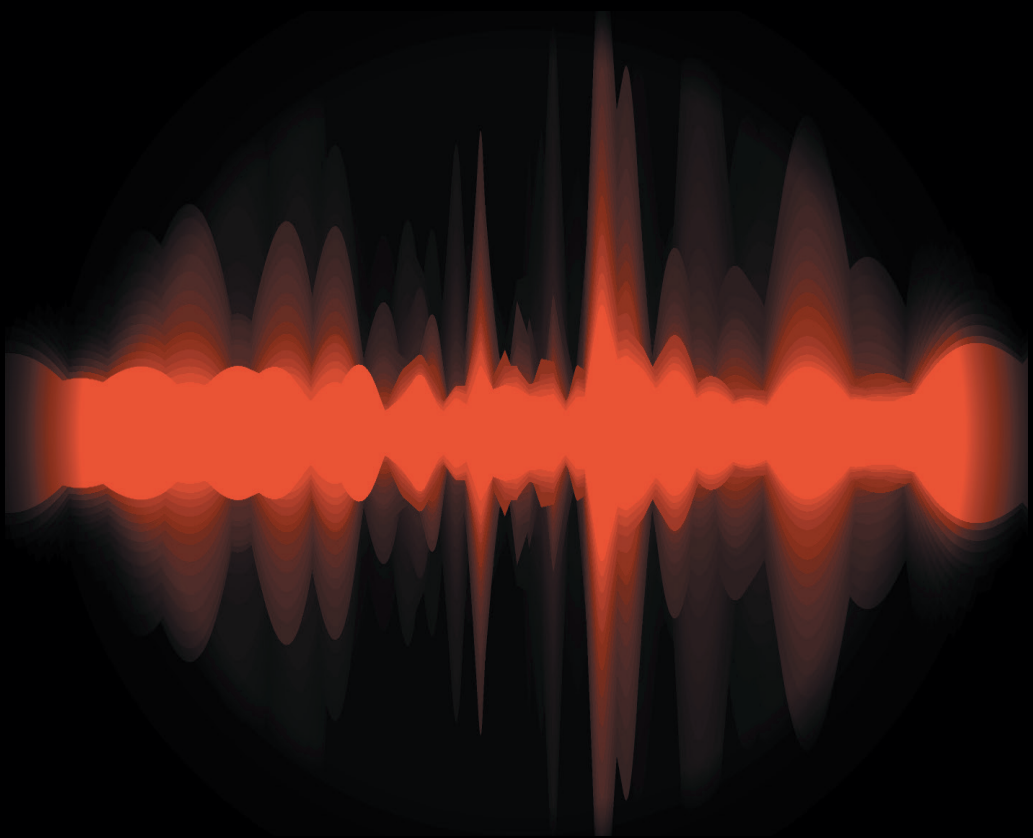
Alessandro Cipriani • Maurizio Giri

Musica Elettronica e Sound Design

Teoria e Pratica con **Max 7** • volume 2



ConTempoNet

Alessandro Cipriani • Maurizio Giri

Questo è un estratto del libro:

MUSICA ELETTRONICA E SOUND DESIGN

Teoria e Pratica con Max 7 - Volume 2

per maggiori informazioni:
www.contemponet.com
www.virtual-sound.com

CIPRIANI A. - GIRI M.
MUSICA ELETTRONICA e SOUND DESIGN
Teoria e Pratica con Max 7
Vol. 2
ISBN 978-88-992120-6-3

Prima edizione 2013
Seconda edizione 2017

Realizzazione Esempi Sonori e Interattivi: Vincenzo Core
Realizzazione indice analitico: Salvatore Mudanò
Immagine di copertina: Federico Foderaro

ConTempoNet s.a.s., Roma
e-mail **posta@contemponet.com**
posta@virtual-sound.com
URL: **www.contemponet.com**
www.virtual-sound.com
facebook **www.facebook.com/electronic.music.and.sound.design**

INDICE

PREFAZIONE
di David Zicarelli

Potrà sembrarvi strano, ma molti anni fa, quando cercavo di imparare come si creano suoni al computer leggendo libri ed articoli, dovevo limitarmi ad immaginare quali suoni producessero le varie tecniche di sintesi. Anche se penso che la mia immaginazione ne sia stata stimolata, sono felice che nel frattempo la tecnologia si sia evoluta al punto che oggi quasi ogni tecnica di sintesi può essere realizzata in tempo reale con un normale computer. L'esperienza percettiva, infatti, è un elemento importantissimo nell'apprendimento delle tecniche di sintesi digitale.

Il libro di Alessandro Cipriani e Maurizio Giri costituisce uno dei primi corsi di musica elettronica che integra esplicitamente percezione, teoria e pratica, usando esempi di sintesi in tempo reale che si possono manipolare e personalizzare. Dal mio punto di vista, la manipolazione del suono costituisce un aspetto estremamente importante nell'apprendimento: consente di acquisire quella che Joel Chadabe chiama "conoscenza predittiva", cioè l'abilità di intuire ciò che succederà ad un suono prima che si compia un'azione per modificarlo. Tutti abbiamo una certa conoscenza predittiva: ad esempio quasi tutti sappiamo che, girando una manopola del volume in senso orario, il suono che viene dal nostro amplificatore aumenterà di intensità. Una volta che entriamo nel regno della sintesi digitale del suono, le cose si fanno molto più complicate di una manopola del volume, e abbiamo bisogno di fare esperienza diretta di manipolazione e di percezione per approfondire la nostra conoscenza predittiva.

Comunque, per istruirsi in modo completo sul suono prodotto digitalmente, abbiamo bisogno di molto di più che la semplice conoscenza predittiva. È necessario sapere perché le nostre manipolazioni generano i cambiamenti che percepiamo. Tale conoscenza teorica rinforza la nostra conoscenza esperienziale e intuitiva e, allo stesso tempo, la nostra esperienza fornisce significato percettivo alle spiegazioni teoriche.

Secondo il mio parere, Cipriani e Giri hanno compiuto un lavoro magistrale consentendo all'esperienza e alla conoscenza teorica di rinforzarsi l'una con l'altra. Questo libro funziona sia come testo all'interno di un corso, sia come mezzo per l'autoapprendimento. In aggiunta, il libro include un'introduzione completa all'elaborazione digitale dei segnali con Max/MSP e costituisce una splendida introduzione ai concetti di programmazione con questo software.

Come vedrete, i capitoli di teoria sono denominati "T", mentre la pratica e la conoscenza esperienziale sono incluse nei capitoli "P". Questi capitoli si alternano, come si muovono la gamba sinistra e quella destra quando si sale una scala, approfondendo e raffinando i concetti a livelli sempre più alti di sofisticazione.

Spero che trarrete vantaggio dagli eccellenti esempi di Max/MSP creati dagli autori: sono insieme divertenti e illuminanti, e suonano abbastanza bene da poter essere utilizzati anche sul palco. Vale la pena esaminarli come modelli per

le vostre patch Max/MSP, o per estenderli in modi sempre nuovi. Ma qualche minuto di "gioco" con gli esempi non è la stessa cosa che studiarli in rapporto ai concetti espressi nel libro. Il libro infatti fornisce il linguaggio per esprimere tali concetti in relazione ai fondamenti teorici. Conoscere la teoria è essenziale, perché presumibilmente state per leggere questo libro perché volete diventare persone che sanno fare molto più che girare una manopola.

Questo è sia l'augurio mio, sia quello degli autori. Voglio augurarvi buona fortuna in questa nuova avventura, e anche ringraziare i miei due amici italiani per aver creato una risorsa per l'apprendimento della musica digitale così completa, proprio quella che desideravo che esistesse quando ero uno studente!

David Zicarelli
Fondatore della Cycling'74, casa produttrice di Max

INTRODUZIONE AL SECONDO VOLUME

Questo è il secondo di una serie di 3 volumi sulla sintesi e l'elaborazione digitale del suono. Il piano dell'opera prevede anche:
- un primo volume che tratta diversi temi fra cui la sintesi additiva, generatori di rumore, filtri, sintesi sottrattiva e segnali di controllo;
- un terzo volume che tratterà argomenti come riverbero e spazializzazione, sintesi non lineare (AM, FM, waveshaping e tecniche di distorsione del suono), sintesi granulare. Nel terzo volume si parlerà anche dell'ambiente Gen, che permette di generare in tempo reale codice eseguibile.

LIVELLO RICHIESTO

Tutti i volumi alternano parti teoriche a sezioni di pratica al computer, che vanno studiate in stretta connessione. Questo secondo volume può essere utilizzato da utenti di diverso livello di preparazione che abbiano però ben chiari i concetti e la pratica su Max delineati nel primo volume. Il percorso di questo volume può essere svolto in auto-apprendimento oppure sotto la guida di un insegnante.

GLI ESEMPI SONORI E GLI ESEMPI INTERATTIVI

Il percorso della parte teorica è accompagnato da molti esempi sonori e interattivi reperibili sul sito all'indirizzo **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.

MAX FOR LIVE

L'ultimo capitolo, o meglio "interludio", del libro riguarda Max for Live, un'applicazione con cui è possibile creare *plug-in* per il software Ableton Live. Si tratta di un capitolo molto "denso", in cui tutte le competenze acquisite nel corso dei primi due volumi vengono messe a frutto per la realizzazione di *device* (così si chiamano i *plug-in* nell'ambiente Live). Particolare attenzione è stata data allo studio delle "Live API" che permettono di creare *device* che controllano altri *plug-in* o lo stesso ambiente Live.

L'IMPOSTAZIONE DIDATTICA

Anche questo tomo, come il primo, va studiato alternando ogni capitolo di teoria a quello corrispettivo di pratica incluse le attività al computer. La novità, rispetto al primo volume è nel tipo di attività pratiche proposte: le attività finali di correzione, analisi, completamento e sostituzione di parti di algoritmi non sono presenti in questo volume; ogni capitolo di pratica presenta, nel corso dell'esposizione, un cospicuo numero di attività da svolgere per verificare e approfondire le competenze acquisite e per metterle in pratica creativamente. L'analisi degli algoritmi e delle patch è svolta dettagliatamente (come nel primo volume) quando si tratta di illustrare nuove tecniche, mentre nel caso di processi già noti abbiamo lasciato l'onere dell'analisi al lettore. Abbiamo in altre parole tenuto conto del fatto che con il secondo volume ci troviamo di fronte a un diverso tipo di lettore. Mentre il nostro "lettore tipo" per il primo volume era infatti

una persona che, pur interessata all'argomento, poteva anche essere completamente priva di esperienza nel campo della musica elettronica, per il secondo volume ci aspettiamo un lettore di tipo "intermedio", che abbia già realizzato dei suoni con Max e/o con altri software, che conosca le basi della sintesi e dell'elaborazione del suono: insomma un lettore che abbia "metabolizzato" le informazioni contenute nel primo volume. Chi avesse già queste competenze, pur non avendo mai letto il primo volume può sicuramente trarre profitto da questo secondo tomo: dobbiamo però avvertire che nel corso del testo facciamo spesso riferimento a concetti, oggetti ed algoritmi trattati nel libro precedente.
Ci preme sottolineare la presenza, in questo volume, del capitolo denominato "L'arte dell'organizzazione del suono: processi di movimento" (cap.8 di teoria e pratica), in cui al lettore è data la possibilità di elaborare le sue interpretazioni delle attività proposte, più complesse e creative rispetto a quelle del primo volume. Ciò significa che viene incoraggiato ancora di più l'uso della percezione, dell'analisi e del pensiero critico, dell'esperienza e della capacità inventiva del lettore. Non deve sfuggire l'importanza di una sezione dedicata all'uso creativo della propria conoscenza e delle proprie abilità. Mentre il software si evolve ed è soggetto a mutamenti nel tempo, infatti, le competenze acquisite tramite una pratica di invenzione attiva e personale rappresentano un bagaglio flessibile, che può essere applicato a contesti tecnologici diversi. Crediamo che un approccio troppo passivo e "libresco" all'apprendimento sia sterile, mentre il nostro scopo è quello di consentire al lettore di mettere in relazione in modo organico, inventivo e personale la propria conoscenza, le proprie abilità, la propria percezione e capacità di analisi, la capacità di saper fare le domande giuste e risolvere problemi e la capacità di creare forme sonore originali. La sfida che vogliamo raccogliere è dunque quella di lavorare nel campo delle competenze.
Per poter dire di essere competenti in questo settore non è solo importante, ad esempio, sapere cosa sia e come si crea un LFO, ma sapere anche come utilizzarlo in contesti specifici in cui la motivazione di tipo creativo o produttivo spinge in una data direzione. In poche parole bisogna sapere, di un LFO, anche cosa farne. Infatti, saper adottare semplicemente delle procedure non vuol dire essere competenti, l'esperto sa anche *interpretare* quelle procedure. In sostanza, sapere come impostare una patch e saper modificare i valori dei parametri di un oggetto non in astratto, ma per raggiungere determinati scopi di movimento o evoluzione del suono nel tempo o nello spazio, è una meta essenziale per gli artisti del suono, i sound designer e i compositori.
Nel primo volume abbiamo semplicemente accennato, tramite alcuni esercizi di *reverse engineering*, alla possibilità che il punto di partenza per l'uso di un sistema per la sintesi e l'elaborazione del suono non fosse tanto la teoria, quanto un contesto concreto, in questo caso la possibilità di partire da un suono esistente proposto da noi, di riconoscerne le caratteristiche e simularne lo spettro, l'inviluppo, etc.
Nel capitolo 8 di questo secondo volume, la conoscenza della teoria e le abilità nella pratica sviluppate sinora saranno ancor di più valorizzate chiamando in campo l'elaborazione originale del lettore e la sua capacità di costruire processi di movimento. Le attività compositive di base proposte si limiteranno, in questo capitolo, ad articolazioni sonore che non superano un minuto. Tali processi dunque saranno costruiti al di fuori di una dimensione e di un

contesto formale più ampi. Ma di che tipo di creazione sonora parliamo? Esistono pratiche completamente diverse, dalla composizione algoritmica alle "orchestre di laptop", dall'elettronica *live* con interazione uomo-macchina alle composizioni acusmatiche fino alla *soundscape composition*, alle installazioni sonore e audiovisive, alla *sound art*, a lavori di *sound design* etc. Infiniti sono anche gli approcci delle diverse scuole, da quelli narrativi a quelli antinarrativi, a quelli di tipo ambientale etc. Il nostro desiderio è quello di dare semplicemente alcuni strumenti per affinare le proprie competenze evitando per quanto possibile di dare norme o regole di comportamento e cercando invece di proporre esperienze personalizzabili. A questo proposito abbiamo deciso di reinterpretare e adattare a scopi creativi alcuni concetti riguardanti la spettromorfologia esposti da Denis Smalley in alcuni suoi articoli, e abbiamo introdotto le categorie dei movimenti semplici, complessi e composti. Nell'interazione fra teoria e pratica proponiamo ad ogni studente, sulla base di indicazioni tecniche e scopi precisi, di interpretare tali movimenti descritti, ponendone in essere la propria versione sonora.

