, uno con tutte le ampiezze delle armoniche dispari a 1 e le pari a 0 e l'altro con le armoniche pari a 1 e le dispari a 0, qual è la differenza più evidente? E perché?

2. Sempre con la *patch* **02_01_spettro_armonico_fisso.pd**, (facendo riferimento alla questione della fondamentale mancante o frequenza fantasma, trattata nel par. 2.1 della teoria) partite da uno spettro in cui tutte le ampiezze delle armoniche siano a 1, poi azzerate l'ampiezza della fondamentale. Sentirete ancora uno spettro la cui fondamentale è pari alla frequenza che abbiamo azzerato. Provate a ridurre a zero l'ampiezza delle armoniche successive, una ad una. A che punto non si percepisce più la fondamentale?

LA FASE

Come avrete osservato, l'oggetto `[osc~]` possiede due *inlet*: il sinistro, che è quello che abbiamo usato fino ad ora, serve ad impostare la frequenza dell'oscillatore e può ricevere un messaggio o un segnale. L'*inlet* destro invece, consente di impostare la fase e riceve solo messaggi. Il valore che si invia a questo *inlet* è un decimale compreso fra zero e uno, che rappresenta la *fase normalizzata*. Un valore di zero corrisponde ad un angolo di 0° o di 0 radianti, un valore di 0.25 corrisponde a 90° ($\pi/2$ radianti); 0.5 equivale a 180°, quindi a π radianti e così via. Il valore di 1 corrisponde al termine di un ciclo, quindi 360°, che è uguale a 0°.

Quando si invia un messaggio all'*inlet* destro di `[osc~]` si "costringe" il generatore a ricominciare la forma d'onda a partire dal punto indicato nel messaggio stesso. Ad esempio il messaggio '0' produce la generazione della forma d'onda a partire dalla fase che corrisponde al valore 0.

Riproducete la *patch* in figura 2.2, attivate il motore audio e fate clic ripetutamente sul *message box* per reimpostare la fase: potrete udire un *click* ogni volta che attivate il *message box* perché l'esecuzione della forma d'onda viene interrotta per essere rieseguita dall'inizio (cioè dal valore indicato nel *message box*).

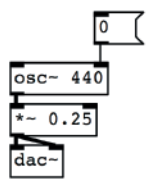


fig. 2.2: reimpostazione della fase in un oggetto `[osc~]`

La figura 2.3 mostra quello che accade in un oscillatore quando viene eseguita la reimpostazione (in inglese *reset*) della fase. Appena il *reset* viene attivato tramite il *message box*, il generatore riesegue la forma d'onda a partire dal punto indicato dal messaggio, generando un punto di *disconinuità* che produce un *click* udibile. In questo caso l'onda viene ripristinata con il messaggio '0' (come in fig. 2.2): tale messaggio costringe l'oscillatore a generare la forma d'onda ricominciando dall'inizio.
L'inizio di un ciclo di una sinusoide, come sappiamo dal cap. 1.2 della parte di teoria, corrisponde a un valore di ampiezza pari a 0. Perché, al contrario, il valore di ampiezza in corrispondenza del *reset* che vedete nell'immagine è 1? Perché l'oggetto `[osc~]` in realtà non segue l'andamento della funzione seno, bensì quello della funzione coseno, il cui inizio di ciclo corrisponde al valore massimo di ampiezza.

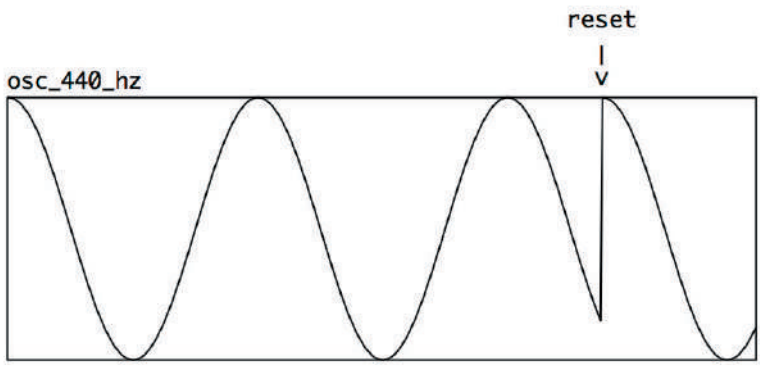


fig. 2.3: rappresentazione grafica del punto di discontinuità generato dal *reset* della fase

Usata in questo modo, la fase non è un parametro molto interessante. Lo diventa invece quando si mettono in relazione due onde con la stessa frequenza che vengono sommate fra loro. Quando le due onde sono perfettamente in fase, la somma sarà un segnale con la stessa forma d'onda, ma di ampiezza doppia.

Copiate la *patch* in figura 2.4, attivate il motore audio e fate *click* sul *message box* a sinistra, etichettato con 'in fase': entrambi gli oscillatori subiscono nello stesso istante un *reset* della fase con lo stesso valore e la loro somma è rappresentata da una nuova forma d'onda con ampiezza doppia.

Cosa succede se invece fate clic sul *number box* etichettato con 'opposizione di fase'? Il generatore a sinistra subisce un *reset* della fase che lo riporta allo stato iniziale, quello a destra subisce ugualmente un *reset*, ma la nuova onda generata inizierà il suo periodo dal valore di fase 0.5, che corrisponde a 180°. Questo valore produce un onda in opposizione di fase rispetto all'altra, quindi, quando i valori delle due onde si sommano, istante per istante, generano un segnale costante pari a 0: non si sente più niente.

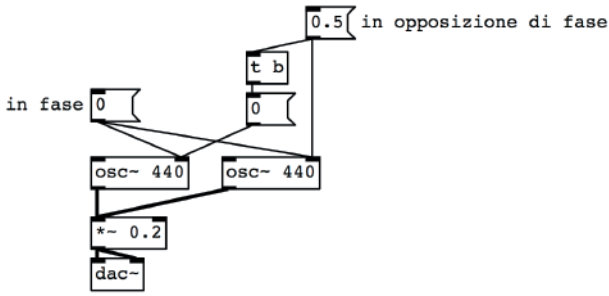


fig. 2.4: *reset* della fase in due oscillatori

Ora aprite il file **02_02_fase.pd**, mostrato in figura 2.5. La fase dell'oscillatore sinistro viene sempre 'resettata' a 0, mentre quella destra può essere cambiata tramite il *number box* in alto. Al variare della fase, un *bang* scrive la forma d'onda anche nei tre *array* che mostrano le singole onde e l'onda risultante. Quando impostate a 0.5 la fase dell'oscillatore destro, l'onda risultante è un segnale costante pari a 0. Quando invece la fase è prossima a 0 o a 1, l'onda risultante eccede i limiti di -1 e 1. Se provate a cambiare i valori della fase – dopo aver attivato il motore audio – vi accorgerete che il comportamento è ciclico: quando il valore è prossimo a 0.5, 1.5, 2.5 e così via le due onde si annullano; quando il valore è pari ad un numero intero – 0, 1, 2, 3 e così via – la somma è un segnale di ampiezza doppia rispetto alle singole onde. Questa ciclicità dipende dal fatto che un valore di fase pari a 1 reimposta l'onda e la fa ripartire dal valore di 360°, che corrispondere a un angolo di 0°. Lo stesso vale per ogni multiplo intero di 1.

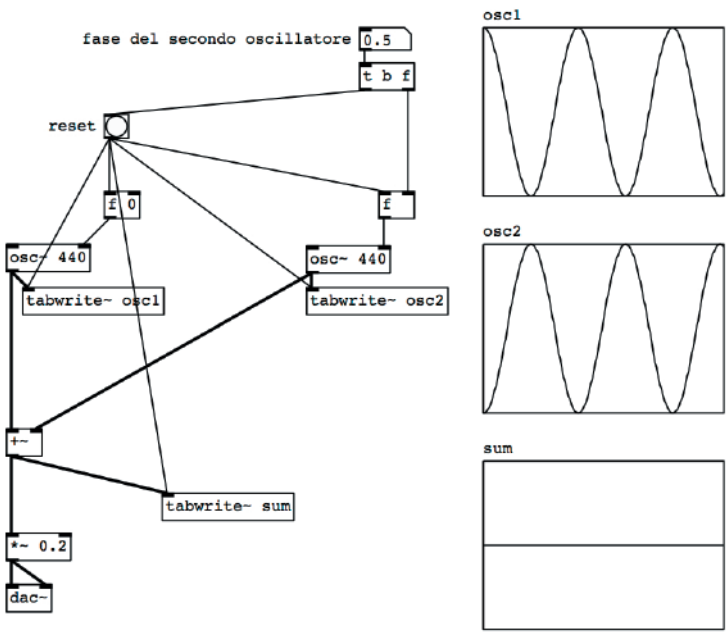


fig. 2.5: file 02_02_fase.pd

A differenza di `[osc~]`, l'oggetto `[vs.osc~]` consente di gestire la fase attraverso un segnale inviato all'ingresso destro. Vediamo come si può utilizzare, ricostruendo la *patch* di fig. 2.6.

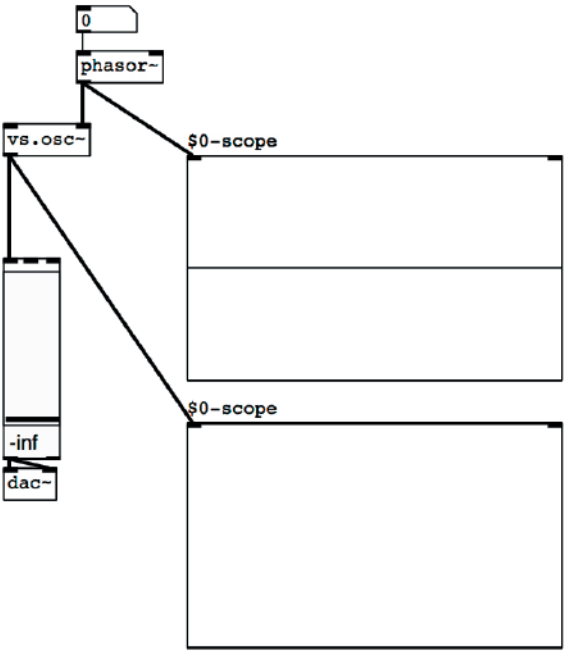


fig. 2.6: oscillatore guidato da un `[phasor~]`

Qui abbiamo un `[vs.osc~]` a 0 Hz (dal momento che non ha nessun argomento e nessun messaggio numerico all'ingresso di sinistra) la cui fase viene modificata da un `[phasor~]` che nell'immagine ha una frequenza di 0 Hz (è quindi fermo). Osservando gli oscilloscopi vediamo che `[phasor~]` genera un flusso di campioni di valore 0 (è quindi fermo all'inizio del suo ciclo), e `[vs.osc~]` genera un flusso di campioni di valore 1 (anche se non si vede perché corrisponde al lato superiore del rettangolo di `[vs.scope~]`); è anch'esso quindi fermo all'inizio del suo ciclo.
Adesso diamo al `[phasor~]` una frequenza maggiore di 0, ad esempio 400 Hz (vedi fig. 2.7).

Provate a ricostruire questa *patch*: l'oggetto `[phasor~]` controlla in modo continuo la fase di `[vs.osc~]` facendolo oscillare alla sua stessa frequenza e con la sua stessa fase. Come sappiamo dal par. 1.2, infatti, `[phasor~]` genera rampe che vanno da 0 a 1: queste rampe, applicate alla fase di `[vs.osc~]`, ne provocano l'oscillazione (e questo ci spiega il motivo per cui l'oggetto `[phasor~]` si chiama così: una delle sue funzioni principali è guidare la fase di un altro oggetto).

(...)

Il capitolo prosegue con:

Uso di array per gli oscillatori
Spettro fisso inarmonico

2.2 BATTIMENTI
Battimenti ritmici
Battimenti armonici

2.3 DISSOLVENZA INCROCIATA DI ARRAY: SINTESI VETTORIALE

2.4 SINTESI ADDITIVA A SPETTRO VARIABILE
vs.oscbank~: un banco di oscillatori
Il controllo mediante mascheratura

ATTIVITÀ
• Analisi di algoritmi, completamento di algoritmi, sostituzione di parti di algoritmi, correzione di algoritmi

VERIFICHE
• Compiti unitari di reverse engineering, Realizzazione di uno studio breve

SUSSIDI DIDATTICI
• Lista oggetti nativi Pd - Lista oggetti della libreria Virtual Sound - Lista messaggi per oggetti specifici

3T
GENERATORI DI RUMORE, FILTRI E SINTESI SOTTRATTIVA

CONTRATTO FORMATIVO

PREREQUISITI PER IL CAPITOLO

- Contenuti dei capp. 1 e 2 (teoria)

OBIETTIVI

Conoscenze

- Conoscere la teoria della sintesi sottrattiva
- Conoscere la teoria e l'uso dei parametri dei filtri principali
- Conoscere la differenza fra la teoria dei filtri ideali e la risposta di quelli digitali
- Conoscere la teoria e la risposta dei filtri FIR e IIR
- Conoscere l'uso di filtri disposti in serie o in parallelo
- Conoscere la teoria e l'uso degli equalizzatori grafici e parametrici
- Conoscere le applicazioni dei filtri a diversi tipi di segnale
- Conoscere le funzioni principali di un tipico sintetizzatore per sintesi sottrattiva

Abilità

- Saper individuare all'ascolto le caratteristiche di base di un filtraggio e saperle descrivere

CONTENUTI

- Sintesi sottrattiva
- Filtri passa-basso, passa-alto, passa-banda ed elimina-banda
- Filtri high shelving, low shelving, peak/notch
- Fattore Q
- Ordini dei filtri
- Filtri finite impulse response e infinite impulse response
- Equalizzatori grafici e parametrici
- Filtraggio di suoni provenienti da generatori di rumore, suoni campionati, impulsi

TEMPI - Cap. 3 (Teoria e Pratica) + Interludio B

Autodidatti

Per 300 ore globali di studio individuale (vol. I, teoria e pratica):

- ca. 110 ore

Corsi

Per un corso globale di 60 ore in classe + 120 di studio individuale (vol. I, teoria e pratica):

- ca. 18 ore frontali + 4 di feedback
- ca. 44 di studio individuale

ATTIVITÀ

- Esempi interattivi

VERIFICHE

- Test a risposte brevi
- Test con ascolto e analisi

SUSSIDI DIDATTICI

- Concetti di base - Glossario - Discografia - Un po' di storia

In questo capitolo parleremo dei *filtri*, argomento fondamentale nel campo del sound design e della musica elettronica, e di una tecnica di sintesi che ne fa largo uso: la sintesi sottrattiva.

3.1 SORGENTI PER LA SINTESI SOTTRATTIVA

La **sintesi sottrattiva** nasce dall'idea di poter creare un suono modificando, attraverso l'uso di filtri, l'ampiezza di alcune componenti di un altro suono. Lo scopo della maggior parte dei filtri digitali, infatti, è quello di alterare in qualche modo lo spettro di un suono. Un **filtro** quindi è un dispositivo che agisce prevalentemente su alcune frequenze contenute in un segnale, di solito attenuandole o enfatizzandole.[1]
Qualsiasi suono può essere filtrato. Attenzione, però! Non possiamo attenuare o enfatizzare componenti che non esistono nel suono originario, ad esempio non ha senso usare un filtro per enfatizzare i 50 Hz se stiamo filtrando la voce di un soprano, semplicemente perché tale frequenza non è presente nel suono originario.
I suoni originari utilizzati in genere nella sintesi sottrattiva sono ricchi dal punto di vista spettrale, e, come abbiamo detto, i filtri servono a modellare lo spettro di questi suoni e ottenere in uscita suoni diversi da quelli originari.
In questo paragrafo ci concentreremo sui suoni tipici utilizzati come sorgenti per la sintesi sottrattiva e per l'uso dei filtri in genere. Affronteremo le tecniche di filtraggio nei paragrafi successivi.

In generale nel lavoro in studio i filtri vengono utilizzati con diversi tipi di suoni:

- Suoni provenienti da generatori di rumore, da generatori di impulsi, da banchi di oscillatori, da altri generatori di segnale e da algoritmi di sintesi
- File audio/suoni campionati
- Suoni provenienti da fonti dal vivo in tempo reale (per esempio un suono proveniente dal microfono di un musicista che sta suonando un oboe)

GENERATORI DI RUMORE: RUMORE BIANCO E RUMORE ROSA

Uno dei suoni più utilizzati come sorgente per la sintesi sottrattiva è il **rumore bianco**, cioè un suono che contiene tutte le frequenze udibili e il cui spettro è essenzialmente piatto (pur essendo l'ampiezza delle singole frequenze distribuita casualmente).

[1] Oltre all'ampiezza un filtro può modificare la fase delle componenti di un suono.

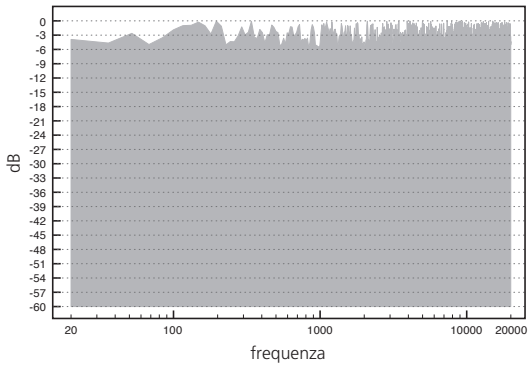


fig. 3.1: spettro del rumore bianco

È chiamato rumore bianco in analogia con l'ottica, dato che il colore bianco contiene tutti i colori dello spettro visibile. Si utilizza spesso il rumore bianco come suono originario perché, dal momento che contiene tutte le frequenze udibili, può essere utilmente filtrato da qualunque tipo di filtro a qualunque frequenza (vedi fig. 3.1).

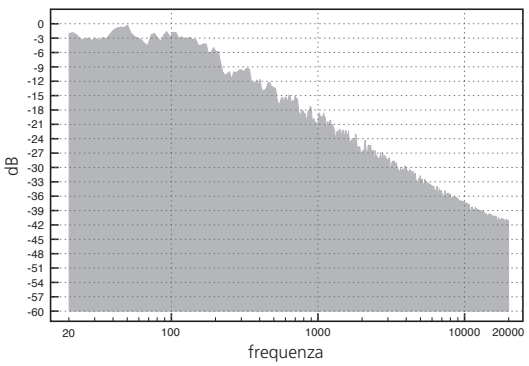


fig. 3.2: spettro del rumore rosa

Un altro tipo di rumore spesso utilizzato nella sintesi sottrattiva è il **rumore rosa**. Quest'ultimo, a differenza del rumore bianco, ha uno spettro la cui energia diminuisce all'aumentare della frequenza, più precisamente l'attenuazione è di 3 dB per ottava[2]; è anche chiamato generatore di rumore 1/f, per indicare che la sua energia spettrale è proporzionale al reciproco della frequenza. Viene spesso utilizzato, insieme ad un analizzatore di spettro, per testare e correggere la risposta in frequenza di impianti audio in relazione ad ambienti in cui si svolgono eventi musicali (fig. 3.2).

Nei sistemi digitali in genere il rumore bianco viene prodotto mediante generatori di numeri *random* (casuali): l'onda casuale che ne deriva contiene tutte le frequenze riproducibili dal sistema digitale usato. In realtà, i generatori di numeri *random* utilizzano procedure matematiche che non sono propriamente casuali: infatti generano serie che si ripetono dopo un certo numero di eventi. Questi generatori vengono perciò definiti **generatori pseudocasuali**.
Modificando alcuni parametri si possono generare tipi di rumore diversi. Il generatore di rumore bianco, ad esempio, genera campioni casuali alla frequenza di

[2] Un altro modo per definire la differenza tra rumore bianco e rumore rosa è questo: mentre lo spettro del rumore bianco ha la stessa energia per ogni frequenza, lo spettro del rumore rosa ha *la stessa energia per ogni ottava*. Dal momento che salendo verso l'acuto un'ottava occupa uno spazio di frequenza doppio dell'ottava precedente, la stessa energia totale viene distribuita in uno spazio sempre maggiore e questo determina l'attenuazione di 3 dB caratteristica del rumore rosa.

campionamento (ovvero, se la frequenza di campionamento del sistema è 48000 Hz, vengono generati 48000 valori casuali al secondo); è possibile però modificare la frequenza con cui questi numeri vengono generati: utilizzando una frequenza di generazione dei numeri pari a 5000 al secondo ad esempio, non avremo più un rumore bianco, ma un rumore con forte attenuazione sulle frequenze acute.