MATERIALE DI SUPPORTO

Tutto il materiale a cui si fa riferimento nel corso del libro è reperibile nella pagina di supporto all'indirizzo **www.**********.

Per iniziare a lavorare con questo testo è necessario scaricare gli **Esempi Sonori e Interattivi** che si trovano alla pagina di supporto. Durante la lettura dei capitoli di teoria si farà costante riferimento a questi esempi.

Per affrontare la parte pratica è invece necessario aver installato il programma **Max**, reperibile al sito www.cycling74.com. Bisogna inoltre scaricare la libreria **Virtual Sound Macros** dalla pagina di supporto di questo testo; 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 Max) relative a tutti i capitoli di pratica.

BIBLIOGRAFIA

Nelle ultime pagine è riportata una bibliografia essenziale, oltre ai riferimenti bibliografici relativi ai testi citati nel libro.

COMMENTI E CORREZIONI

Correzioni e commenti sono benvenuti. Vi preghiamo di inviarli per e-mail a: a.cipriani@edisonstudio.it oppure maurizio@giri.it

RINGRAZIAMENTI

Si ringraziano:
Vincenzo Core e Salvatore Mudanò per il loro paziente e lungo lavoro;
Lorenzo Seno per le preziose indicazioni sull'audio digitale;
Richard Boulanger, Marco Massimi e David Zicarelli, per la loro disponibilità.

DEDICA

Questo volume è dedicato ad Arianna Giri, Sara Mascherpa e Gian Marco Sandri.

5T
AUDIO DIGITALE E SUONI CAMPIONATI

PREREQUISITI PER IL CAPITOLO

- Contenuti del volume 1 (teoria)

OBIETTIVI

Conoscenze

- Conoscere i meccanismi su cui si basa la conversione analogico-digitale e digitale-analogica del suono
- Conoscere le caratteristiche qualitative della conversione e delle schede audio
- Conoscere i fondamenti della compressione dei dati
- Conoscere le cause e le conseguenze del foldover e del rumore di quantizzazione
- Conoscere i vari metodi di editing e organizzazione dei suoni all'interno di un campionatore
- Conoscere i vari metodi di segmentazione e manipolazione del pitch di suoni campionati

Abilità

- Saper individuare all'ascolto le differenze base fra decimazione e diminuzione dei bit e saperle descrivere
- Saper individuare all'ascolto la differenza base fra un suono originale e il suo reverse e saperla descrivere
- Saper individuare all'ascolto la differenza base fra un suono trattato mediante tecnica dei blocchi e uno trattato con la tecnica dello slicing

CONTENUTI

- Schede audio e conversione AD-DA del suono
- Teorema di Nyquist e foldover
- Rumore di quantizzazione e dithering
- Organizzazione dei suoni in un campionatore
- Segmentazione di suoni campionati: tecnica dei blocchi e slicing
- Modulazione del pitch nei suoni campionati
- Compressione dei dati audio
- Trasmissione dei dati audio e jitter

ATTIVITÀ

- Esempi sonori e interattivi

VERIFICHE

- Test a risposte brevi
- Test con ascolto e analisi

SUSSIDI DIDATTICI

Concetti di base - Glossario

5.1 IL SUONO DIGITALE

Abbiamo affermato, nel par. 1.2T, che un suono è un 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. Un suono acustico può essere amplificato, riprodotto, elaborato; tuttavia, per fare ciò, si deve necessariamente trasformare questo suono in un segnale che sia misurabile, ripetibile e modificabile in modo semplice. Ipotizziamo di avere un amico flautista che sta suonando; per trasformare il suo suono acustico possiamo adoperare un microfono. Il microfono opererà una trasduzione elettroacustica, cioè un rilevamento continuo delle variazioni di pressione nell'aria. Allo stesso tempo il microfono genererà un segnale elettrico analogo a quello originale, nel senso che l'andamento della sua tensione elettrica in uscita corrisponde, ovvero è analogo, a quello dell'onda sonora in entrata. Per questo il segnale in uscita dal microfono viene detto **segnale analogico**. Attenzione però: il segnale analogico non è mai identico a quello originale, ma presenta sempre una certa distorsione (anche minima) e un'introduzione di rumore (fig. 5.1).

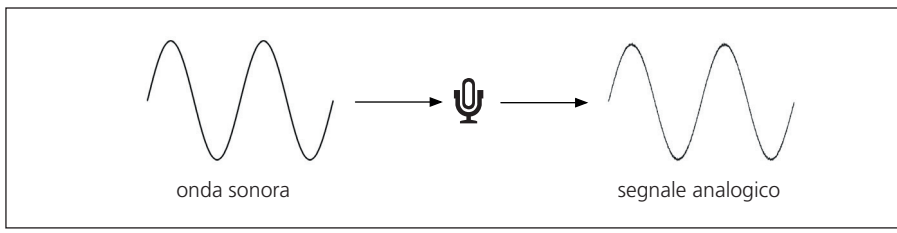


fig. 5.1: il segnale analogico

Nei **segnali digitali** (dall'inglese *digit* = cifra), invece, il segnale viene rappresentato da una serie di numeri. Ognuno di questi numeri rappresenta il valore della pressione istantanea, cioè il valore che la pressione sonora assume in un dato istante. Per generare il segnale digitale l'ampiezza del suono viene misurata a intervalli regolari (fig. 5.2); tale processo si chiama campionamento ed è interamente analogico[1].

I singoli campioni analogici disposti nel tempo, ognuno dei quali assume il valore in ampiezza del segnale in quell'istante, vengono poi convertiti in un flusso di dati numerici (binari): quest'ultimo processo si chiama **conversione analogico/digitale**. Per poter riascoltare il segnale convertito in digitale è necessaria una **conversione digitale/analogica**, cioè un processo per mezzo del quale il segnale digitale viene riconvertito in segnale analogico da inviare all'amplificatore e agli altoparlanti.

[1] Il sistema di campionamento maggiormente in uso è il PCM (*Pulse Code Modulation*), basato sull'acquisizione del segnale analogico a intervalli di tempo regolari. Il processo di campionamento avviene mediante modulazione a prodotto, che consiste nella moltiplicazione del segnale per una serie di impulsi. Il risultato di tale moltiplicazione è uno spettro contenente frequenze pari alle somme e alle differenze delle frequenze contenute nei segnali coinvolti. Si tratta in altre parole di una modulazione ad anello, una tecnica che tratteremo in dettaglio nel capitolo del terzo volume dedicato alla sintesi non lineare.

Le schede audio che troviamo nei nostri computer comprendono sia il convertitore analogico/digitale (o **ADC** - ***Analog to Digital Converter***) sia il convertitore digitale/analogico (o **DAC** - ***Digital to Analog Converter***).

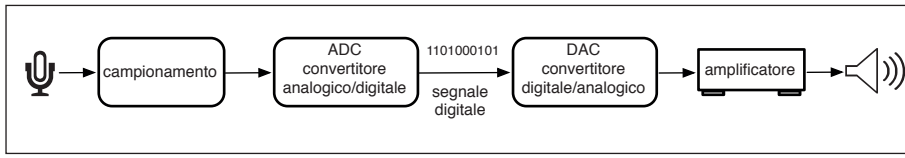


fig. 5.2: campionamento e conversione

CONVERSIONE ANALOGICO/DIGITALE

Come abbiamo detto, mediante la trasformazione di un segnale da analogico a digitale, detta appunto conversione analogico/digitale, possiamo "tradurre" un segnale dato da una tensione elettrica in un segnale numerico, ridefinendo a intervalli di tempo regolari il valore della tensione stessa in linguaggio binario. In realtà, sono necessari diversi "step" per passare da un suono analogico ad uno digitale, come vedremo alla fine di questo paragrafo, quando entreremo più in dettaglio. Per ora basterà sapere che nell'ADC avviene un campionamento analogico e una conversione in digitale.

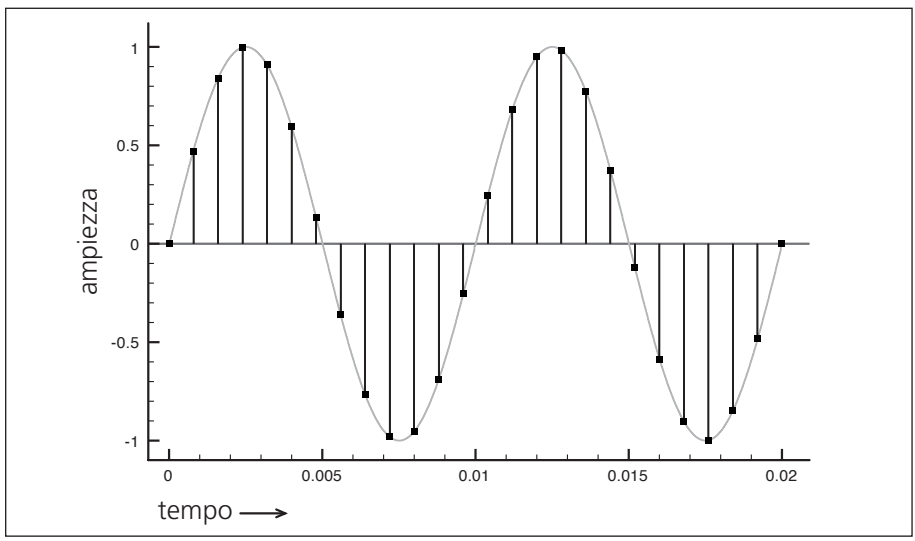


fig. 5.3: il segnale campionato

Nella fig. 5.3 la linea continua rappresenta in modo sintetico l'andamento del segnale analogico (tensione elettrica), mentre i punti sovrapposti rappresentano il valore del segnale campionato dopo che è stato convertito in valori numerici. Il campionamento e la conversione del primo valore di ampiezza (cioè il valore 0) di questo segnale avviene all'istante 0; dopo un dato tempo (all'istante 0.001) avverrà di nuovo un campionamento e una conversione, nel frattempo il segnale analogico ha avuto un'evoluzione del valore dell'ampiezza istantanea,

e presenta in questo istante un valore di ampiezza pari a circa 0.5. Il successivo campionamento avviene all'istante 0.002, e il valore d'ampiezza convertito è di poco superiore a 0.8. Come si vede il campionamento registra solo alcuni tra gli infiniti valori che il segnale analogico continuo assume nel tempo, e si potrebbe quindi pensare che il segnale digitale risultante contenga degli errori, ovvero che non sia fedele all'originale. Ma, come vedremo tra poco, se il segnale analogico contiene solo frequenze inferiori alla metà della frequenza di campionamento, dai campioni digitali è possibile ricostruire senza ambiguità il segnale originale. L'intervallo di tempo che passa fra un campionamento e il successivo si chiama **periodo di campionamento**, e si misura in secondi o in un sottomultiplo (millisecondi o microsecondi). Nella figura 5.4 è indicato con *pc*.

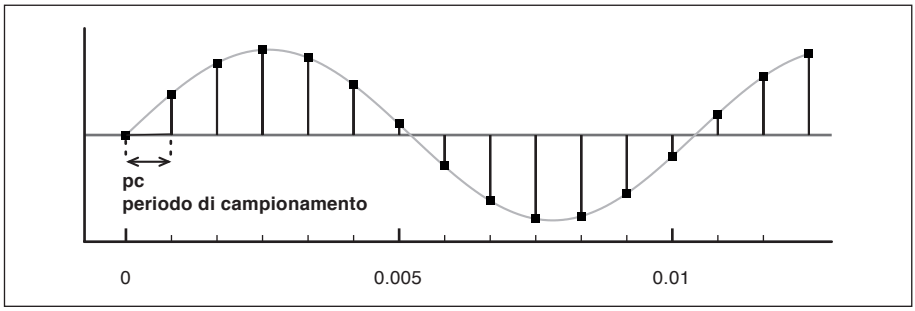


fig. 5.4: periodo di campionamento

L'inverso del periodo di campionamento viene definito **frequenza (o tasso) di campionamento** (*sample rate*, che abbrevieremo di seguito con sr), e si misura in Hz (vedi cap. 1.5T del primo volume). Ad esempio, se utilizziamo una frequenza di campionamento di 48000 Hz per campionare un suono, vuol dire che in un secondo la sua ampiezza verrà misurata 48000 volte.