Quando la frequenza di generazione dei campioni è minore della frequenza di campionamento si crea il problema di come "riempire i vuoti" tra un campione e il successivo: un sistema DSP (vedi glossario cap. 1T) infatti deve sempre produrre un numero di campioni al secondo pari alla frequenza di campionamento. Ci sono vari modi per risolvere questo problema. In particolare possiamo fare tre esempi:

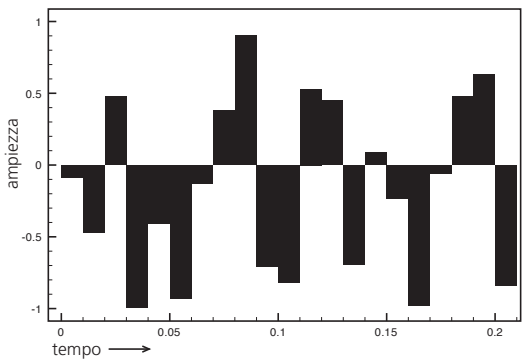


fig. 3.3: generazione di valori pseudo-casuali

• **Generatori di campioni pseudocasuali semplici**.
Generano valori casuali ad una frequenza data e mantengono il valore di ciascun campione fino alla generazione del valore successivo, creando un andamento a gradini. In fig. 3.3 vediamo il grafico relativo ad un generatore di rumore a 100 Hz: ogni valore casuale generato viene ripetuto nei campioni successivi per un periodo pari ad 1/100 di secondo, dopo di che si genera un nuovo valore. Se la frequenza di campionamento fosse 48000 Hz, ad esempio, ogni valore casuale verrebbe ripetuto per 48000/100 = 480 campioni.

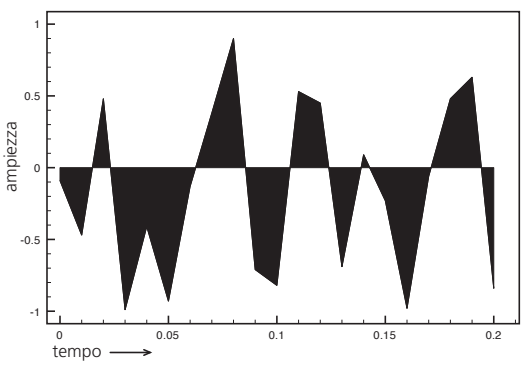


fig. 3.4 : generazione di valori pseudo-casuali con interpolazione lineare

• **Generatori di campioni pseudocasuali con interpolazione** fra un numero *random* e il successivo (vedi la sezione sull'interpolazione lineare nel cap. 2.1): come possiamo vedere in fig. 3.4 i campioni che si trovano fra i valori casuali prodotti dal generatore formano un segmento di retta che porta gradualmente da un valore all'altro.
L'interpolazione tra un valore e l'altro può essere lineare, come quella

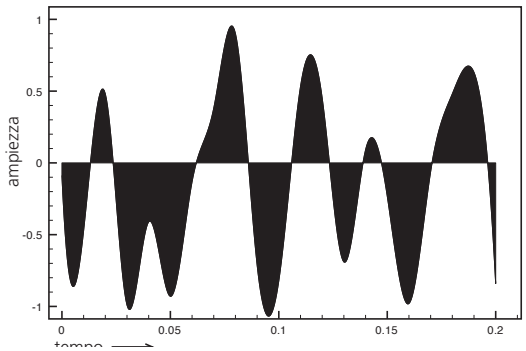


fig. 3.5: generazione di valori pseudo-casuali con interpolazione polinomiale

illustrata in figura, oppure *polinomiale*, realizzata cioè utilizzando delle funzioni polinomiali (che non approfondiremo in questa sede) che collegano i valori tramite delle curve anziché tramite delle rette (vedi fig. 3.5). Le interpolazioni polinomiali più usate nella computer music sono l'interpolazione *quadratica* (realizzata cioè con un polinomio di secondo grado) e quella *cubica* (polinomio di terzo grado): normalmente i linguaggi di programmazione per la sintesi e l'elaborazione del suono hanno già degli algoritmi pronti che realizzano queste interpolazioni.

- **Generatori di campioni pseudocasuali con filtro:** in questo tipo di sistema il segnale in uscita viene filtrato mediante un filtro passa-basso. Parleremo di questo tipo di generatore nella sezione dedicata ai filtri passa-basso.

ESEMPIO INTERATTIVO 3A • *GENERATORI DI RUMORE* - PRESET 1-4

OSCILLATORI E ALTRI GENERATORI DI SEGNALE

Nel paragrafo 1.2 abbiamo visto alcune forme d'onda "classiche" impiegate normalmente nei sintetizzatori, come l'onda quadra, l'onda a dente di sega e la triangolare; nel paragrafo 2.1, inoltre, abbiamo visto che tali forme d'onda, quando sono geometricamente perfette (cioè perfettamente quadrate, triangolari etc.) contengono un numero infinito di componenti. La presenza di infinite componenti, però, causa alcuni importanti problemi nella riproduzione del suono digitale: ciò è dovuto al fatto che una scheda audio non può riprodurre frequenze superiori alla metà della frequenza di campionamento[3] (approfondiremo questo argomento nel capitolo 5).
Quando si tenta di riprodurre digitalmente un suono in cui la frequenza delle componenti supera le capacità della scheda, si ottengono componenti indesiderate quasi sempre inarmoniche. Per evitare questo problema vengono spesso usati, nella musica digitale, gli **oscillatori limitati in banda**. Tali oscillatori, che riproducono le forme d'onda classiche, sono fatti in modo che le componenti non superino mai la metà della frequenza di campionamento. I suoni generati da questo

[3] È per questo motivo che la frequenza di una scheda audio è quasi sempre superiore al doppio della massima frequenza udibile dall'uomo.

tipo di oscillatori possono quindi essere un buon punto di partenza per ottenere sonorità particolari mediante l'uso di filtri, e sono largamente impiegati, infatti, nell'implementazione di sintetizzatori che operano mediante la sintesi sottrattiva. Nel par. 3.5 analizzeremo la struttura di un tipico sintetizzatore sottrattivo.
Naturalmente sarà possibile utilizzare per la sintesi sottrattiva anche suoni sintetici ricchi di parziali realizzati con le diverse tecniche (ad esempio le tecniche di sintesi non lineare o la sintesi per modelli fisici) di cui parleremo nei prossimi capitoli.

FILTRAGGIO DI SUONI CAMPIONATI

Uno degli utilizzi più comuni dei filtri e degli equalizzatori, al là della sintesi sottrattiva, è il filtraggio dei suoni campionati. A differenza del rumore bianco, che contiene tutte le frequenze alla stessa ampiezza, un suono campionato conterrà un numero limitato di frequenze e rapporti d'ampiezza fra le componenti che possono variare a seconda del suono considerato.
È necessario perciò, prima di effettuare un filtraggio, essere coscienti del *range* frequenziale del suono da elaborare. Infatti, come già accennato in precedenza, *possiamo attenuare o esaltare solo frequenze che siano già contenute nel file di partenza*. Ciò vale anche per i suoni che giungono al nostro computer da una fonte dal vivo.

(...)

Il capitolo prosegue con:

3P
GENERATORI DI RUMORE, FILTRI E SINTESI SOTTRATTIVA

CONTRATTO FORMATIVO

PREREQUISITI PER IL CAPITOLO

- Contenuti dei capp. 1 e 2 (teoria e pratica), cap.3 (teoria), Interludio A

OBIETTIVI

Abilità

- Saper generare e controllare diversi tipi di segnali complessi per la sintesi sottrattiva (rumore bianco, rumore rosa, impulsi etc.)
- Saper costruire algoritmi con filtri passa-basso, passa-alto, passa-banda, elimina-banda, filtri shelving e filtri risonanti, e controllarne, fra i vari parametri, anche il Q e l'ordine dei filtri.
- Saper costruire filtri FIR o non ricorsivi e filtri IIR o ricorsivi
- Saper costruire semplici sintetizzatori in sintesi sottrattiva
- Saper scrivere algoritmi con collegamenti in serie e in parallelo dei filtri
- Saper costruire equalizzatori grafici e parametrici

Competenze

- Saper analizzare un breve studio sonoro basato sulle tecniche di sintesi sottrattiva e memorizzarlo su file audio.

CONTENUTI

- Sorgenti per la sintesi sottrattiva
- Filtri passa-basso, passa-alto, passa-banda, elimina-banda, filtri shelving e filtri risonanti
- Il Quality Factor e l'ordine dei filtri
- Filtri FIR e IIR
- Il collegamento in serie e in parallelo dei filtri
- Gli equalizzatori grafici e parametrici

TEMPI - Cap. 3 (Teoria e Pratica) + Interludio B

Autodidatti

Per 300 ore globali di studio individuale (vol. I, teoria e pratica):

- ca. 110 ore

Corsi

Per un corso globale di 60 ore in classe + 120 di studio individuale (vol. I, teoria e pratica):

- ca. 18 ore frontali + 4 di feedback
- ca. 44 di studio individuale

ATTIVITÀ

- Attività al computer: sostituzione di parti di algoritmi, correzione, completamento e analisi di algoritmi, costruzione di nuovi algoritmi

VERIFICHE

- Realizzazione di uno studio breve - Compiti unitari di Reverse Engineering

SUSSIDI DIDATTICI

- Lista oggetti nativi Pd - Lista oggetti della libreria virtual sound - Lista messaggi per oggetti specifici - Glossario

3.1 SORGENTI PER LA SINTESI SOTTRATTIVA

Come sappiamo dal paragrafo 3.1 della teoria, lo scopo di un filtro è generalmente quello di modificare in qualche modo lo spettro di un segnale. Introduciamo quindi per prima cosa un oggetto della libreria *Virtual Sound* che ci serve per visualizzare lo spettro: `[vs.spectroscope~]`. Per creare l'oggetto è sufficiente scriverne il nome all'interno di un *object box* generico.

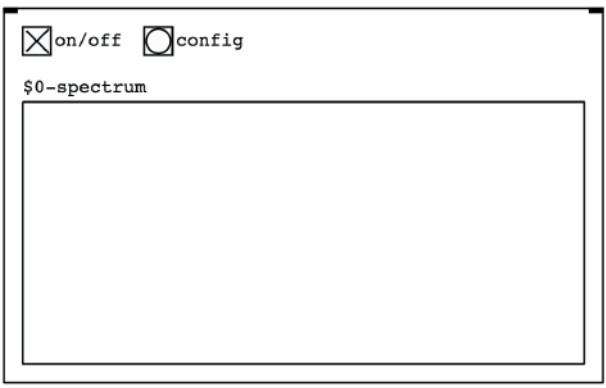


fig. 3.1: l'oggetto `[vs.spectroscope~]`

Aprite ora il file **03_01_spectroscope.pd** (fig. 3.2).

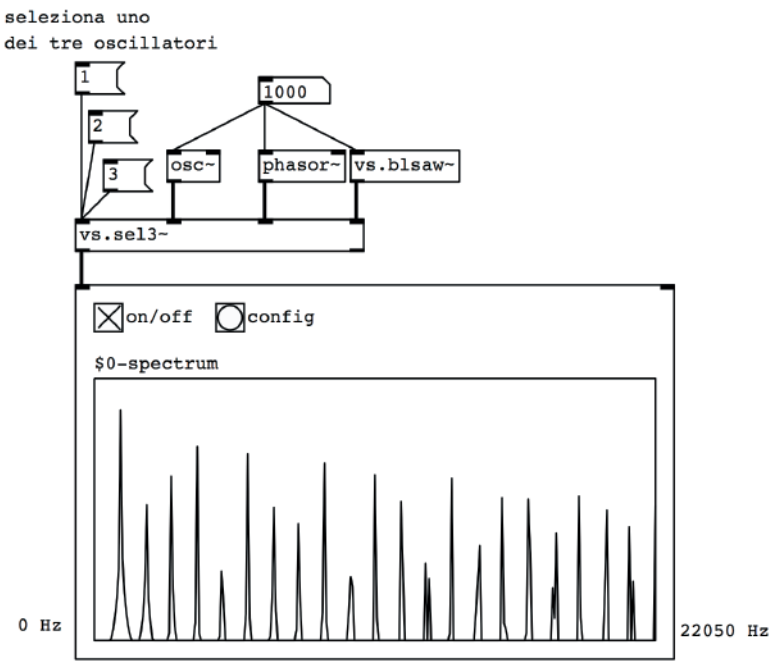


fig. 3.2: file 03_01_spectroscope.pd

Qui abbiamo collegato allo spettroscopio un oggetto `[vs.sel3~]` che ci permette di "smistare" tre oscillatori con forma d'onda sinusoidale (`[osc~]`), a dente di sega non limitata in banda (`[phasor~]`) e a dente di sega limitata in banda (`[vs.blsaw~]`)[1]. All'ingresso di sinistra di `[vs.sel3~]` sono collegati tre *message box* che servono a selezionare uno dei tre oscillatori: impostando la frequenza nel *number box* e selezionando i tre oscillatori possiamo vedere gli spettri relativi. Notate che la sinusoide contiene una sola componente, mentre all'oggetto `[phasor~]`, che genera una forma d'onda non limitata in banda, corrisponde lo spettro più ricco[2].

Fate alcune prove, cambiando la frequenza e selezionando le diverse forme d'onda e osservate le immagini che si producono nello spettroscopio. Come abbiamo detto quello che viene visualizzato è lo spettro del suono in ingresso: le diverse componenti del suono sono distribuite da sinistra a destra e vengono visualizzate le frequenze da 0 Hz a 22050 Hz.

Provate ad aggiungere l'oggetto `[vs.spectroscope~]` alle *patch* che avete già realizzato, in modo da familiarizzarvi con la relazione che c'è tra un suono e il suo contenuto spettrale: potete aggiungere l'oggetto anche alle *patch* dei capitoli precedenti, provate ad esempio con **01_13_audiofile.pd** (collegando lo spettroscopio all'uscita di sinistra dell'oggetto `[vs.splayer~]` oppure con **IA_06_random_walk.pd** (collegandolo all'uscita di `[pd monosynth]`): in quest'ultima *patch* riuscite a cogliere la relazione tra la frequenza del suono e la forma dello spettro?

Passiamo adesso al rumore bianco, che viene generato in Pd dall'oggetto `[noise~]` (vedi fig. 3.3).

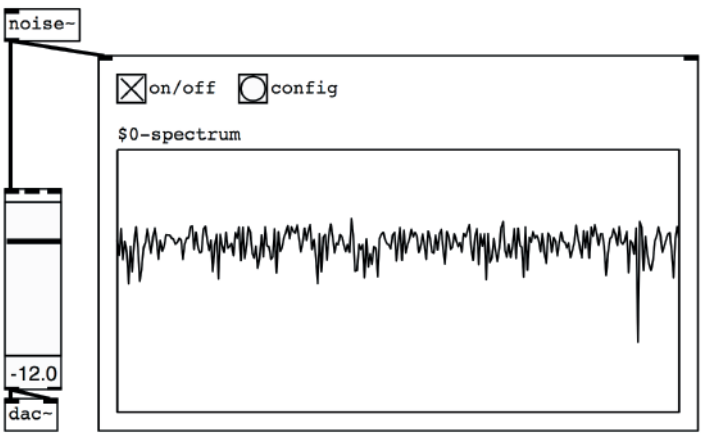


fig. 3.3: generatore di rumore bianco

[1] Vedi paragrafo 1.2

[2] L'oggetto `[vs.spectroscope~]` contiene un algoritmo di analisi spettrale denominato *Fast Fourier Transform*, sui cui torneremo con maggiori dettagli in un capitolo successivo

Nella *patch* dell'immagine (che vi invitiamo a ricostruire) abbiamo collegato il generatore di rumore all'oggetto `[vs.spectroscope~]` tramite il quale possiamo vedere che lo spettro del rumore bianco contiene energia a tutte le frequenze. A differenza degli altri generatori di suono che abbiamo incontrato finora, il generatore di rumore bianco non ha bisogno di alcun parametro: la sua funzione infatti è quella di generare un segnale costituito da valori casuali compresi tra -1 e 1 alla frequenza di campionamento (vedi par. 3.1 della teoria). Un altro tipo di generatore di rumore che abbiamo a disposizione nella libreria *Virtual Sound* è l'oggetto `[vs.pink~]` che genera rumore rosa (fig. 3.4).

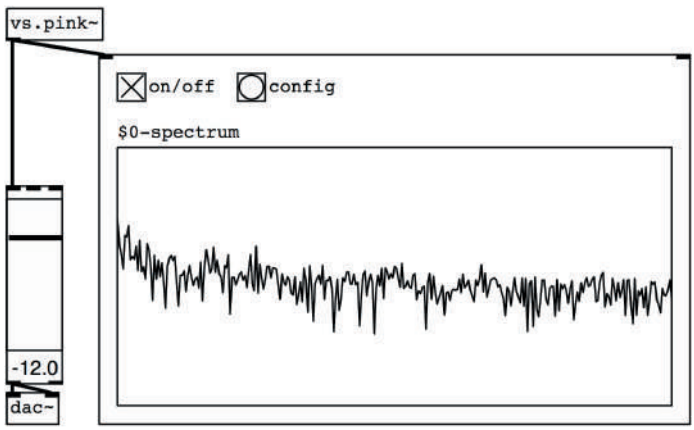


fig. 3.4: rumore rosa

Notate che, a differenza del rumore bianco, lo spettro del rumore rosa subisce un'attenuazione man mano che si procede verso le frequenze più alte e l'attenuazione (come sappiamo dal par. 3.1 della teoria) è di 3 dB per ottava. Ricostruite la *patch* e ascoltate bene la differenza tra il rumore rosa e il rumore bianco: quale dei due suoni vi sembra più gradevole (o meno sgradevole) e perché?
Aggiungete alle due *patch* appena realizzate un oscilloscopio (`[vs.scope~]`) e osservate le differenze tra la forma d'onda del rumore bianco e quella del rumore rosa. In fig. 3.5 vediamo le due forme d'onda affiancate.

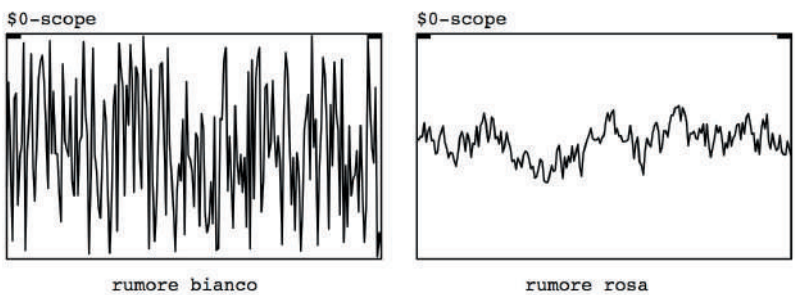


fig. 3.5: forma d'onda del rumore bianco e del rumore rosa

Senza entrare nei dettagli tecnici possiamo osservare che mentre il rumore bianco è, come sappiamo, una successione di valori casuali, il rumore rosa viene generato con un algoritmo più complesso, in cui un campione, che è sempre "casuale", non può però discostarsi eccessivamente dal precedente, e questo genera l'andamento "serpeggiante" della forma d'onda che vediamo in figura. Il comportamento delle due forme d'onda corrisponde al loro contenuto spettrale: maggiore infatti è la differenza tra un campione e il successivo e maggiore è l'energia alle frequenze più alte[3], e come abbiamo detto il rumore bianco ha maggiore energia alle frequenze alte rispetto al rumore rosa.