Valgono quindi le relazioni:

sr = 1 / pc

pc = 1 / sr

Ovvero la frequenza di campionamento (cioè il numero di campioni al secondo) è pari all'inverso del periodo di campionamento, e viceversa.

Per capire quale dovrà essere la *sr* che permetta un campionamento corretto del suono dobbiamo rifarci al **teorema di Nyquist** (detto anche **teorema del campionamento**), il quale afferma che è necessario che la frequenza di campionamento sia maggiore del doppio della massima frequenza contenuta nel segnale. Indicando con *fmax* la frequenza massima contenuta nel segnale, la frequenza di campionamento *sr* che dovremo utilizzare per riprodurla correttamente dovrà essere:

sr > 2 · fmax

Ciò significa naturalmente che la massima frequenza rappresentabile sarà inferiore alla metà della frequenza di campionamento. La frequenza pari alla metà di *sr* è detta **frequenza di Nyquist**.

Prima di effettuare un campionamento è necessario che le frequenze superiori alla frequenza di Nyquist eventualmente contenute nel segnale analogico vengano eliminate, perché queste creano un effetto indesiderato, detto *foldover*, di cui parleremo tra poco. A questo scopo, i sistemi di conversione prevedono alcuni accorgimenti importanti, fra cui l'applicazione di un filtro passa-basso analogico prima del campionamento (detto **filtro anti-aliasing**) in modo da eliminare le frequenze superiori alla frequenza di Nyquist.

Bisogna però tenere presente che è impossibile realizzare un filtro analogico con pendenza infinitamente ripida ed è tecnicamente molto difficile realizzarne uno con pendenza molto elevata. Il rischio, dunque, è quello che rimanga, dopo il filtraggio, una parte delle frequenze superiore a quella di Nyquist. Per risolvere questo problema, dopo il filtraggio analogico (effettuato con un filtro a pendenza non ripida), i sistemi di campionamento (*sampling*) utilizzano una tecnica di sovracampionamento (*oversampling*). Con la tecnica dell'*oversampling* il suono viene campionato ad una frequenza più alta di quella da noi scelta, in modo da avere una nuova frequenza di Nyquist molto al di sopra della frequenza di taglio del filtro analogico. Successivamente, al segnale sovracampionato viene applicato dal sistema un filtro passa-basso digitale, con pendenza ripida, che elimina le frequenze al di sopra della prima frequenza di Nyquist (quella precedente il sovracampionamento). Infine il suono viene ricampionato alla frequenza da noi scelta tramite un processo di *downsampling*. In questo modo siamo certi che, quando campioniamo un suono, non avremo effetti di *foldover*.
I risultati migliori, al di là delle tecniche citate, si ottengono utilizzando frequenze di campionamento superiori a 44.100 Hz, in modo tale da spostare a frequenze più alte le immagini degli spettri indesiderati. Per questo sono state introdotte frequenze di campionamento superiori.
Quali sono le frequenze di campionamento più in uso? Nel caso del compact disc si è adottata una *sr* di 44.1 kHz (che permette di riprodurre frequenze fino a 22050 Hz), mentre per il DVD e per il Blu-ray Disc la *sr* può arrivare a 192 kHz (e permette quindi frequenze fino a 96000 Hz, ben al di sopra della massima frequenza udibile).

IL FOLDOVER NEI SUONI GENERATI IN DIGITALE

Cosa succederebbe se il sistema di campionamento non applicasse alcun filtro anti-aliasing e lasciasse passare le frequenze superiori alla frequenza di Nyquist? O se, invece di campionare un suono, generassimo digitalmente (dall'interno del computer) un segnale di frequenza superiore a quella di Nyquist? In entrambi i casi otterremmo un effetto di *foldover* (ripiegamento). Il ***foldover*** è il fenomeno per cui le componenti frequenziali che superano la metà della *sr* vengono riflesse al di sotto di questa. Per esempio, una componente frequenziale di 11000 Hz, convertita con una *sr* di 20000 Hz, darà luogo a una componente di *foldover* di 9000 Hz, come vedremo in dettaglio tra poco.

Immaginiamo di voler generare una sinusoide con frequenza 12000 Hz. Se utilizziamo una frequenza di campionamento (sr) uguale a 18000 Hz, avremo una frequenza di Nyquist pari a 9000 Hz. Il suono che vogliamo generare, quindi, *supera di 3000 Hz* la frequenza di Nyquist. Tale suono, perciò, non avrà la sua frequenza originale, (nel caso descritto 12000 Hz) ma comparirà, con il segno invertito[2], 3000 Hz al di sotto della frequenza di Nyquist (ovvero 6000 Hz, ma con segno negativo).
Nel caso in cui, quindi, la frequenza di un oscillatore venga impostata al di sopra della frequenza di Nyquist, la frequenza generata dal *foldover* si calcolerà con la seguente formula; detta **sr** la frequenza di campionamento, **fc** la frequenza da convertire ed **fg** la frequenza generata, abbiamo:

$$\mathbf{f_g = f_c - sr}$$

Applichiamo la formula al caso precedente:
12000 - 18000 = -6000

Notate che questa formula vale solo nel caso di frequenze *fc* comprese tra la metà della frequenza di campionamento (ovvero la frequenza di Nyquist) e 1.5 volte la frequenza di campionamento *sr*. Vedremo, nella sezione dedicata all'*aliasing*, una formula generale valida per tutte le frequenze.

Questi fenomeni hanno luogo, naturalmente, non solo in relazione a segnali sinusoidali, ma per qualsiasi componente di un segnale complesso. Osserviamo la fig. 5.5: abbiamo un segnale costituito da varie parziali con frequenza fondamentale pari a 2000 Hz: le linee continue rappresentano le componenti effettivamente presenti nel segnale dopo la conversione, quelle tratteggiate le componenti originali, che non sono più presenti nel segnale convertito, e quelle in grassetto le stesse componenti che vengono "riflesse" al di sotto della frequenza di Nyquist (che in questo esempio sono pari a 22050 Hz). Prendiamo in esame la frequenza di 24000 Hz (cioè la 12ª armonica della fondamentale di 2000 Hz): essendo superiore alla frequenza di Nyquist, è soggetta a *foldover*, e diventa:

freq. da convertire	**- freq. di campionamento**	**=**	**freq. generata**
24000	**- 44100**	**=**	**-20100**

mentre la frequenza di 28000 Hz. (14ª armonica della fondamentale di 2000 Hz) diventa:

28000	**- 44100**	**=**	**-16100**

[2] Il comportamento di ogni parziale, quando si inverte il segno, è dipendente dalla forma d'onda e dalla fase. Ad esempio le sinusoidi, quando il segno è invertito, invertono la fase, mentre le cosinusoidi, invertendo il segno, rimangono uguali. Da questo punto di vista quindi non è facilmente calcolabile l'effetto di *foldover* sullo spettro in uscita.

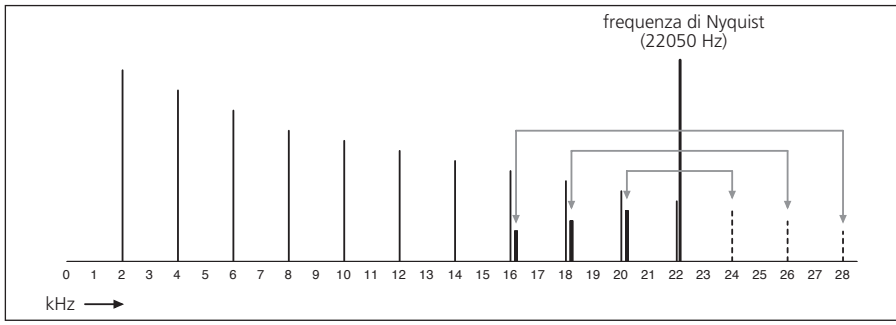


fig. 5.5: *foldover*

Nei tre esempi interattivi seguenti possiamo ascoltare tre eventi diversi, tutti basati su una frequenza di campionamento a 22050 e una frequenza di Nyquist a 11025. Da ciò risulta che qualsiasi suono superi gli 11025 Hz subirà un effetto di *foldover*:

- nel primo esempio facciamo glissare un suono sinusoidale da 20 Hz a 10000 Hz, il *foldover* è assente, e ascoltiamo un semplice glissato ascendente;

- nel secondo esempio il suono glissa da 20 a 20000 Hz. Nel momento in cui il glissato supera gli 11025 Hz si ha il fenomeno del *foldover*. Superata la soglia della frequenza di Nyquist infatti il glissato diventa discendente perché quanto più la frequenza sale, tanto più il *foldover* fa scendere la frequenza riflessa. Il suono glissa in modo discendente fino a fermarsi a 2050 Hz: infatti secondo la formula citata;

fc	**- sr**	**=**	**freq. generata**
20000 Hz	**- 22050**	**=**	**-2050**

- nel terzo esempio il suono glissa da 20 a 30000 Hz. In questo caso si verifica un *foldover* doppio; vediamolo in dettaglio:
  1) Nella fase iniziale da 20 a 11025 Hz la frequenza generata corrisponde a quella programmata.
  2) Nel momento in cui il glissato supera gli 11025 Hz si verifica il primo *foldover* che fa glissare la frequenza generata in modo discendente fino ad arrivare allo 0 (man mano che la frequenza da convertire glissa da 11025 a 22050).

fc	**- sr**	**=**	**freq. generata**
22050 Hz	**- 22050**	**=**	**0**

  3) Il glissato della frequenza programmata continua oltre i 22050 fino ad arrivare a 30000 Hz. Nel momento in cui il suono riflesso supera (in negativo) la soglia dello zero, la frequenza comincia di nuovo a glissare verso l'acuto, per effetto di un altro *foldover* che si ha quando la frequenza del segnale da convertire è minore di zero. In generale, le frequenze sotto lo zero ricompaiono con segno invertito e a specchio nel campo positivo (-200 Hz diventa 200 Hz, -300 Hz diventa 300 Hz etc.).

Per effetto di questo secondo *foldover* la frequenza generata risale fino a 7950 Hz. Da dove ricaviamo questa frequenza finale? 7950 Hz è uguale a 30000 Hz (frequenza finale del segnale da convertire) meno 22050 Hz (frequenza di campionamento). Notate che, essendo causata da un doppio *foldover*, la frequenza finale subisce una doppia inversione del segno.

Ricapitolando, quando la frequenza programmata supera la frequenza di Nyquist (in questo caso 11025 Hz) si verifica un primo *foldover*; quando la frequenza programmata supera la frequenza di campionamento (in questo caso 22050 Hz) si verifica un secondo *foldover* (fig. 5.6 in basso).

fc	**- sr**	**=**	**freq. generata**
30000 Hz	**- 22050**	**=**	**7950**

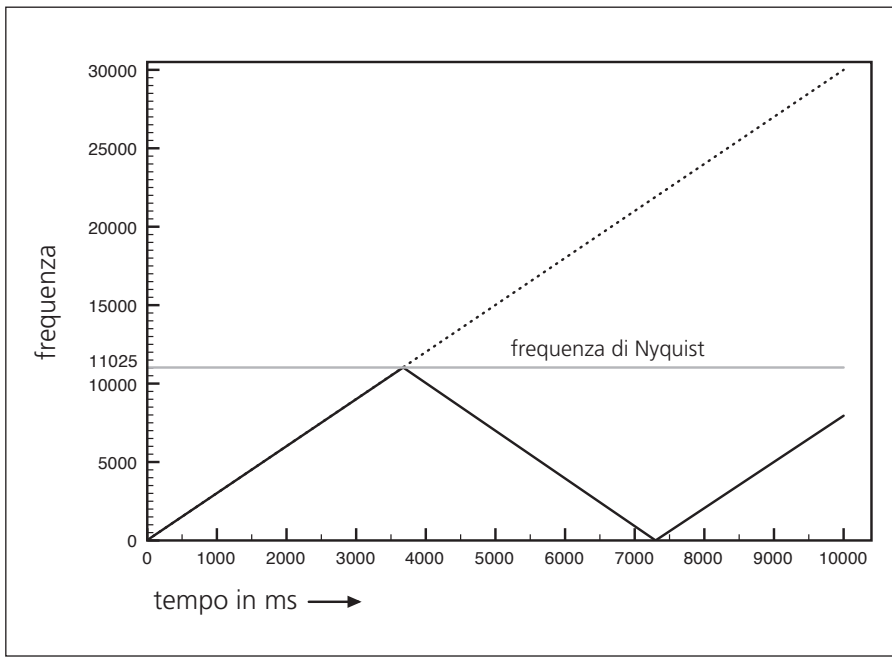


fig. 5.6: *foldover* doppio

ESEMPI SONORI 5A • Foldover

5A.1 Glissato ascendente semplice da 20 Hz a 10000 Hz e sr = 22050
5A.2 Glissato ascendente con *foldover* da 20 Hz a 20000 Hz e sr = 22050
5A.3 Glissato ascendente con *foldover* doppio da 20 Hz a 30000 Hz e sr = 22050

ALIASING: LE CAUSE

Fin qui abbiamo spiegato gli effetti del *foldover*, ma per approfondirne le cause abbiamo bisogno di esplicitare meglio il contenuto del teorema di Nyquist. Per avere un'idea più precisa del limite imposto dal teorema di Nyquist, immaginiamo di avere un sistema di campionamento privo di filtro anti-aliasing con una frequenza di campionamento **sr**, e campioniamo una sinusoide la cui frequenza **f** sia minore della frequenza di Nyquist; quello che otteniamo è una serie di campioni che rappresenta la sinusoide. Ebbene, se con la stessa **sr** campioniamo una sinusoide la cui frequenza è (**f + sr**), cioè la cui frequenza è pari alla somma della frequenza della prima sinusoide più la frequenza di campionamento (al di sopra quindi della frequenza di Nyquist), otteniamo esattamente la stessa serie di campioni che abbiamo ottenuto campionando la sinusoide di frequenza **f**. Non solo, ma otterremmo la stessa serie di campioni anche campionando le sinusoidi di frequenza (**f + 2 · sr**), (**f + 3 · sr**), (**f + 4 · sr**), e così via all'infinito. Generalizzando possiamo dire che, data una frequenza di campionamento **sr**, tutte le sinusoidi di frequenza **f + un multiplo intero della sr** vengono convertite nella stessa serie di campioni.
Facciamo un esempio: se abbiamo una frequenza di campionamento **sr** pari a 5000 Hz, e campioniamo una sinusoide con frequenza **f** di 1000 Hz, otterremo una certa serie di valori campionati. Se ora campioniamo una sinusoide di 6000 Hz, cioè di una frequenza pari a **f** (che vale 1000) più **sr** (che vale 5000) otteniamo la stessa serie di campioni.