Un altro generatore interessante è `[vs.rand1~]`[4], che genera campioni casuali ad una frequenza regolabile a piacere e li collega tramite linee, o meglio segmenti di retta (fig. 3.6). A differenza di `[noise~]` e `[vs.pink~]`, che generano un nuovo campione casuale ad ogni ciclo del "motore" DSP (cioè generano ogni secondo un numero di campioni casuali pari alla frequenza di campionamento, ad es. 44100), con `[vs.rand1~]` è possibile stabilire la frequenza con cui i campioni casuali verranno generati, e il passaggio tra un campione e il successivo avviene gradualmente, tramite un'interpolazione lineare.

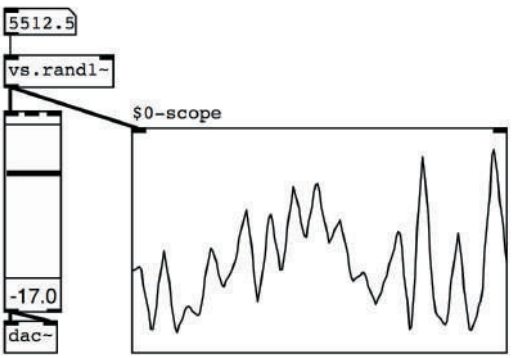


fig. 3.6: l'oggetto `[vs.rand1~]`

Questo generatore produce uno spettro che varia, come è ovvio, in relazione alla frequenza impostata, e che presenta una banda di frequenza principale che va da 0 Hz alla frequenza impostata, seguita da bande secondarie progressivamente attenuate la cui larghezza è pari alla frequenza impostata, vedi fig. 3.7.

[3] Per comprendere questa affermazione osserviamo che la forma d'onda di un suono acuto oscilla velocemente, mentre quella di un suono grave oscilla lentamente. Nel primo caso, a parità di ampiezza, la differenza di valore tra due campioni successivi è mediamente maggiore che nel secondo caso.

[4] Il prefisso vs indica che l'oggetto fa parte della libreria *Virtual Sound*, come tutti gli altri oggetti preceduti da questo prefisso.

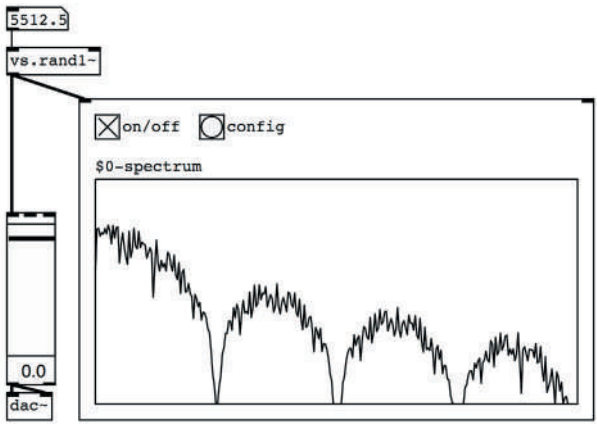


fig. 3.7: spettro generato dall'oggetto `[vs.rand1~]`

Nell'esempio vediamo che la frequenza di `[vs.rand1~]` è di 5512.5 Hz (un quarto della massima frequenza visualizzabile nello spettroscopio in figura), e la prima banda va da 0 Hz a 5512.5 Hz. A questa seguono tre bande secondarie, progressivamente attenuate, tutte larghe 5512.5 Hz. Variando la frequenza di `[vs.rand1~]` si varia la larghezza delle bande e il loro numero: ad esempio se raddoppiamo la frequenza e la portiamo a 11025 Hz otteniamo due bande larghe appunto 11025 Hz.

Un altro generatore di rumore è `[vs.rand0~]` (l'ultimo carattere prima della tilde è uno zero) che genera campioni casuali ad una frequenza data come `[vs.rand1~]`, ma non effettua alcuna interpolazione e mantiene il valore di ciascun campione fino alla generazione del campione successivo, creando un andamento a gradini.

Lo spettro è diviso in bande come lo spettro di `[vs.rand1~]` in fig. 3.7, ma l'attenuazione delle bande secondarie è molto minore a causa del brusco passaggio fra un campione e il successivo (vedi fig. 3.8).

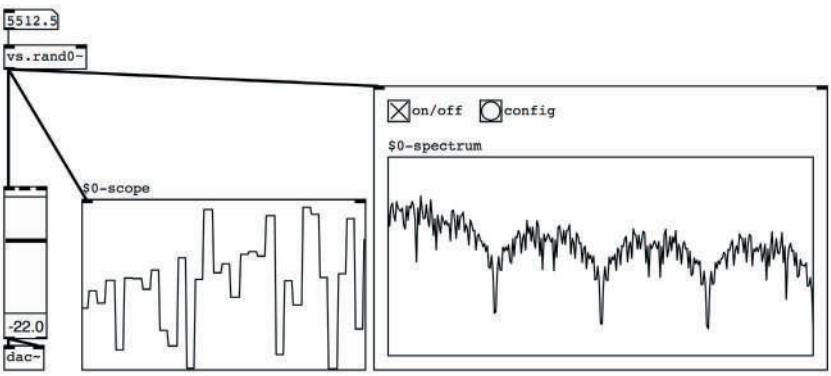


fig. 3.8: l'oggetto `[vs.rand0~]`

Interludio B

ALTRI ELEMENTI DI PROGRAMMAZIONE CON PURE DATA

CONTRATTO FORMATIVO

PREREQUISITI PER IL CAPITOLO

- Contenuti dei capp. 1, 2 e 3 (teoria e pratica), Interludio A

OBIETTIVI

Abilità

- Saper utilizzare diversi oggetti e segnali MIDI semplici
- Saper utilizzare operazioni ricorsive e conversioni in Pd
- Saper costruire un arpeggiatore, anche con uso di intervalli probabilistici
- Saper smistare segnali e messaggi selezionando ingressi e uscite
- Saper confrontare valori e analizzarne la relazione
- Saper scomporre liste di dati
- Saper costruire sequenze ripetitive, tramite loop di dati
- Saper generare liste casuali per la simulazione di corpi risonanti
- Saper costruire uno Shepard Tone, o glissando infinito

CONTENUTI

- Uso di base del protocollo MIDI
- Operazioni ricorsive e sequenze ripetitive
- Arpeggiatori e intervalli probabilistici
- Confronto di valori, conversioni e smistamento di segnali e messaggi
- Scomposizione di liste e generazione di liste casuali
- Shepard Tone

TEMPI - Cap.3 (teoria e pratica) + Interludio B

Autodidatti

Per 300 ore globali di studio individuale (vol. I, teoria e pratica):

- ca. 110 ore

Corsi

Per un corso globale di 60 ore in classe + 120 di studio individuale (vol. I, teoria e pratica):

- ca. 18 ore frontali + 4 di feedback
- ca. 44 di studio individuale

ATTIVITÀ

Attività al computer:

- sostituzione di parti di algoritmi, correzione, completamento e analisi di algoritmi, costruzione di nuovi algoritmi

VERIFICHE

- Compiti unitari di Reverse Engineering

SUSSIDI DIDATTICI

- Lista oggetti nativi di Pd - Lista oggetti della libreria virtual sound - Glossario

IB.1 CENNI SUL MIDI

Il MIDI è un sistema di comunicazione tra computer e strumenti musicali elettronici e/o digitali: tramite questo sistema è possibile, ad esempio, collegare (con un apposito cavetto MIDI) un computer ad un sintetizzatore e far sì che quest'ultimo venga "suonato" dal computer. In altre parole grazie al MIDI il computer può dire al sintetizzatore, tra le altre cose, quali note suonare, con che intensità, con che durata etc.
Gli strumenti musicali digitali, inoltre, possono anche essere "virtuali", cioè possono essere delle applicazioni residenti nel nostro computer che simulano il comportamento di strumenti digitali reali: anche con gli strumenti virtuali è possibile comunicare via MIDI, realizzando una connessione, in questo caso virtuale (cioè non con un cavo fisico), tra il programma che invia i comandi MIDI (ad esempio Pd) e il programma (lo strumento virtuale) che li riceve.
Come vedremo successivamente in modo più approfondito, Pd ha diversi oggetti che sfruttano il protocollo MIDI e dal momento che nel seguito di questo Interludio ne verranno utilizzati alcuni, è bene configurare adeguatamente il vostro sistema affinché Pd possa comunicare con i dispositivi MIDI presenti nel computer. Per una trattazione più dettagliata di questo argomento potete scaricare il documento pdf relativo (**MESD_MIDI_config.pdf**). In linea di massima per collegare Pd ad un dispositivo MIDI virtuale su un sistema *Windows* è sufficiente aprire Pd, e dal menu *Media/MIDI Settings* impostare in corrispondenza dell'uscita *Output Device 1* la voce *Microsoft GS Wavetable*[1].

Su Mac invece:

1. aprite la finestra di configurazione MIDI Audio (accessibile nella cartella */Application/Utility* o dal *Launchpad*) e verificate che il driver IAC sia attivo;
2. aprite Pd e dal menu *Media/MIDI settings* selezionate *IAC Driver* dal menu *Output Device 1*;
3. aprite un software che possa ospitare strumenti virtuali (*Garageband* o simili).

Una volta configurato il MIDI aprite il file **IB_01_note_MIDI.pd**; in fig. IB.1 vediamo la parte superiore della *patch*.

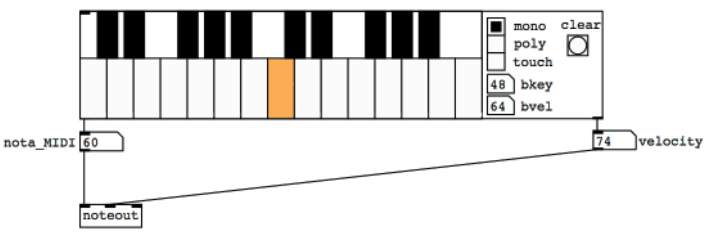


fig. IB.1: file IB_01_note_MIDI.pd, parte superiore

[1] La denominazione potrebbe cambiare fra versioni diverse del sistema *Windows*, ma non vi spaventate, riconoscerete comunque il dispositivo!