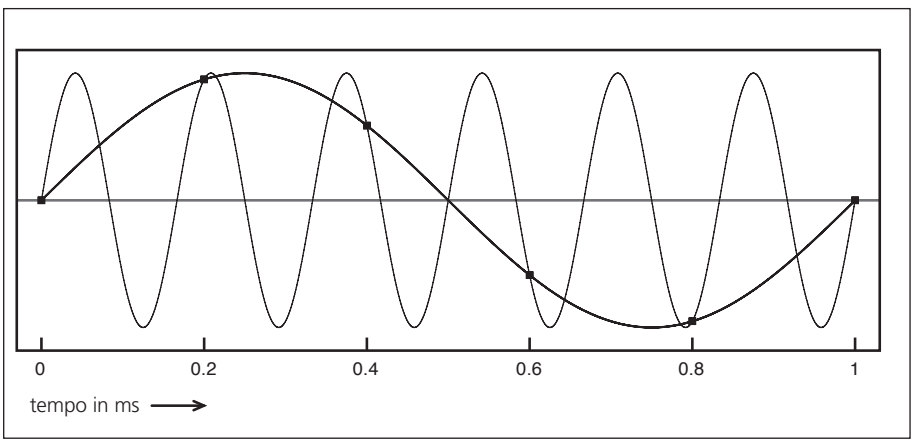


fig. 5.7: due sinusoidi, una di 1000 Hz e l'altra di 6000 Hz campionate a 5000 Hz

In fig. 5.7 vediamo come due sinusoidi di frequenza diversa, rispettivamente di 1000 Hz e 6000 Hz, campionate con la stessa **sr** di 5000 Hz, diano luogo agli stessi campioni (individuati nel grafico dai quadrati). La stessa serie di valori si può ottenere campionando una sinusoide di 11000 Hz (**f + 2 · sr**) o di 16000 Hz (**f + 3 · sr**), etc.
Questa equivalenza si ha anche utilizzando i *multipli interi negativi* della frequenza di campionamento: ovvero anche le sinusoidi di frequenza (**f - sr**), (**f - 2 · sr**), (**f - 3 · sr**) etc. generano la stessa serie di campioni.

Per tornare al nostro esempio, una sinusoide di frequenza -4000 Hz, ovvero la somma dei valori di frequenza 1000 Hz (**f**) e di -5000 Hz (**-sr**), genera gli stessi valori di ampiezza della sinusoide di frequenza 1000 Hz. Una sinusoide di frequenza -4000 Hz equivale, come sappiamo, a una di 4000 Hz con il segno invertito. (vedi fig. 5.8).

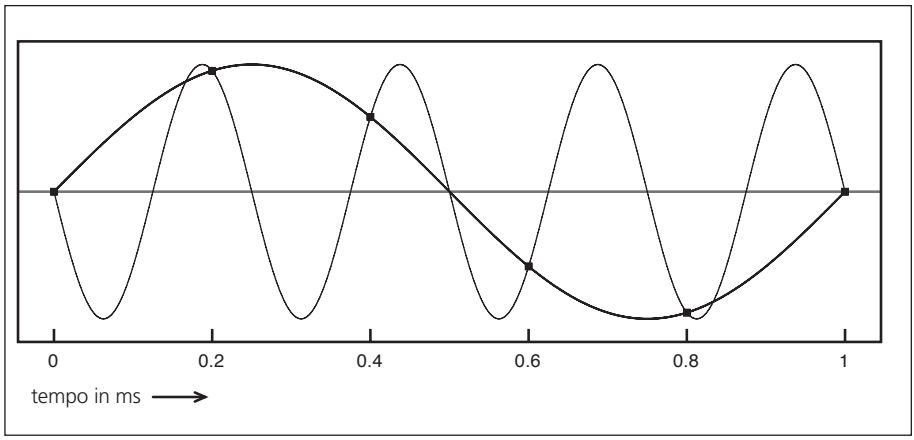


fig. 5.8: due sinusoidi, una di 1000 Hz e l'altra di -4000 Hz campionate a 5000 Hz

Riassumendo possiamo dire che: data una frequenza di campionamento **sr**, e definito con **k** un numero intero qualsiasi, positivo o negativo, non possiamo distinguere tra i valori campionati di una sinusoide di frequenza **f** Hz e quelli di una sinusoide di frequenza (**f + k · sr**) Hz.

Tornando al nostro esempio, ecco le frequenze delle sinusoidi che si trovano all'interno della banda audio e che generano gli stessi campioni:

frequenza **(f+k · sr)**
1000 Hz = 1000 + (0 · 5000)
6000 Hz = 1000 + (1 · 5000)
11000 Hz = 1000 + (2 · 5000)
16000 Hz = 1000 + (3 · 5000)
-4000 Hz = 1000 + (-1 · 5000)
-9000 Hz = 1000 + (-2 · 5000)
-14000 Hz = 1000 + (-3 · 5000)

La situazione non cambia anche convertendo un suono più complesso di una sinusoide: la relazione (**f + k · sr**), infatti, varrà per ciascuna componente spettrale del suono campionato.
Da questa relazione possiamo ricavare la formula che ci permette di calcolare la frequenza generata da un'arbitraria frequenza da campionare. Indicando con **fc** la frequenza da campionare, con **sr** la frequenza di campionamento, con **fg** la frequenza generata e con **N** il multiplo intero di **sr** più vicino a **fc**, avremo:

fg = fc-N · sr

Come si vede è una formula molto simile a quella, semplificata, già vista nella sezione dedicata al *foldover*.

Facciamo alcuni esempi:

1) **fc** = 6000, **sr** = 10000, **N** = 6000/10000 = 0.6 = 1 (valore intero più vicino)

da cui: 6000 - 1 • 10000 = -4000

2) **fc** = 13000, **sr** = 10000, **N** = 13000/10000 = 1.3 = 1 (valore intero più vicino)

da cui: 13000 - 1 • 10000 = 3000

3) **fc** = 21000, **sr** = 10000, **N** = 21000/10000 = 2.1 = 2 (valore intero più vicino)

da cui: 21000 - 2 • 10000 = 1000

4) **fc** = 2500, **sr** = 10000, **N** = 2500/10000 = 0.25 = 0 (valore intero più vicino)

da cui: 2500 - 0 • 10000 = 2500 (in questo caso non c'è *foldover*, perché **fc < sr/2**)

Osserviamo ora cosa succede nel dominio della frequenza:
In un ipotetico sistema di campionamento senza filtri *anti-aliasing*, le componenti, a causa di quanto abbiamo appena descritto, formerebbero più immagini dello stesso spettro, dette **alias**, replicate periodicamente intorno ai multipli della frequenza di campionamento. Più precisamente si otterrebbe uno spettro periodico che, in un campionamento ideale, si ripete all'infinito lungo l'asse delle frequenze. Nella figura 5.9 vediamo come le varie repliche si dispongano nel dominio della frequenza nel caso di una sinusoide a 1000 Hz campionata con *sr* = 5000. Le frequenze mostrate nel grafico sono le stesse della precedente tabella: 1000, 6000, 11000, 16000, -4000, -9000, -14000. Come sappiamo, le frequenze negative si riflettono nel campo positivo, quindi nel grafico sono mostrate componenti a 1000 (frequenza campionata, che chiameremo *f*), 4000 (**sr-f**, cioè 5000-1000), 6000 (**sr+f**), 9000 (**sr · 2-f**), 11000 (**sr · 2+f**), 14000 (**sr · 3-f**), 16000(**sr · 3+f**), etc. In figura 5.9 lo spettro è rappresentato come simmetrico attorno allo 0 e, per traslazione, le successive repliche sono simmetriche attorno ai multipli della frequenza di campionamento.

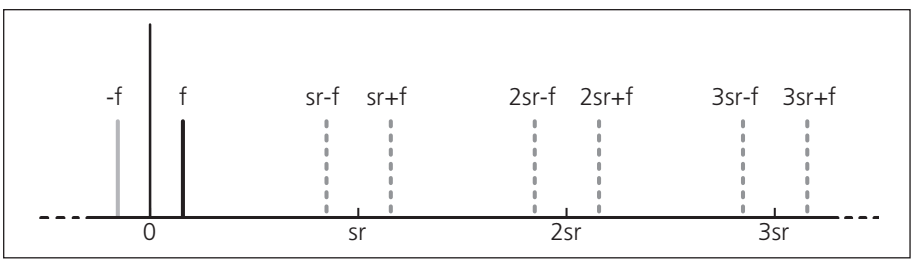


fig. 5.9: *aliasing*

Naturalmente, nei sistemi di campionamento reali, le frequenze al di sopra di quella di Nyquist vengono eliminate prima del campionamento per annullare i fenomeni di cui abbiamo parlato. Come vedremo più avanti, nel caso della decimazione, ovvero del sottocampionamento di un segnale digitale, le frequenze derivate dall'*aliasing* saranno invece presenti e udibili, e dovranno essere previamente filtrate con un apposito filtro passa-basso digitale.

Cosa succede se invece di una sinusoide campioniamo un suono composto da più parziali? Ognuna delle componenti verrà replicata allo stesso modo. Di conseguenza il grafico conterrà repliche dello spettro complesso, come nella figura 5.10.

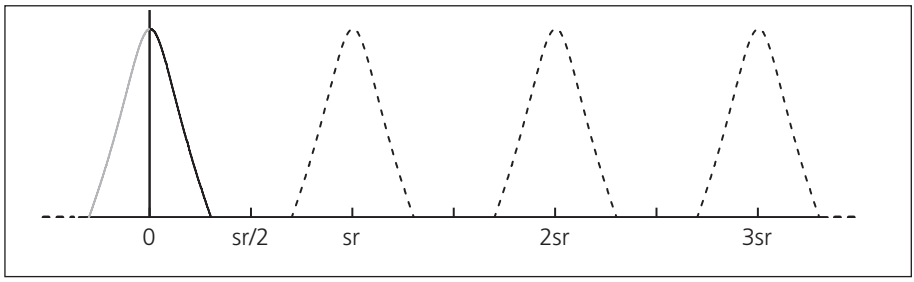


fig. 5.10: *aliasing* di un suono complesso

CONVERSIONE DIGITALE/ANALOGICA

Consideriamo ora il processo inverso, la conversione digitale/analogica. In questo caso il segnale digitale convertito in analogico diventa un segnale a gradini, in quanto, dopo ogni campione e fino al campione successivo, viene applicato un meccanismo di *sample and hold* (in italiano "campiona e mantieni", vedi 4.7T e la nota 14 del par. 3.4P) che mantiene fisso il valore della tensione elettrica fino al campione successivo (non essendo presenti altri valori fra un campione e l'altro). Mediante questo sistema il segnale costituito da campioni viene convertito in un segnale continuo. A causa di questi gradini (che non sono presenti nel segnale analogico originale) si ha una modificazione del segnale dovuta all'alterazione della sua forma d'onda, con conseguente introduzione di componenti non presenti nel segnale originale. Queste componenti, dette *alias*, che formano più immagini dello stesso spettro attorno alla frequenza di sovracampionamento, hanno ora un profilo decrescente al crescere della frequenza. Nella figura 5.11 possiamo osservare come il meccanismo di *sample and hold* riduca le componenti ad alta frequenza. Più precisamente, nella parte alta della figura vediamo un campionamento ideale effettuato con impulsi di durata infinitesimale: lo spettro risultante si estende all'infinito senza perdita di ampiezza delle componenti. Nella parte bassa vediamo il campionamento realizzato tramite *sample and hold* che comporta, come abbiamo detto, una progressiva riduzione dell'ampiezza delle repliche dello spettro[3].

[3] Le repliche dello spettro in un campionamento *sample and hold* decrescono secondo la funzione sen(x)/x (vedi fig. 5.11, grafico in basso a destra).

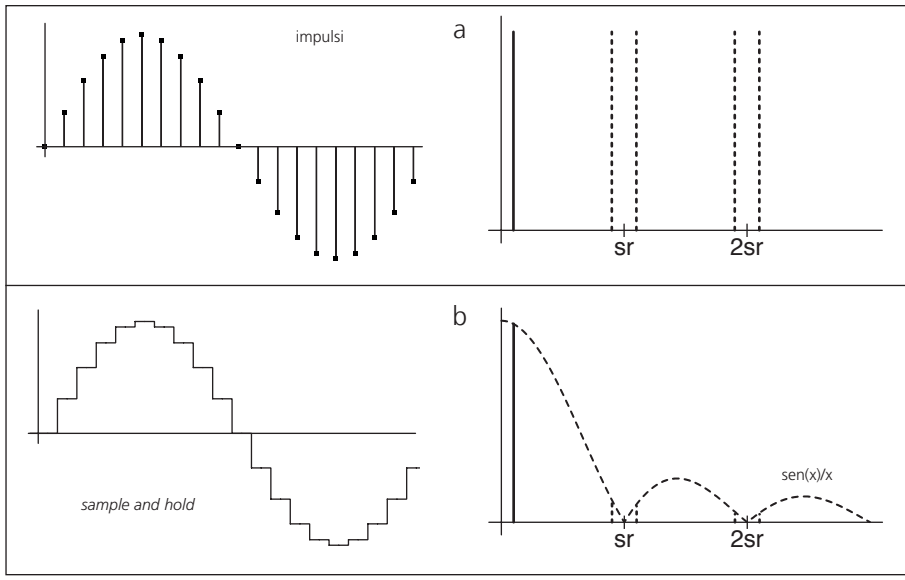


fig. 5.11: campionamento e *aliasing*

Naturalmente anche in questo caso, se lo spettro è composto da più parziali avremo repliche degli spettri con ampiezze decrescenti al crescere della frequenza.

(...)

il capitolo prosegue con:

Foldover: le cause
Il processo di conversione nel dettaglio
Schede audio, segnali digitali e analogici
Compressione dei dati audio

5.2 QUANTIZZAZIONE e DECIMAZIONE
Quantizzazione, rumore di quantizzazione e dithering
Decimazione, downsampling e riduzione dei bit

5.3 USO DEI SUONI CAMPIONATI: IL CAMPIONATORE E TECNICA DEL LOOPING
Il campionatore
Tecnica del looping e altre funzioni del campionatore
Rimozione del DC offset

5.4 SEGMENTAZIONE DI SUONI CAMPIONATI: TECNICA DEI BLOCCHI E SLICING
Tecnica dei blocchi (blocks technique)
Oltre la tecnica dei blocchi:
posizione del cue random o stabile
Slicing

5.5 MANIPOLAZIONE DEL PITCH NEI SUONI CAMPIONATI: AUDIO SCRUBBING

- ESEMPI SONORI E INTERATTIVI
- TEST A RISPOSTE BREVI
- TEST CON ASCOLTO E ANALISI
- CONCETTI DI BASE - GLOSSARIO

5P
AUDIO DIGITALE E SUONI CAMPIONATI

PREREQUISITI PER IL CAPITOLO

- Contenuti del volume 1 (teoria e pratica) e del capitolo 5 (teoria)

OBIETTIVI

Abilità

- Saper gestire le impostazioni globali dell'audio di MSP
- Saper registrare un suono
- Saper utilizzare suoni campionati e modificarne ampiezza, frequenza e numero di bit
- Saper gestire la lettura di un suono in reverse
- Saper gestire creativamente i loop di suoni campionati
- Saper importare in una tabella un suono campionato e servirsi di tale tabella per la generazione del suono
- Saper gestire creativamente la decimazione di un suono

Competenze

- Saper realizzare un breve studio basato sull'uso di suoni campionati, con utilizzo di loop, reverse, diversi punti di inizio della lettura dei file, inviluppi e glissandi dei suoni campionati etc.

CONTENUTI

- Impostazioni Audio in MSP
- Foldover
- Riduzione dei bit e decimazione
- Rumore di quantizzazione e dithering
- Acquisizione e sistemi di lettura di suoni campionati
- La costruzione di un campionatore
- Tecnica dei blocchi e slicing
- Audio scrubbing e programmazione di un random scrubber

ATTIVITÀ

- Costruzione e modifiche di algoritmi

SUSSIDI DIDATTICI

- Lista oggetti Max - Lista attributi e messaggi per oggetti Max specifici - Glossario

5.1 IL SUONO DIGITALE

IMPOSTAZIONI GLOBALI DELL'AUDIO IN MSP

Per gestire le impostazioni globali dell'audio e scegliere la scheda audio con cui MSP deve comunicare ci si può servire della finestra *Audio Status* che si trova nel menù *Options* (fig. 5.1).

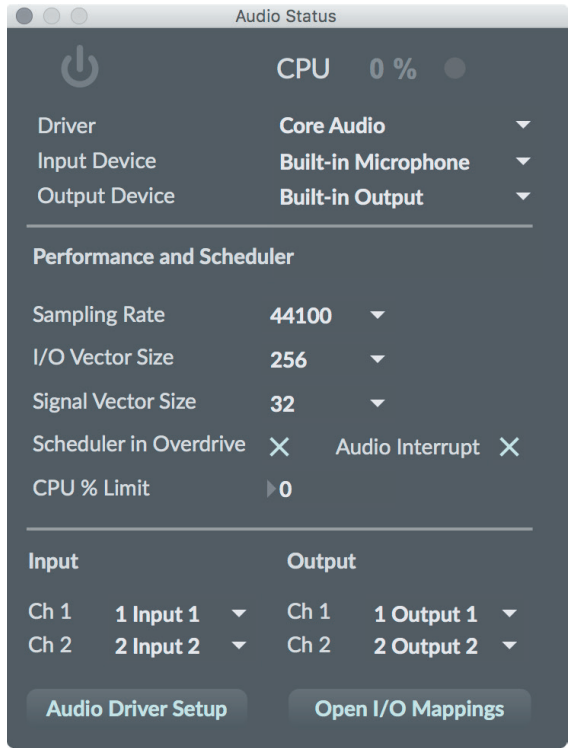


fig. 5.1: la finestra *Audio Status*

Vedremo ora le caratteristiche principali di questa importante finestra. Non è necessario memorizzare tutte le informazioni che riportiamo qui di seguito, potrete ritornare a questo paragrafo ogni volta che avrete bisogno di informazioni sulla finestra *Audio Status*.

La finestra è divisa in tre riquadri ciascuno dei quali contiene un gruppo di parametri. Il primo riquadro riguarda in particolare la scheda audio, vediamone i parametri principali: nella parte alta della finestra abbiamo a sinistra un pulsante per avviare il "motore" DSP. Nella parte destra un valore (CPU) che indica in percentuale il lavoro che il processore del computer deve svolgere per eseguire l'algoritmo della *patch* attiva. Ovviamente una stessa *patch* necessiterà di percentuali diverse su computer di diversa potenza. Quando una *patch* richiede una percentuale pari o superiore al 95%, il computer diventa difficile da gestire e risponde lentamente ai comandi: si tratta quindi di un parametro da tenere d'occhio.

Driver: scelta dell'*audio driver* utilizzato da MSP. Qui si può scegliere il *driver* di una scheda audio presente nel computer (in figura vediamo che è stato scelto il *driver* "*CoreAudio*" di un computer Mac), o selezionare la voce "ad_rewire" per inviare il segnale ad un'applicazione compatibile con il protocollo *ReWire*[1]. Si può anche selezionare l'opzione "NonRealTime" per generare un segnale non in tempo reale, molto utile se l'algoritmo di generazione è troppo complesso per poter essere gestito in tempo reale dal computer; il segnale naturalmente andrà salvato su disco e potrà essere ascoltato alla fine dell'elaborazione. Infine, selezionando l'opzione "Live" i possessori di Max For Live[2] possono mandare direttamente il segnale al programma Ableton Live.

Input Device, Output Device: questi due parametri permettono di specificare i dispositivi di ingresso e di uscita. Le loro funzioni dipendono dal tipo di *driver* e schede audio utilizzate.

Il secondo riquadro ci permette di regolare il rapporto tra efficienza e latenza nell'elaborazione audio e di impostare la precisione temporale dei messaggi Max:

Sampling Rate: qui possiamo impostare la frequenza di campionamento. Le frequenze disponibili variano da scheda a scheda.

I/O Vector Size: il segnale non passa da MSP alla scheda audio un campione alla volta, ma a gruppi, o vettori. La dimensione in campioni del vettore input/output può essere impostata qui. Più il vettore è piccolo, minore è la latenza (il ritardo) tra l'input e l'output. D'altra parte l'elaborazione di ogni vettore ha un certo costo computazionale; ciò significa che utilizzando un *vector size* molto piccolo si incrementerà la percentuale di CPU necessaria rispetto a un *vector size* grande, in quanto dovranno essere calcolati più vettori al secondo. Come se non bastasse anche un vettore troppo grande può creare dei problemi, in quanto MSP potrebbe non riuscire a calcolarlo tutto nel tempo a disposizione e questo può generare dei clic all'uscita audio. Si consiglia una regolazione non superiore a 256 campioni, ma il *range* dei valori possibili dipende dalla scheda audio (alcune schede potrebbero ad esempio avere un I/O delay minimo di 512).

Signal Vector Size: questo parametro indica il numero di campioni che la *patch* MSP elabora in una volta. Anche in questo caso più grande è il vettore, minore è il costo computazionale. A differenza dell'altro vettore questo non ha nessuna influenza diretta sulla latenza, e non può essere impostato a valori superiori a quelli dell'*I/O Vector Size*. Per certi oggetti (che vedremo al momento opportuno) può essere comunque utile impostare valori bassi. Si consiglia una regolazione compresa tra 16 e 128 campioni.

Scheduler in Overdrive: questa opzione è impostabile anche tramite il menù *Options*. Quando Max è in *overdrive* dà la priorità agli eventi temporizzati

[1] Per il protocollo *ReWire* vedi http://www.propellerheads.se.

[2] Parleremo di Max For Live alla fine di questo volume.

(come ad esempio i *bang* prodotti da `metro`) e ai messaggi MIDI che riceve, rispetto ad altri compiti secondari come aggiornare l'aspetto grafico della *patch* o rispondere all'input del mouse o della tastiera del computer. Questo significa che Max sarà ritmicamente molto più preciso, ma rischia di non rispondere più ai comandi da tastiera o al mouse se i dati temporizzati che deve gestire sono troppi. I segnali MSP hanno comunque sempre la priorità sui messaggi Max.

Audio Interrupt: questa opzione è disponibile solo quando Max è in modalità *overdrive*. Quando è impostata fa sì che gli eventi temporizzati vengano prodotti immediatamente prima del calcolo di ogni *Signal Vector*. Questo permette di sincronizzare in modo molto più preciso l'audio con i messaggi Max[3]. In questo caso bisogna però scegliere un *Signal Vector Size* abbastanza piccolo (meno di 64 campioni) altrimenti i messaggi Max rischiano di essere prodotti a tempi sensibilmente differenti da quelli attesi; questo avviene perché Max deve "aspettare" che MSP produca il suo *Signal Vector* prima di poter generare il messaggio temporizzato. Se il *Signal Vector* fosse, poniamo, lungo 1024 campioni e la frequenza di campionamento fosse di 44100 Hz, i vettori verrebbero calcolati ogni 23 millisecondi circa, e una sequenza di eventi Max verrebbe riprodotta a "scatti" successivi di 23 millisecondi. Con un *Signal Vector* di 16 campioni invece l'intervallo tra due vettori è meno di mezzo millisecondo, e ciò rende i piccoli ritardi che Max comunque potrebbe avere assolutamente impercettibili.

CPU % Limit: qui è possibile impostare la percentuale di CPU che MSP può utilizzare (il valore pari a zero equivale a "nessun limite"). Può essere utile per evitare che il programma si "impossessi" di tutte le risorse.

Il terzo riquadro ci permette di impostare alcuni importanti parametri utilizzati da MSP e dalla scheda audio per l'elaborazione del segnale:

Input Channels, Output Channels: il numero dei canali in ingresso e in uscita utilizzati dalla scheda.
È inoltre possibile impostare tramite quattro menù i due canali di ingresso e i due canali di uscita della scheda audio utilizzata. Per impostare gli eventuali altri canali bisogna fare clic sul pulsante "I/O Mappings" in basso a destra.
Il pulsante in basso a sinistra, "Audio Driver Setup", ci permette di accedere alle preferenze della scheda utilizzata.

[3] Questa opzione è molto utile in *patch* in cui i messaggi Max attivano la produzione di suoni MSP come ad esempio nella *patch* **IB_04_sequenza.maxpat** di cui abbiamo parlato nell'Interludio B del primo volume: vi consigliamo di tenerla attivata anche per le *patch* che illustreremo nei paragrafi successivi di questo capitolo.

FOLDOVER

Facendo riferimento al paragrafo 5.1T della parte teorica, vediamo che cosa succede quando portiamo la frequenza di un oscillatore sinusoidale al di sopra della frequenza di Nyquist; ricostruite la *patch* di figura 5.2.

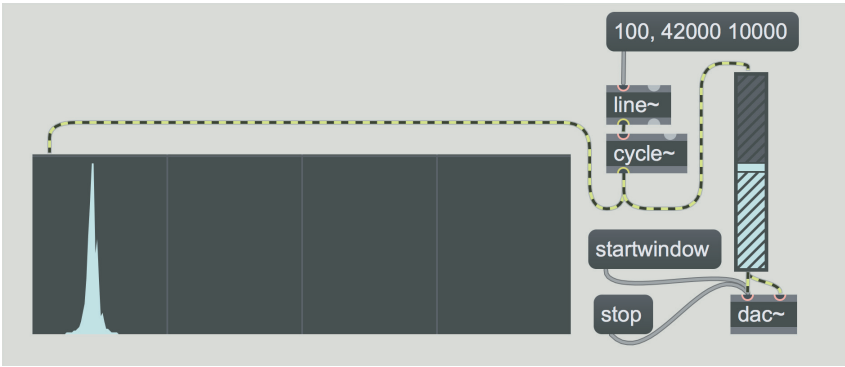


Fig. 5.2: glissando oltre la frequenza di Nyquist

L'oscillatore `cycle~` è connesso all'oggetto `spectroscope~` che come sappiamo ci permette di visualizzare lo spettro di un segnale. Con un clic sul *message box* connesso all'oggetto `line~` eseguiamo un glissando (da 100 a 42000 Hz in 10 secondi) che supera la frequenza di Nyquist.