Abbiamo collegato l'oggetto `[vs.midikeyboard]` (la tastiera musicale) a dei *number box* e, tramite questi, all'oggetto `[noteout]`. Come possiamo vedere, facendo clic su un tasto di `[vs.midikeyboard]` avremo all'uscita di sinistra il valore MIDI della nota selezionata (cfr. anche il par. 1.4) e all'uscita di destra il valore di *velocity*, ovvero l'intensità della nota: facendo clic nella parte alta di un tasto di `[vs.midikeyboard]` si ottengono valori alti di *velocity*, facendo clic sulla parte bassa si ottengono valori bassi. La *velocity* può variare tra 1 e 127. I valori di nota MIDI e *velocity* vengono inviati agli ingressi di sinistra e centrale dell'oggetto `[noteout]`: quest'ultimo invia il comando relativo all'esecuzione della nota allo strumento (reale o virtuale) a cui è collegato.[2]

Nel protocollo MIDI questo comando si definisce *note-on*. Se fate clic su un tasto del `[vs.midikeyboard]` (abbastanza in alto in modo da ottenere una *velocity* alta, superiore a 90) e se avete configurato il MIDI in modo corretto, dovreste sentire un suono di pianoforte[3]. Questo suono non proviene da Pd, ma dallo strumento virtuale contenuto nel vostro computer che solitamente è impostato sul suono del pianoforte. Se provate a suonare più note vi accorgerete che, per ogni coppia nota-*velocity* che l'oggetto `[noteout]` riceve, viene suonata una nuova nota, ma le precedenti non vengono interrotte: è come se i tasti fossero "incantati". Il problema è che, tramite `[noteout]`, stiamo dicendo allo strumento virtuale quando iniziare a suonare una nota, ma non gli diciamo quando interromperla!
Per interrompere una nota dobbiamo inviare nuovamente il valore MIDI relativo associato ad una *velocity* pari a 0. Il valore di *velocity* 0 corrisponde al comando *note-off*, ovvero "solleva il dito dal tasto".
Per poter "spegnere" una nota MIDI tramite il `[vs.midikeyboard]` dobbiamo cambiare il modo con cui quest'ultimo gestisce i messaggi di nota MIDI: nella parte destra dell'oggetto selezionate il modo denominato *poly* (prima eravate in modalità *mono*).
Adesso la prima volta che fate clic su un tasto del `[vs.midikeyboard]` verrà suonata una nota con la *velocity* corrispondente, un secondo clic sullo stesso tasto invierà nuovamente la nota, ma con *velocity* pari a 0, facendo terminare il suono: provate! Questo modo è definito *poly*, cioè polifonico perché, a differenza del *mono*, cioè monofonico, ci permette di tenere attive più note contemporaneamente.
In figura IB.2 vediamo la parte inferiore del file **IB_01_note_MIDI.pd**.

Qui abbiamo collegato l'oggetto `[vs.midikeyboard]` (in modalità *mono*) all'oggetto `[makenote]`: quest'ultimo ogni volta che riceve un comando MIDI *note-on* genera il corrispondente comando MIDI *note-off* dopo un intervallo di tempo stabilito. L'oggetto ha tre ingressi, rispettivamente per il valore di nota MIDI, la *velocity* e la durata in millisecondi (ovvero il tempo che deve passare tra

[2] L'ingresso di destra dell'oggetto `[noteout]` serve per impostare il canale MIDI, che al momento non ci serve: maggiori dettagli verranno successivamente.
[3] Se non sentite niente significa che non avete configurato correttamente il sistema e quindi dovete tornare all'inizio del capitolo oppure consultare il pdf **MESD_MIDI_config.pdf**, che potete scaricare dalle pagine di supporto per questo testo (http://www.**********).

un *note-on* e il successivo *note-off*), e due uscite, per il valore di nota MIDI e la *velocity*.

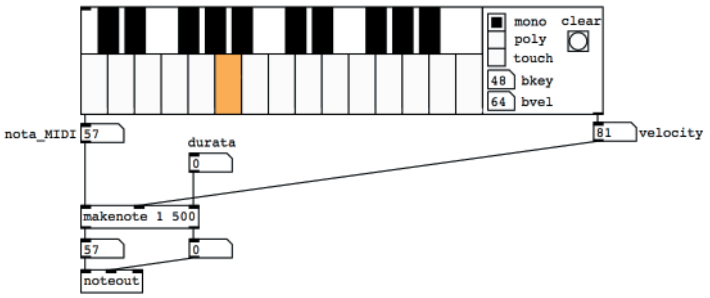


fig. IB.2: file IB_01_note_MIDI.pd, parte inferiore

Gli argomenti sono due: la *velocity* e la durata in millisecondi; nella *patch* abbiamo quindi una *velocity* pari a 1 e una durata di 500 millisecondi (mezzo secondo). Quando l'oggetto riceve un *note-on* lo invia direttamente alle uscite, dopo di che attende la durata prescritta (nel nostro caso 500 millisecondi) dopo di che invia il *note-off*. Notate che la *velocity* che inviamo tramite il `[vs.midikeyboard]` (nel caso in figura il valore 81) annulla e sostituisce il valore 1 che avevamo scritto come argomento: quest'ultimo ci è servito infatti solo per permetterci di scrivere il secondo argomento, cioè la durata, che in quanto "secondo argomento" deve essere per forza preceduto dal primo!
Anche la durata può essere modificata, inviando il nuovo valore all'ingresso di destra, che sostituirà il valore che abbiamo messo come secondo argomento.
Provate a suonare alcune note e a cambiare la durata nell'oggetto `[makenote]`: osservate come dalla sua seconda uscita venga generato prima un valore di *velocity* identico a quello generato dal `[vs.midikeyboard]` e, dopo il tempo stabilito, il valore 0.
Ora aggiungiamo un sommatore alla parte inferiore della *patch* nel modo illustrato in fig. IB.3.

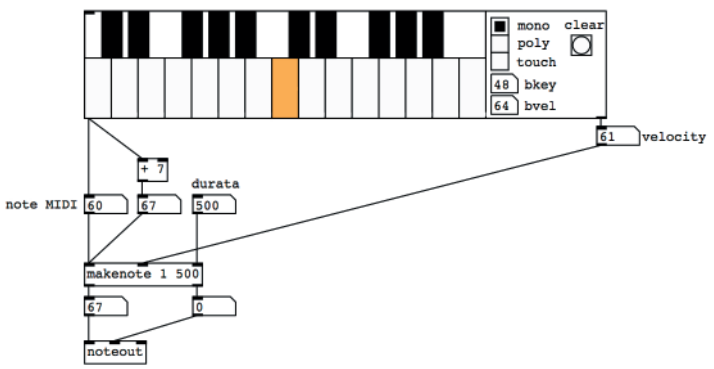


fig. IB.3: trasposizione MIDI

Questa *patch* è simile a quella del file **IA_01_trasposizione.pd** che abbiamo visto al primo paragrafo dell'interludio A. Anche in questo caso per ogni tasto premuto nell'oggetto `[vs.midikeyboard]` vengono generate due note a distanza di 7 semitoni, cioè di una quinta. Ogni volta che facciamo clic su un tasto di `[vs.midikeyboard]`, infatti, il valore di nota MIDI corrispondente viene inviato all'oggetto `[makenote]` e contemporaneamente ad un sommatore che aggiunge il valore 7 alla nota prima di inviarla a sua volta al `[make-note]`. Notate che per ottenere queste coppie di note non abbiamo avuto bisogno di sdoppiare anche il valore della *velocity*, né quello della durata: questi ultimi infatti corrispondono agli ingressi "freddi" dell'oggetto `[makenote]` e aggiornano soltanto le variabili interne dell'oggetto. Il contenuto di queste variabili interne viene riutilizzato ogni volta che un nuovo valore arriva all'ingresso "caldo": nel caso in figura, ad esempio, entrambe le note (DO e SOL centrali) hanno una *velocity* pari a 61 e una durata di 500 millisecondi.
Qui possiamo vedere che la regola secondo cui l'ingresso "caldo" di un oggetto è quello più a sinistra è il logico complemento della regola dell'ordine di esecuzione da destra a sinistra: i primi messaggi ad essere trasmessi sono quelli più a destra, che raggiungono gli ingressi "freddi" e aggiornano le variabili interne di un oggetto (ad esempio la *velocity* in `[makenote]`), mentre gli ultimi sono quelli più a sinistra, che raggiungono un ingresso "caldo" e provocano un *output* dopo che tutte le variabili interne sono state aggiornate.

(...)