La frequenza di arrivo indicata in figura (42000 Hz) presuppone una frequenza di campionamento (sr) di 44100 Hz. Se la vostra scheda ha una diversa frequenza (verificatelo nella finestra *Audio Status*) dovreste portare la frequenza di campionamento a 44100 Hz o impostare una diversa frequenza di arrivo: ad esempio per una sr di 48000 Hz l'arrivo dovrebbe essere a circa 46000 Hz, mentre per una sr di 96000 Hz l'arrivo dovrebbe essere a circa 94000 Hz e così via. Eseguendo la *patch* vedremo nello spettroscopio la frequenza della sinusoide che sale fino alla frequenza di Nyquist e poi "rimbalza" all'indietro per fermarsi alla frequenza riflessa di 2100 Hz (in realtà -2100 Hz), che corrisponde alla formula che abbiamo visto nel paragrafo 5.2 della teoria:

fc	**- sr**	**=**	**freq. generata**
42000 Hz	**- 44100 Hz**	**=**	**-2100 Hz**

Sostituiamo adesso la sinusoide con un oscillatore a dente di sega (fig. 5.3).
Come oscillatore usiamo l'oggetto `phasor~` che genera una rampa da 0 a 1, con un paio di semplici operazioni aritmetiche facciamo in modo che generi una rampa da -1 a 1 (come avevamo già fatto nel primo capitolo).
Abbiamo abbassato la frequenza di arrivo a 21500 Hz perché non è necessario che la fondamentale superi la frequenza di Nyquist, basterà che a farlo siano le armoniche di cui il segnale è ricco (regolate comunque la frequenza di arrivo in relazione alla frequenza di campionamento della vostra scheda).

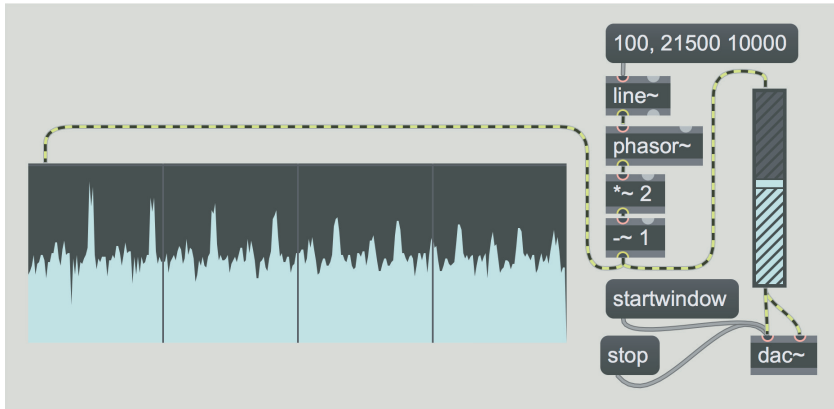


Fig. 5.3: glissando di una forma d'onda non limitata in banda

Se attiviamo la *patch* sentiremo una "pioggia" di armoniche che rimbalza sul muro di Nyquist mentre viene eseguito il glissando ascendente; le armoniche riflesse raggiungono rapidamente la frequenza di 0 Hz e rimbalzano nuovamente verso l'alto, per poi tornare ancora verso il basso e così via conferendo un aspetto zigzagante allo spettro del suono che glissa: dopo 10 secondi lo spettro si stabilizza in una configurazione inarmonica.
Lo spettro di questa forma d'onda è ricchissimo di armoniche, essendo una dente di sega molto simile a quella ideale (che contiene, come sappiamo, un numero infinito di componenti). Questo significa che è praticamente impossibile mandare all'uscita un segnale prodotto dall'oggetto `phasor~` senza generare un *foldover*[4].
Si possono eliminare le parziali che superano il valore di Nyquist applicando al generatore non limitato in banda un filtro passa-basso, ad esempio a 20000 Hz? No, perché le parziali riflesse dal *foldover* all'interno della banda riproducibile, sono del tutto indistinguibili da un suono effettivamente generato a quella frequenza, e il filtro arriverebbe solo "dopo" tale riflessione. Ad esempio, presupponendo una frequenza di campionamento di 48000 Hz, se generiamo con un oggetto `phasor~` un'onda a dente di sega non limitata in banda con fondamentale a 10000 Hz abbiamo, per le prime armoniche, la seguente situazione:

Fondamentale: 10000 Hz
II parziale: 20000 Hz
III parziale: 30000 Hz = 24000 - (30000 - 24000) = 18000 Hz
IV parziale: 40000 Hz = 24000 - (40000 - 24000) = 8000 Hz
etc.

La III e IV parziale superano la frequenza di Nyquist e pertanto vengono riflesse rispettivamente alla frequenza di 18000 e 8000 Hz. Questi suoni riflessi sono

[4] In realtà, quando la fondamentale ha una frequenza molto bassa, sono solo le armoniche più lontane che superano la frequenza di Nyquist, e queste armoniche, come abbiamo visto nel paragrafo 2.1 della teoria, sono estremamente deboli. In questo caso il *foldover* è trascurabile.

effettivamente a 18000 e 8000 Hz già all'uscita dell'oggetto **`phasor~`** e un filtraggio a 20000 Hz non avrebbe alcun effetto. Per filtrarli dovremmo utilizzare un filtro passa-basso con frequenza di taglio inferiore agli 8000 Hz, ma in questo modo elimineremmo anche la fondamentale e la seconda parziale! Provate ad esempio ad applicare il filtro passa-basso **`vs.butterlp~`** alla *patch* di figura 5.3, e potrete rendervi conto che il segnale viene filtrato solo dopo che si è verificato il *foldover*.

Nel paragrafo 2.1 della parte pratica abbiamo visto come, tramite l'oggetto **`vs.buf.gen10`**, sia possibile generare delle approssimazioni di forme d'onda ideali (come ad esempio quella a dente di sega) che contengano un numero limitato (non infinito) di armoniche. Approssimando, con il metodo illustrato in quel paragrafo, un'onda a dente di sega con 20 armoniche, e supponendo una frequenza di campionamento di 44100 Hz, è possibile generare un'oscillazione fino a circa 1102 Hz senza *foldover* (a 1102 Hz la 20ma armonica si trova a 22040 Hz, poco sotto la frequenza di Nyquist): al di sopra di questo limite le componenti cominceranno a rimbalzare indietro.
Esiste però anche un gruppo di oscillatori *limitati in banda* (che abbiamo già conosciuto nel paragrafo 1.2 della parte pratica), che generano le forme d'onda "classiche" (dente di sega, triangolare e quadrata) e che ci permettono di utilizzare qualsiasi frequenza al di sotto di quella di Nyquist senza provocare *foldover*. In pratica queste forme d'onda non sono scritte in una tabella fissa, ma vengono generate mediante un algoritmo che limita il numero di armoniche in base alla frequenza fondamentale: se questa è, poniamo, di 100 Hz la forma d'onda avrà circa 200 armoniche, se è di 1000 Hz ne avrà circa 20 e così via (consideriamo sempre una frequenza di campionamento di 44100 Hz, per altre frequenze il numero delle componenti varia in relazione).
Proviamo a inserire, nella nostra *patch*, l'oggetto **`saw~`**, che genera appunto un'onda a dente di sega limitata in banda, al posto del **`phasor~`** e dei due moduli aritmetici collegati (fig. 5.4).

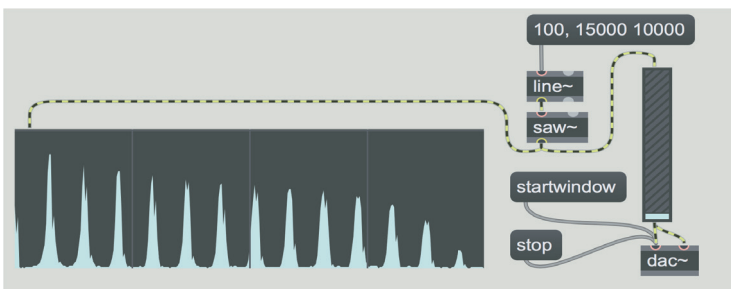


Fig. 5.4: lo spettro di una forma d'onda limitata in banda

Se eseguiamo la *patch* vedremo nello spettroscopio che le armoniche "svaniscono" mano a mano che si avvicinano alla frequenza di Nyquist: alla fine, quando l'oscillatore supera la frequenza di 11025 (un quarto della frequenza di campionamento, la metà di quella di Nyquist) non resta che una singola componente, la fondamentale, perché già la seconda armonica (che è superiore a 11025 * 2, ovvero superiore alla frequenza di Nyquist) genererebbe il *foldover*.

5.2 QUANTIZZAZIONE E DECIMAZIONE

RIDUZIONE DEI BIT, RUMORE DI QUANTIZZAZIONE E DITHERING

All'interno di MSP i campioni sono elaborati come numeri in virgola mobile e, a partire dalla versione 6 di Max, hanno una dimensione di 64 bit. Oltre al bit segno abbiamo 11 bit per il fattore di scala e 52 bit per la parte frazionaria. I livelli di quantizzazione sono quindi 2^{53} (sui concetti di virgola mobile, fattore di scala etc. cfr. il paragrafo 5.2T).
Proviamo ora a vedere come possiamo diminuire i livelli di quantizzazione di un segnale prodotto con MSP: questo ci potrà servire, ad esempio, per produrre quelle sonorità *LO-FI* (a bassa fedeltà) che ormai da tempo hanno acquisito una propria dignità estetica.
Ricreate la *patch* di fig. 5.5.

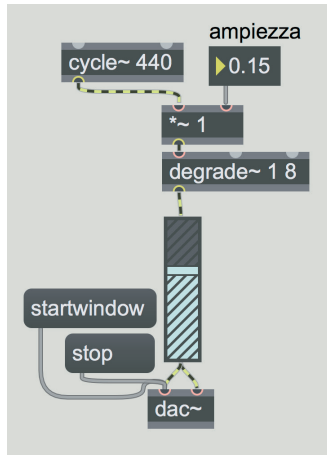


Fig. 5.5: generare un suono a 8 bit

L'oggetto **`degrade~`** serve a diminuire il numero di bit di un segnale e la sua frequenza di campionamento. Questo oggetto ha 2 argomenti, che possono anche essere variati tramite il secondo e il terzo ingresso: il primo argomento indica il fattore di riduzione della frequenza di campionamento[5] e il secondo la dimensione in bit del segnale. Nel caso in figura abbiamo quindi un segnale a 8 bit, equivalente a 256 livelli di quantizzazione. Quando l'ampiezza del segnale (l'oscillatore sinusoidale) è massima, vengono sfruttati tutti i livelli di quantizzazione disponibili, ma il rumore di quantizzazione è già chiaramente apprezzabile. Se abbassate il *number box* che controlla l'ampiezza e lo portate a 0.15, ad esempio, udrete un effetto molto più pronunciato, perché ora il segnale utilizza una parte minore dei livelli di quantizzazione.

[5] Il valore 1 indica una frequenza inalterata rispetto alla frequenza della scheda audio: maggiori dettagli più sotto.

Vediamo ora una *patch* che ci permette di impostare un numero di bit a piacimento e di calcolare i relativi livelli di quantizzazione: aprite il file **05_01_quantizza.maxpat** (fig. 5.6).

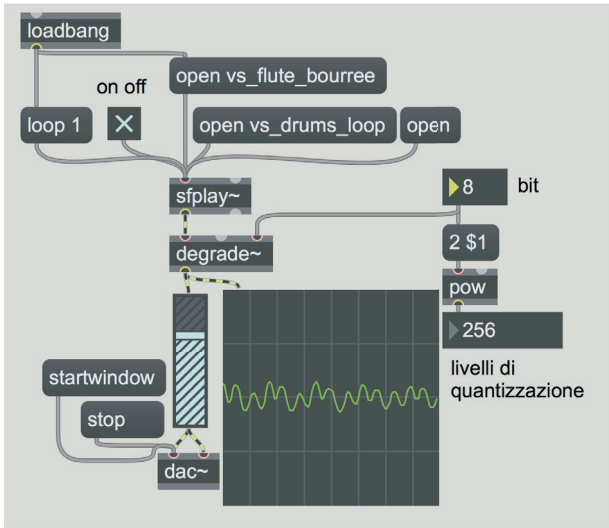


Fig. 5.6: file **05_01_quantizza.maxpat**

Nella parte destra abbiamo l'algoritmo che ci permette di calcolare il numero di livelli di quantizzazione dato il numero di bit. Dal momento che i livelli di quantizzazione sono pari a 2 elevato al numero di bit (sono, in altre parole, potenze di 2), abbiamo utilizzato l'oggetto `pow` che calcola appunto la potenza di un numero. Questo oggetto riceve una lista di due valori: il primo (la base) è un 2 e il secondo (l'esponente) è l'argomento $1 che possiamo variare tra 1 e 24, e che ci permette quindi di calcolare i livelli di quantizzazione tra 1 e 24 bit.
Anche in questo caso per variare il numero di bit del segnale usiamo l'oggetto `degrade~` a cui inviamo il numero di bit desiderato al terzo ingresso.
All'apertura della *patch* l'oggetto `loadbang` attiva 2 *message box* che dicono a `sfplay~` di aprire il file di suono **vs_flute_bourree.wav**[6] e di attivare la modalità *loop*. Per azionare `sfplay~` bisogna fare clic sul `toggle` collegato (dopo aver fatto clic su "startwindow", naturalmente), e a questo punto è possibile modificare il numero di bit e di conseguenza i livelli di quantizzazione. Provate a caricare altri suoni e sentite come vengono "degradati" diminuendo la lunghezza in bit dei campioni: ricordate che ogni volta che caricate un nuovo suono dovete farlo partire riattivando il `toggle`.

[6] Questo file si trova, come gli altri file audio che utilizzeremo, all'interno della libreria Virtual Sound Macros, nella cartella "soundfiles". Notate che non è necessario aggiungere l'estensione del file audio (in questo caso wav). Attenzione però, nel caso in cui ci fossero, nella stessa cartella, due file audio con lo stesso nome ma con un'estensione diversa, Max caricherebbe quello con l'estensione che in ordine alfabetico viene per prima. In altre parole se esistessero due file vs_flute_bourree.wav e vs_flute_bourree.aif verrebbe caricato il file con l'estensione aif.

Per cercare di migliorare il suono ed eliminare per quanto possibile le distorsioni, simuliamo il *dithering*: caricate il file **05_02_dither.maxpat** (fig. 5.7).

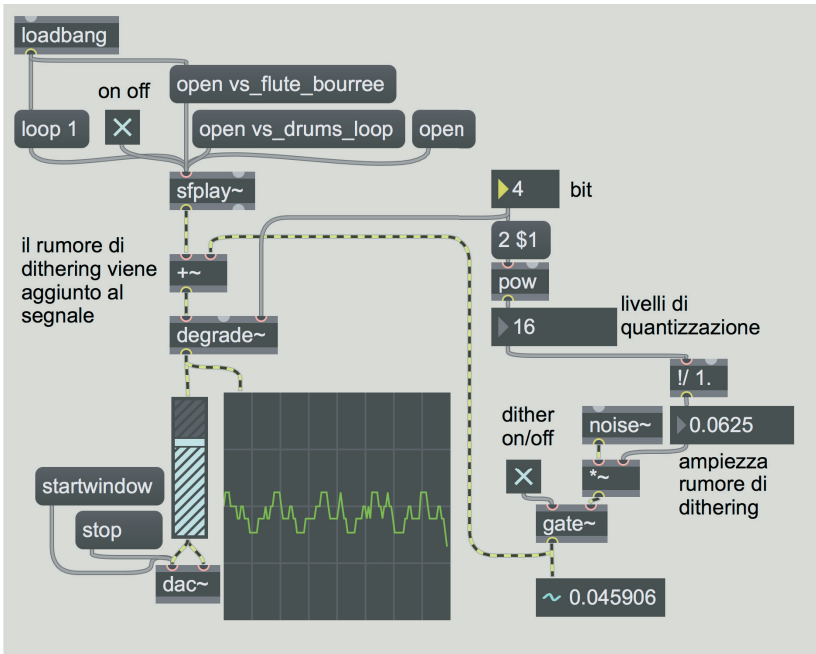


Fig. 5.7: file **05_02_dither.maxpat**

Nella parte destra della *patch* c'è un oggetto `noise~` che produce rumore bianco che viene scalato ad un'ampiezza pari a quella dell'intervallo di quantizzazione. Tramite l'operatore `!/` con argomento "1.", infatti, viene generato un numero che rappresenta l'ampiezza di un singolo livello di quantizzazione: ad esempio quando, come in figura, i livelli di quantizzazione sono 16, l'ampiezza di un singolo livello è pari a 1/16, ovvero 0.0625. Questa ampiezza serve da fattore di moltiplicazione per il rumore bianco.
Il rumore così riscalato viene aggiunto al segnale (vedi la parte sinistra della *patch*), e il risultato della somma viene inviato all'oggetto `degrade~`. È possibile attivare e disattivare il *dithering* con il `toggle` in basso a destra: questo `toggle` è collegato ad un oggetto `gate~` che lascia passare il segnale che entra nel suo ingresso destro se nel suo ingresso sinistro entra un 1 o lo blocca se nel suo ingresso sinistro entra uno 0. Quando il numero dei bit è basso (sotto i 12) il rumore aggiunto dal *dithering* è molto presente, ma indubbiamente contribuisce ad eliminare le parziali prodotte dalla distorsione. Provate ad utilizzare il suono di batteria **vs_drums_loop.aif** con un numero di bit molto basso (ad es. 4): nonostante sia molto rumoroso il *dithering* permette di ricostruire il suono di batteria in modo accettabile (soprattutto considerando che si tratta di campioni a 4 bit), mentre escludendolo il suono appare estremamente distorto.

Per diminuire il rumore del *dithering* possiamo usare dei filtri; aprite il file **05_03_dither_filter.maxpat** (fig. 5.8).

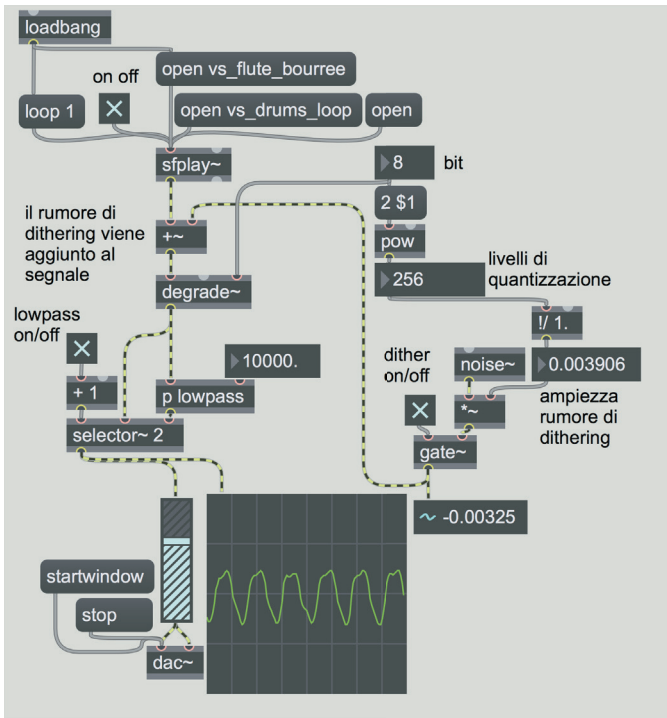


Fig. 5.8 file **05_03_dither_filter.maxpat**

(...)

il capitolo prosegue con:

Interludio C
GESTIONE DEL TEMPO, POLIFONIA, ATTRIBUTI E ARGOMENTI

PREREQUISITI PER IL CAPITOLO

- Contenuti del volume 1 e del capitolo 5 (teoria e pratica)

OBIETTIVI

Abilità

- Saper gestire i diversi tipi di durate, scansioni metriche e valori di tempo, incluso il sistema di scansione globale del tempo nell'ambiente Max
- Saper costruire e gestire uno step sequencer
- Saper gestire la polifonia in Max
- Saper gestire attributi e argomenti nelle abstraction

CONTENUTI

- Durate, scansioni metriche e valori di tempo in Max
- Sistema di scansione globale del tempo in Max
- Argomenti e attributi
- Algoritmi per lo step sequencer
- Patch polifoniche
- Abstraction e argomenti

SUSSIDI DIDATTICI

- Lista oggetti Max - Lista attributi, messaggi ed elementi grafici per oggetti Max specifici - Glossario

IC.1 COME SCORRE IL TEMPO (IN MAX)

Premessa: anche per questo capitolo vi suggeriamo di attivare, nella finestra *Audio Status*, le opzioni "Scheduler in Overdrive" e "in Audio Interrupt", e di impostare un *Signal Vector Size* compreso tra 16 e 64 campioni (preferibilmente 16).

Finora abbiamo usato, come unità di misura del tempo, i millisecondi (ad esempio per indicare l'intervallo tra i *bang* generati dall'oggetto `metro`) o i campioni (ad esempio per indicare il ritardo di un segnale audio nell'oggetto `delay~`[1]). Con Max è possibile utilizzare altre unità di misura, e soprattutto è possibile sincronizzare diversi oggetti tra loro grazie ad un ***master clock*** (un sistema di scansione globale del tempo nell'ambiente Max) controllato dall'oggetto **`transport`** che consente, fra l'altro, di attivare e disattivare il passaggio del tempo globale. Ricostruite la *patch* di figura IC.1.

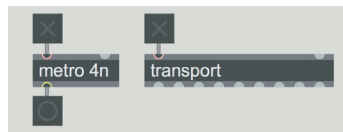


fig. IC.1: l'oggetto `transport`

Notate che l'argomento dell'oggetto `metro` non è un valore in millisecondi, ma un simbolo: *4n* (il cui significato vedremo tra poco). Questo fa sì che il funzionamento dell'oggetto `metro` sia legato al *master clock*.
Per prima cosa attivate il `toggle` al di sopra dell'oggetto `metro`: contrariamente alle aspettative l'oggetto non emette alcun *bang*. Ora attivate anche il `toggle` al di sopra dell'oggetto `transport`: l'oggetto `metro` comincia a produrre dei *bang*.
L'oggetto `transport` attiva il *master clock* che a sua volta attiva l'oggetto `metro`; per quest'ultimo, in pratica, il tempo "scorre" solo quando il *master clock* è attivo.
Qual è la velocità di scansione di `metro`? Il simbolo 4n che abbiamo usato come argomento è un *valore di tempo relativo*, e precisamente è un *note value* (valore di nota), che indica che la scansione corrisponde alla durata di una nota da un quarto (ovvero una semiminima). L'oggetto `metro`, quindi, genera un *bang* ogni semiminima.
Quanto dura una semiminima? La durata di una semiminima dipende dall'oggetto `transport`, o meglio dall'attributo **"tempo"** dell'oggetto. Questo attributo esprime il numero di battiti (o pulsazioni) per minuto[2] (abbreviato in bpm) e di *default* il suo valore è 120 bpm (120 pulsazioni al minuto, ovvero una pulsazione ogni mezzo secondo).
Il tempo metronomico può essere variato inviando all'oggetto `transport` il messaggio "tempo" seguito dal valore in bpm.

[1] Ne abbiamo parlato nel primo volume, al paragrafo 3.6P.
[2] Si tratta quindi del classico tempo di metronomo.

Modifichiamo la *patch* precedente nel modo illustrato in figura IC.2.

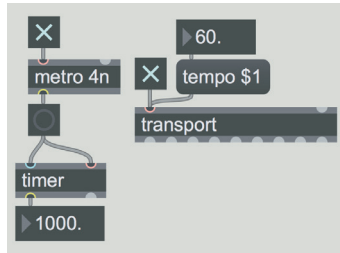


fig. IC.2: modifica del tempo di metronomo

Oltre al messaggio per modificare il tempo in bpm di `transport`, abbiamo aggiunto un oggetto `timer` che ci permette di calcolare l'intervallo (in millisecondi) tra un *bang* e il successivo. Come vedete, portando il tempo metronomico a 60 bpm, viene generato un *bang* al secondo (1000 millisecondi). Provate a variare il tempo in bpm e osservate come cambia il tempo calcolato dall'oggetto `timer` (notate che nella *patch* l'oggetto `timer` ha entrambi gli ingressi collegati al *button* soprastante).
L'oggetto **`timer`** è un "cronometro" che si avvia quando riceve un *bang* all'ingresso di sinistra, e genera all'uscita il tempo trascorso quando riceve un *bang* all'ingresso di destra. Contrariamente alla grande maggioranza degli oggetti Max, quindi, l'ingresso "caldo" di `timer` è quello destro.

ATTIVITÀ

Create una nuova *patch* per fare delle prove con l'oggetto `timer` inviando dei *bang* separatamente all'ingresso sinistro e destro; osservate i valori all'uscita di `timer`. Poi spiegate perché, come si vede in fig. IC.2, collegando uno stesso *bang button* a entrambi gli ingressi di un `timer`, è possibile calcolare il tempo trascorso tra un *bang* e il successivo.

Ci sono ovviamente diversi simboli per indicare i principali *note value*. Questi inoltre vengono suddivisi in unità minime chiamate *ticks*: una semiminima, ovvero una nota da un quarto, è identificata come abbiamo visto dal simbolo 4n, e si suddivide in 480 *ticks*. Questo significa che una nota da un ottavo (simbolo 8n) corrisponde a 240 *ticks*, una da un sedicesimo (simbolo 16n) a 120 *ticks* e così via. Anche la durata dei *ticks*, ovviamente, dipende dal tempo di metronomo dell'oggetto `transport`.

Ecco ora una corrispondenza tra simboli di *note value*, valore frazionario e valore in *ticks*:

1nd Intero (Semibreve) col punto - 2880 *ticks*
1n Intero - 1920 *ticks*

1nt Terzina di interi - 1280 *ticks*
2nd Metà (Minima) col punto - 1440 *ticks*
2n Metà - 960 *ticks*
2nt Terzina di metà - 640 *ticks*
4nd Quarto (Semiminima) col punto - 720 *ticks*
4n Quarto - 480 *ticks*
4nt Terzina di quarti - 320 *ticks*
8nd Ottavo (Croma) col punto - 360 *ticks*
8n Ottavo - 240 *ticks*
8nt Terzina di ottavi - 160 *ticks*
16nd Sedicesimo (Semicroma) col punto - 180 *ticks*
16n Sedicesimo - 120 *ticks*
16nt Terzina di sedicesimi - 80 *ticks*
32nd Trentaduesimo (Biscroma) col punto - 90 *ticks*
32n Trentaduesimo - 60 *ticks*
32nt Terzina di trentaduesimi - 40 *ticks*
64nd Sessantaquattresimo (Semibiscroma) col punto - 45 *ticks*
64n Sessantaquattresimo - 30 *ticks*
128n Centoventottesimo (Fusa) - 15 *ticks*

Nella *patch* di figura IC.2 sostituite l'argomento 4n di `metro` con gli altri *note value*, e verificate i tempi di scansione dei *bang* con l'oggetto `timer`.
Modificate ora la *patch* come illustrato in figura IC.3.