Il capitolo prosegue con:

ATTIVITÀ
- Analisi di algoritmi, completamento di algoritmi, sostituzione di parti di algoritmi, correzione di algoritmi,

VERIFICHE
- Compiti unitari di reverse engineering

SUSSIDI DIDATTICI
- Lista oggetti nativi Pd - Lista oggetti della libreria Virtual Sound - Glossario

4T
SEGNALI DI CONTROLLO

CONTRATTO FORMATIVO

PREREQUISITI PER IL CAPITOLO

- Contenuti dei capp. 1, 2, 3 (teoria)

OBIETTIVI

Conoscenze

- Conoscere la teoria e l'uso dei parametri degli oscillatori a bassa frequenza (LFO)
- Conoscere l'uso del DC Offset applicato agli LFO
- Conoscere l'uso della modulazione di frequenza per il vibrato
- Conoscere l'uso della modulazione di ampiezza per il tremolo
- Conoscere l'uso della modulazione del duty cycle (Pulse Width Modulation)
- Conoscere l'uso degli LFO per generare segnali di controllo per i filtri
- Conoscere l'uso di generatori di segnali pseudocasuali come LFO di controllo
- Conoscere l'uso degli oscillatori di controllo per lo spostamento del suono nei sistemi stereofonici e multicanale

Abilità

- Saper individuare all'ascolto le modifiche dei parametri base di un LFO e saperle descrivere

CONTENUTI

- Oscillatori a bassa frequenza: depth, rate e delay
- Gestione dei parametri degli LFO e uso del DC Offset
- Gestione del vibrato, del tremolo e del PWM mediante gli LFO
- Gestione dei parametri dei filtri mediante LFO
- Spostamento del suono nei sistemi stereo e multicanale
- Oscillatori di controllo modulati da altri LFO

TEMPI - Cap. 4 (Teoria e Pratica)

Autodidatti

Per 300 ore globali di studio individuale (vol. I, teoria e pratica):

- ca. 30 ore

Corsi

Per un corso globale di 60 ore in classe + 120 di studio individuale (vol. I, teoria e pratica):

- ca. 5 ore frontali + 1 di *feedback*
- ca. 12 di studio individuale

ATTIVITÀ

- Esempi interattivi

VERIFICHE

- Test a risposte brevi
- Test con ascolto e analisi

SUSSIDI DIDATTICI

- Concetti di base - Glossario

4.1 SEGNALI DI CONTROLLO: IL PANNING STEREOFONICO

Come abbiamo visto nel Cap. 1 è possibile variare i parametri di un suono (ad esempio la frequenza o l'ampiezza) tramite inviluppi che descrivono l'andamento nel tempo dei parametri in questione. I segnali (come quelli che controllano gli inviluppi), che non servono a generare un suono ma a variarne le caratteristiche, si definiscono **segnali di controllo**. Finora abbiamo usato, come segnali di controllo, soltanto segmenti di retta o di esponenziale. Questa tecnica si rivela efficace quando dobbiamo descrivere il cambiamento di un parametro attraverso pochi valori: ad esempio, per un inviluppo ADSR abbiamo bisogno di 4 segmenti, mentre il glissando di una nota può essere realizzato con una singola curva esponenziale. I parametri di un suono possono però variare anche in modo più complesso. Pensiamo ad esempio al vibrato di uno strumento ad arco: si tratta di un'oscillazione continua della frequenza della nota intorno ad un'altezza centrale. Per simulare questa vibrazione avremmo bisogno di decine o centinaia di segmenti, ma sarebbe evidentemente poco pratico e molto faticoso. Possiamo invece utilizzare un **oscillatore di controllo**, cioè un oscillatore che non serve a produrre suoni ma semplicemente valori che variano da un minimo a un massimo con una certa velocità e che vengono generalmente assegnati ai parametri degli *oscillatori audio*. Tali valori possono essere assegnati anche a parametri di altri algoritmi di sintesi ed elaborazione del suono.
È importante osservare che gli oscillatori di controllo sono *oscillatori in bassa frequenza* (**LFO, Low Frequency Oscillators**): essi cioè oscillano con frequenze generalmente inferiori a 30 Hz, e producono valori di controllo in continuo mutamento, che seguono la forma d'onda dell'oscillatore stesso. Ogni ampiezza istantanea dell'onda generata dall'oscillatore di controllo corrisponde ad un valore numerico che viene poi applicato ai parametri audio che vogliamo.
Facciamo un esempio: in figura 4.1 vediamo un LFO che controlla la posizione del suono nello spazio e genera una sinusoide che oscilla fra MIN (valore minimo) e MAX (valore massimo).

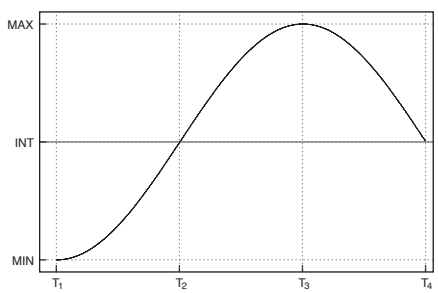


fig. 4.1: LFO che controlla la posizione del suono nello spazio

I valori minimo e massimo si riferiscono ai valori dell'*ampiezza* dell'oscillatore di controllo (di solito definita come **depth**, che in italiano significa profondità), la velocità dell'oscillazione dipende invece dalla *frequenza* dell'oscillatore di controllo (di solito definita come **rate**, che in italiano significa velocità, o frequenza).
I valori delle ampiezze istantanee della sinusoide generata da un LFO (o oscillatore di controllo) vengono usati come moltiplicatori delle ampiezze dei due canali in uscita di un oscillatore audio: quando la sinusoide dell'oscillatore di controllo raggiungerà il valore minimo MIN avremo il suono dell'oscillatore audio completamente a

sinistra, quando l'oscillatore di controllo raggiungerà il valore massimo MAX avremo il suono dell'oscillatore audio completamente a destra, quando la sinusoide dell'oscillatore di controllo si troverà in corrispondenza del valore intermedio (INT) le ampiezze del suono sul canale sinistro e destro saranno uguali e di conseguenza percepiremo il suono al centro.
È ovviamente possibile anche utilizzare altre forme d'onda (triangolare, casuale etc.) per variare questi parametri di controllo; ad esempio, se usiamo un'onda quadra lo spostamento da destra a sinistra e viceversa non avverrà in modo continuo, come con la sinusoide, ma in modo alternato (MIN-MAX-MIN-MAX etc.).

ESEMPIO INTERATTIVO 4A • *Panning mediante LFO con diverse forme d'onda*

La velocità (cioè il *rate*) con cui tali valori oscillano dipende dalla frequenza che assegniamo all'oscillatore di controllo. Se utilizzassimo la frequenza 1 avremmo un'oscillazione fra MAX e MIN e ritorno una volta al secondo, se applicassimo una frequenza pari a .2 avremmo un'oscillazione completa ogni 5 secondi. E se utilizzassimo la frequenza 220? L'oscillazione sarebbe troppo veloce per poter percepire lo spostamento da destra a sinistra e ritorno (220 volte al secondo!); inoltre questa frequenza rientrerebbe nel campo audio e ciò aggiungerebbe nuove componenti allo spettro del suono risultante, come vedremo nel capitolo 10 nel paragrafo dedicato alla modulazione di ampiezza.

ESEMPIO INTERATTIVO 4B • *Panning mediante LFO sinusoidale a diverse frequenze*

Con gli oscillatori di controllo possiamo, in modi analoghi a quello descritto sopra, controllare ampiezza (*depth*) e velocità (*rate*) di un vibrato, di un tremolo, della variazione dei parametri di un filtro, come vedremo nei prossimi paragrafi.

(...)

Il capitolo prosegue con:

4P
SEGNALI DI CONTROLLO

CONTRATTO FORMATIVO

PREREQUISITI PER IL CAPITOLO

- Contenuti dei capp. 1, 2, 3 (teoria e pratica), cap.4 (teoria), Interludio A e B

OBIETTIVI

Abilità

- Saper far oscillare un suono nello spazio stereofonico
- Saper realizzare effetti di vibrato
- Saper simulare strumenti controllati in frequenza, come il theremin
- Saper realizzare effetti di tremolo
- Saper realizzare algoritmi di Pulse Width Modulation
- Saper variare in modo oscillante la frequenza di taglio, la frequenza centrale e il Q dei filtri mediante segnali di controllo
- Saper utilizzare segnali di controllo pseudocasuali
- Far ruotare (tramite un segnale di controllo) il suono a quattro o più canali

Competenze

- Saper realizzare un breve studio sonoro basato sulle tecniche di controllo dei parametri mediante LFO

CONTENUTI

- Oscillatori a bassa frequenza: depth, rate e delay
- Gestione dei parametri degli LFO e uso del DC Offset
- Gestione del vibrato, del tremolo e del PWM mediante gli LFO
- Gestione dei parametri dei filtri mediante LFO
- Segnali di controllo pseudocasuali
- Spostamento del suono nei sistemi stereo e multicanale

TEMPI - Cap. 4 (Teoria e Pratica)

Autodidatti

Per 300 ore globali di studio individuale (vol. I, teoria e pratica):

- ca. 30 ore

Corsi

Per un corso globale di 60 ore in classe + 120 di studio individuale (vol. I, teoria e pratica):

- ca. 5 ore frontali + 1 di feedback
- ca. 12 di studio individuale

ATTIVITÀ

- Attività al computer: sostituzione di parti di algoritmi, correzione, completamento e analisi di algoritmi, costruzione di nuovi algoritmi

VERIFICHE

- Realizzazione di uno studio breve
- Compiti unitari di Reverse Engineering

SUSSIDI DIDATTICI

- Lista oggetti nativi di Pd – Lista oggetti della libreria Virtual Sound - Glossario

4.1 SEGNALI DI CONTROLLO: IL PANNING STEREOFONICO

Per far oscillare un segnale nello spazio stereofonico, come descritto nel par. 4.1 della teoria, possiamo utilizzare, come segnale di controllo sinusoidale, il normale oggetto `[osc~]`, al quale daremo frequenze molto basse, al di sotto della minima frequenza udibile.
Come abbiamo visto nel par. 1.6, si può definire la posizione stereofonica di un segnale con un valore *n* compreso fra zero e uno. La radice quadrata di tale valore rappresenta il fattore di moltiplicazione dell'ampiezza di un canale mentre la radice quadrata di 1 – *n* quello per l'altro canale. La *patch* in figura 4.1, che vi invitiamo a riprodurre, mostra l'algoritmo per il controllo della posizione stereofonica di un segnale costituito da un'onda a dente di sega limitata in banda.

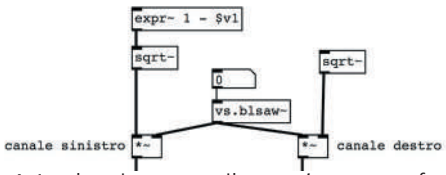


fig. 4.1: algoritmo per il *panning* stereofonico

Ora è necessario collegare all'*inlet* dell'oggetto `[expr~]`[1] e all'*inlet* di `[sqrt~]` a destra il segnale generato da `[osc~]` per controllare la posizione stereofonica. Come sappiamo `[osc~]` genera una sinusoide con valori compresi fra -1 e 1, mentre noi abbiamo bisogno di un segnale compreso fra 0 e 1. Si potrebbe modificare l'intervallo di oscillazione di `[osc~]` con un paio di semplici calcoli, ma questo lo vedremo nel prossimo paragrafo; qui preferiamo usare un oggetto di cui abbiamo già parlato nell'Interludio IB.9. Completate la *patch* come da figura 4.2.

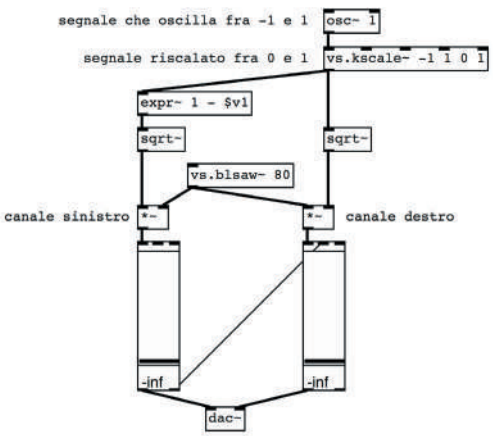


fig. 4.2: *panning* stereofonico controllato da un LFO

[1] Ricordiamo che l'oggetto `[expr~]` è l'omologo per il segnale di `[expr]`. Le variabili per il segnale iniziano sempre con il prefisso '$v' seguito dal numero della variabile. L'*inlet* di sinistra deve sempre essere di tipo segnale, quindi la prima variabile sarà sempre '$v1' mentre le altre variabili (quindi gli altri *inlet*) possono anche essere di tipo 'f' o 'i'.

L'oggetto `[vs.kscale~]`, come sappiamo, ha quattro argomenti, i primi due specificano l'intervallo di entrata e gli ultimi due l'intervallo di uscita. Nel nostro caso, gli argomenti [-1 1 0 1] indicano che se mandiamo a `[vs.kscale~]` un segnale che varia fra -1 e 1 avremo in uscita un segnale riscalato che varia fra 0 e 1, che è esattamente quello che ci serve. L'oggetto `[osc~]` genera una sinusoide di controllo alla frequenza di 1 Hz, il suono fa quindi un 'viaggio' dal canale sinistro al destro e ritorno nel tempo di un secondo: collegando un *number box* a `[osc~]` possiamo variare la frequenza di oscillazione.
Provate con frequenze diverse, ma non superiori a 20 Hz: le frequenze superiori generano fenomeni di modulazione che tratteremo successivamente.

Possiamo semplificare la *patch* usando l'oggetto `[vs.pan~]`, della libreria *Virtual Sound*, che realizza l'algoritmo di *panning* prendendo un suono dall'ingresso sinistro e spostandolo nel fronte stereo secondo il segnale di controllo ricevuto dall'ingresso destro (fig. 4.3).

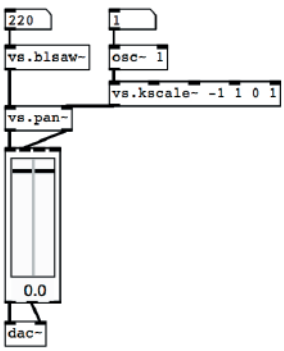


fig. 4.3: *panning* stereofonico con l'oggetto `[vs.pan~]`

Come si vede, l'oggetto `[vs.pan~]` funziona come l'algoritmo di figura 4.2, ma permette di liberare spazio nella nostra *patch*. In questa patch abbiamo anche sostituito i due `[vs.gain~]` con `[vs.sgain~]`, che ne incorpora il funzionamento.

Possiamo usare come segnale di controllo un'altra forma d'onda, ad esempio quella quadra (fig. 4.4).

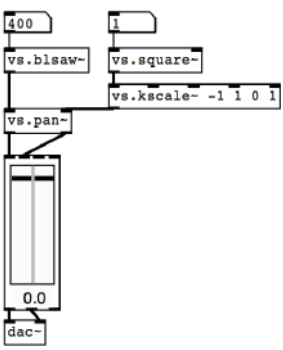


fig. 4.4: controllo del *panning* con LFO a forma d'onda quadra

In questo caso il suono si sposta da un canale all'altro senza passare per posizioni intermedie: questa discontinuità genera un clic indesiderato che può essere eliminato filtrando il segnale di controllo con un filtro passa-basso (`[lop~]`) che serve a 'smussare' gli spigoli dell'onda quadra (fig. 4.5).

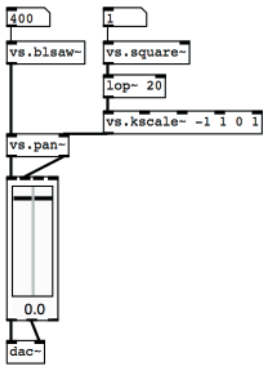


fig. 4.5: filtraggio di un LFO

Qui abbiamo impostato una frequenza di taglio di 20 Hz: questo significa, grosso modo, che il segnale di controllo non può 'saltare' da un valore all'altro in un tempo inferiore a 1/20 di secondo. Provate a variare la frequenza di taglio del filtro per sentirne l'influenza sul percorso del suono: più la frequenza di taglio è bassa e più graduale è il passaggio da un canale all'altro.

(...)

Il capitolo prosegue con:

4.3 SEGNALI DI CONTROLLO PER LA FREQUENZA
Il vibrato
Depth del vibrato
Rate del vibrato

4.4 SEGNALI DI CONTROLLO PER L'AMPIEZZA
Simuliamo un Theremin

4.5 MODULAZIONE DEL DUTY CYCLE (PULSE WIDTH MODULATION)

4.6 SEGNALI DI CONTROLLO PER I FILTRI

4.7 ALTRI GENERATORI DI SEGNALI DI CONTROLLO

4.8 SEGNALI DI CONTROLLO: IL PANNING MULTICANALE

ATTIVITÀ
- Analisi di algoritmi, completamento di algoritmi, correzione di algoritmi,

VERIFICHE
- Compiti unitari di reverse engineering, Realizzazione di uno studio breve

SUSSIDI DIDATTICI
- Lista oggetti nativi Pd - Lista oggetti della libreria Virtual Sound - Glossario

Francesco Bianchi • Alessandro Cipriani • Maurizio Giri

Pure Data: Musica Elettronica e Sound Design

Teoria e Pratica • volume 1

Argomenti trattati

Sintesi ed Elaborazione del Suono - Frequenza, Ampiezza e Forma d'Onda - Inviluppi e Glissandi - Sintesi Additiva e Sintesi Vettoriale - Sorgenti di Rumore - Filtri - Sintesi Sottrattiva - Realizzazione di Sintetizzatori Virtuali - Equalizzatori, Impulsi e Corpi Risonanti - Segnali di Controllo e LFO - Tecniche di Programmazione con Pure Data

Questo è il primo volume di un sistema didattico organico in più volumi. Ad ogni capitolo di teoria corrisponde un capitolo di pratica con il software Pure Data (uno dei più potenti e diffusi software open source per l'elaborazione del suono in tempo reale, per Windows, Mac OSX e Linux) e una sezione online: in questo modo lo studente acquisisce conoscenze, abilità e competenze teorico-pratiche in modo integrato. Il testo può essere studiato autonomamente oppure sotto la guida di un insegnante. È ideale quindi per chi inizia da zero, ma utilissimo anche per chi voglia approfondire la propria competenza nel campo del sound design e della musica elettronica. Pure Data è un linguaggio di programmazione visuale, in cui oggetti grafici vengono collegati tra loro. Tali oggetti, che possono eseguire calcoli o elaborare segnale audio, vengono connessi al fine di creare entità anche molto complesse come sintetizzatori e processori di segnale, fino ad autentici dispositivi sonori automatici. Pure Data è un software gratuito e Open Source: può essere liberamente scaricato e utilizzato sui più diffusi sistemi operativi; è anche possibile manipolare il codice sorgente al fine di creare versioni personalizzate del programma.

FRANCESCO BIANCHI si occupa prevalentemente di composizione e informatica musicale. Ha composto musica elettronica, vocale e strumentale, eseguita in festival e istituzioni di diversi paesi (Huddersfield University, Leeds University, Conservatori di Roma, Torino e Perugia). Ha vinto il terzo premio al concorso Valentino Bucchi nel 2008 con il brano "Cercle". In occasione di Expo 2015 ha collaborato ad un progetto di sonificazione di dati captati da un dispositivo di analisi alimentare sviluppato presso l'università di Parma. Ha creato diverse librerie di software per estendere le funzioni di Pure Data e Max. Per questi software sta lavorando attualmente a biosLib, un insieme di oggetti che implementano algoritmi di vita artificiale.

ALESSANDRO CIPRIANI è coautore del testo "Virtual Sound" sulla programmazione in Csound. Le sue composizioni sono state eseguite nei maggiori festival internazionali di musica elettronica e pubblicate da Computer Music Journal, International Computer Music Conference etc. Ha scritto musiche per il Teatro dell'Opera di Pechino e per film e documentari in cui ambienti sonori, dialoghi e musica, elaborati al computer, si fondono ed hanno funzioni interscambiabili. Ha tenuto seminari in numerose università europee e americane (University of California, Sibelius Academy Helsinki, Conservatorio Tchaikovsky di Mosca, etc.). È titolare della Cattedra di Musica Elettronica del Conservatorio di Frosinone e socio fondatore di Edison Studio. È membro dell'Editorial Board della rivista Organised Sound (Cambridge University Press).

MAURIZIO GIRI è docente in Composizione ed insegna tecniche di programmazione con Max nei Conservatori di Perugia e Frosinone. Ha scritto musica strumentale ed elettroacustica. Attualmente si occupa di musica elettronica e nuove tecnologie applicate all'elaborazione digitale del suono, all'improvvisazione e alla composizione musicale. Ha scritto software di composizione algoritmica, improvvisazione elettroacustica e live electronics. Ha fondato la società Amazing Noises, che sviluppa applicazioni musicali e plug-in per dispositivi mobili e computer tradizionali. Ha pubblicato tutorial su Max in riviste specializzate. È stato artista residente a Parigi (Cité Internationale des Arts) e a Lione (GRAME). Ha collaborato con l'Institut Nicod alla École Normale Supérieure di Parigi ad un progetto di filosofia del suono.

ISBN 978-88-99212-05-6

9 788899 212056

www.contemponet.com

€ 42.00 iva inclusa