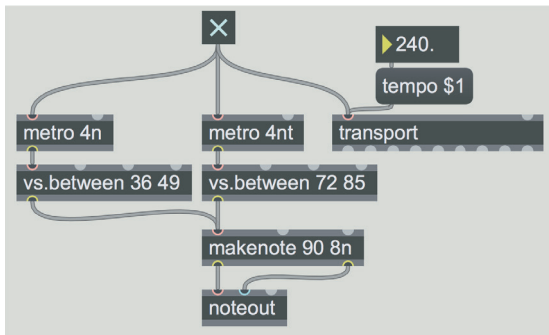


fig. IC.3: due note contro tre

Abbiamo aggiunto un secondo oggetto `metro` il cui intervallo di scansione è pari alla terzina di semiminima (4nt): le due diverse scansioni servono a generare due flussi casuali di note MIDI. Nel tempo in cui vengono generate due note nel registro basso, ne vengono generate tre nel registro acuto. Notate che il secondo argomento dell'oggetto `makenote`, corrispondente alla durata, è 8n, ovvero un ottavo (croma): anche l'oggetto `makenote` (di cui abbiamo parlato nel par. IB.1), infatti, può utilizzare i valori di tempo relativi.
Usiamo un unico `toggle` per avviare i due `metro` e il `transport` per essere sicuri che i due flussi partano contemporaneamente. Cambiate il tempo in bpm dell'oggetto `transport` e notate che i due flussi rimangono sempre sincronizzati.

Attenzione! Il tempo in bpm, e tutti gli altri comandi che inviate a `transport` sono globali, cioè valgono per l'intero ambiente Max. Se avete due *patch* aperte, ed entrambe utilizzano i valori di tempo relativo, attivando e disattivando il *master clock* o modificando il tempo in bpm in una delle due *patch*, attiveremo e disattiveremo il *master clock* e modificheremo il tempo anche per l'altra.
È possibile anche eliminare l'oggetto `transport` dalla *patch* e utilizzare una finestra indipendente (chiamata ***Global Transport***) che contiene i comandi per il *master clock* (questa finestra si visualizza richiamando la voce *GlobalTransport* dal menù *Extras*). Tramite questa finestra possiamo attivare e disattivare il *master clock*, cambiare il tempo in bpm, visualizzare il tempo trascorso, etc.
Torniamo alla figura IC.3: provate a cambiare l'argomento del primo `metro` in 8n. Ora ogni 4 note nel registro basso ne sentiamo 3 nel registro acuto, perché 4 ottavi (crome) equivalgono a una terzina di quarti (semiminime).

ATTIVITÀ

Indichiamo il rapporto metrico dei due flussi di note casuali di fig. IC.3 con 2/3 (ovvero due note contro tre). Modificate gli argomenti dei due oggetti `metro` in modo da ottenere due flussi in rapporto 3/4, 3/8, 2/6, 4/9 (quest'ultimo non è immediato: dovete usare le note puntate per il numeratore).

Con i simboli disponibili per i *note value* si possono creare solo alcuni rapporti metrici; ma, per fare un esempio, non è possibile avere un rapporto 4/5 (quattro note contro cinque). Per ottenere questo ed altri rapporti metrici è necessario usare i *ticks*. Nel caso del rapporto 4/5, dal momento che la semiminima normale corrisponde a 480 *ticks*, una semiminima in una quintina corrisponde a 480 * 4 / 5 = 384 *ticks*. Questo valore non ha un simbolo corrispondente nella lista dei *note value*. Non è possibile indicare il valore in *ticks* direttamente come argomento di `metro`, ma è necessario utilizzare l'attributo **"interval"** che si può impostare nell'*inspector* o inviare come messaggio seguito da un valore e dall'unità di misura. Modificate la *patch* come in figura IC.4.

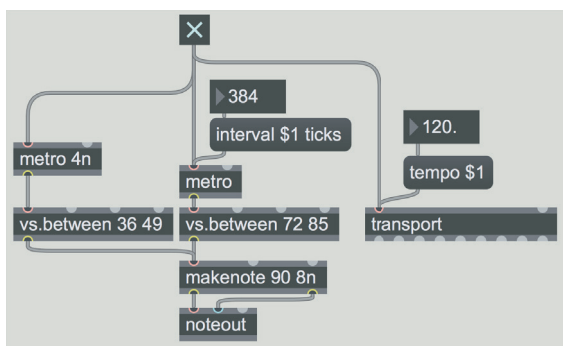


fig. IC.4: quattro note contro cinque

Tramite un *message box* inviamo al secondo `metro` il messaggio "interval 384 ticks", che corrisponde, come abbiamo visto sopra, alla durata di una semiminima all'interno di una quintina.

ATTIVITÀ

- Utilizzando l'attributo "interval" per entrambi gli oggetti `metro` realizzate questi rapporti metrici: 3/10, 5/9, 15/8.

- Fate fare a Max i calcoli: modificate la *patch* precedente in modo che sia sufficiente inserire in due *number box* i valori del rapporto metrico (ad es. 4 e 3), e lasciate che sia la *patch* i a calcolare gli esatti valori in *ticks*.
Suggerimento: è sufficiente aggiungere un oggetto per ogni `metro`.

(...)

il capitolo prosegue con:

Argomenti e attributi
Valori di tempo

IC.2 REALIZZIAMO UNO STEP SEQUENCER

IC.3 LA POLIFONIA

IC.4 ABSTRACTION E ARGOMENTI

- Lista oggetti MAX
- Lista attributi, messaggi ed elementi grafici per oggetti Max specifici
- Glossario

6T
LINEE DI RITARDO: ECO, LOOP, FLANGER, CHORUS, FILTRI COMB E ALLPASS, PHASER, PITCH SHIFTING, REVERSE, ALGORITMO DI KARPLUS-STRONG

PREREQUISITI PER IL CAPITOLO

- Contenuti del volume 1 e del capitolo 5 (teoria)

OBIETTIVI

Conoscenze

- Conoscere l'uso di linee di ritardo utilizzate a diversi scopi
- Conoscere l'uso di simulazioni di eco singole e multiple
- Conoscere l'uso del loop e dello slapback, multitap e ping-pong delay
- Conoscere i parametri e l'uso del flanger, del chorus e del phaser
- Conoscere la teoria di base e alcune possibili applicazioni dei filtri comb e allpass
- Conoscere l'uso del pitch shifting e del reverse
- Conoscere la teoria di base dell'algoritmo di Karplus-Strong
- Conoscere alcune applicazioni per la simulazione del suono di corde pizzicate e percussioni tramite l'algoritmo di Karplus-Strong

Abilità

- Saper individuare all'ascolto i differenti tipi di eco e saperli descrivere
- Saper individuare all'ascolto le differenze di base fra chorus, phaser e flanger e saperle descrivere
- Saper individuare all'ascolto le modifiche dei parametri base dell'algoritmo di Karplus-Strong e saperli descrivere.

CONTENUTI

- Linee di ritardo
- Vari tipi di eco
- Uso di linee di ritardo per loop, chorus, flanger, phaser, pitch shifting e reverse
- Filtri comb e allpass
- Sintesi mediante algoritmo di Karplus-Strong

ATTIVITÀ

- Esempi sonori e interattivi

VERIFICHE

- Test a risposte brevi
- Test con ascolto e analisi

SUSSIDI DIDATTICI

Concetti di base - Glossario - Discografia

6.1 IL *DELAY TIME*: DAI FILTRI ALL'ECO

Il ritardo del segnale è uno degli strumenti più potenti e versatili che abbiamo a disposizione per la computer music: le tecniche di sintesi ed elaborazione del suono più diverse, dalla sintesi sottrattiva a quella per modelli fisici, dalla costruzione dei riverberi all'applicazione della maggior parte degli effetti "classici" (che affronteremo in questo capitolo), hanno in comune l'utilizzo di un ritardo più o meno lungo del segnale. Nel par. 3.6 abbiamo visto, ad esempio, che per realizzare i filtri digitali è necessario utilizzare delle linee di ritardo: in quel caso il tempo fra il segnale in ingresso e il segnale in uscita dal filtro (*delay time*) è brevissimo, calcolato in singoli campioni (ad es. il tempo che separa un campione dal successivo, ad una frequenza di campionamento di 48000 Hz è di 1/48000 di secondo, cioè 0.00002 secondi ca.). Nel momento in cui, da questa dimensione "microscopica" del *delay time*, si passa ad una dimensione più ampia, di diversi millisecondi, si possono ottenere in uscita dal *delay* effetti diversi. In particolare, in questo capitolo, ci occuperemo del *flanger* (in genere da 1 a 20 msec. con *delay time* continuamente variato) di *chorus* (tipicamente da 20 a 30 msec. con *delay time* continuamente variato), di *slapback delay*, cioè di effetti di raddoppio quasi simultaneo di un suono (da 10 msec a 120 msec ca.) e di diversi effetti di *eco* (da 100 msec. fino a diversi secondi). Inoltre ci occuperemo del filtro *comb* che viene usato in modi diversi, fra cui quello di creare risonanze multiple in rapporto armonico fra loro. Vedremo anche due diverse implementazioni del filtro *allpass* (passa-tutto): la prima viene usata unitamente al *comb* per la creazione dei riverberi (di cui parleremo nel terzo volume); la seconda serve a realizzare l'effetto *phaser*. In figura 6.1 vediamo un grafico indicativo: i valori di tempo servono a dare un'idea approssimativa del ritardo necessario per ottenere i diversi effetti, e vanno considerati come valori puramente indicativi che possono variare sensibilmente in relazione al materiale sonoro in ingresso.

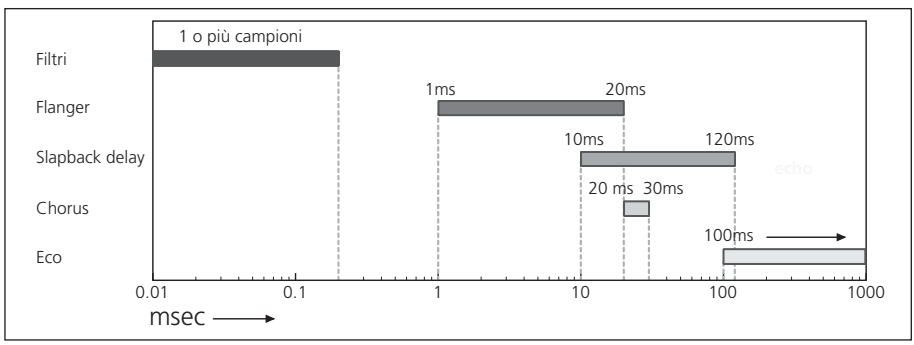


fig. 6.1: diversi tempi di ritardo, dai filtri all'eco

6.2 ECO

L'effetto di **eco**, cioè la ripetizione di un suono, è percepibile chiaramente se la replica del suono ha luogo dopo un tempo superiore a quello definito come *zona di Haas* (25-35 msec.). Se la replica del suono invece arriva all'ascoltatore prima di tale tempo, l'ascoltatore stesso avrà difficoltà a percepire il secondo suono

come suono separato, e nei casi di suoni con attacco lento tenderà a percepire la replica non come un suono a sé stante ma come un suono "fuso" con il primo. Per simulare un semplice effetto di eco abbiamo bisogno di un algoritmo contenente un *delay* (ritardo), il quale riceve in ingresso un segnale audio, e lo replica in uscita dopo un tempo (*delay time*) che possiamo determinare e che può variare da pochi millisecondi a vari secondi. Il suono ritardato poi può essere sommato, oppure no, al suono originario (in genere detto suono *dry*). Se si desidera ottenere solo il suono dell'effetto (in genere definito *wet*), è sufficiente dare ampiezza zero al suono *dry*. La regolazione del rapporto fra la quantità di suono *dry* e di suono *wet* viene chiamata *balance* e viene spesso indicata in percentuale (ad es. 100% significa solo suono *wet*, 50% pari ampiezza al suono *wet* e *dry*, 0% solo suono *dry*). In fig. 6.2 un esempio semplice di *delay*.

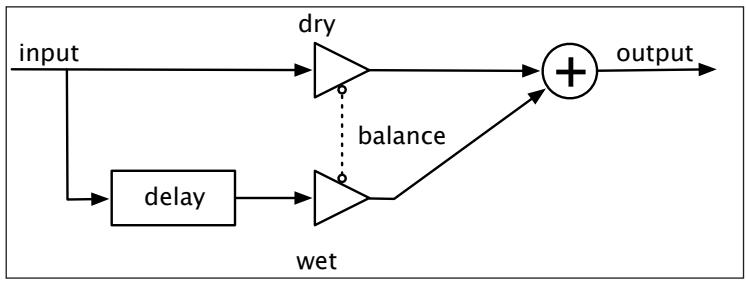


fig. 6.2: esempio semplice di *delay*

ESEMPIO INTERATTIVO 6A.1 • *Effetti eco con* delay time *fuori e dentro zona Haas*

Se le repliche del suono sono più di una otteniamo un'eco multipla. Per ottenere eco multiple abbiamo bisogno di inserire nell'algoritmo un *feedback* cioè la possibilità di rimandare nell'*input* il segnale di uscita del *delay*.
In fig. 6.3 un esempio di *delay* con *feedback*.

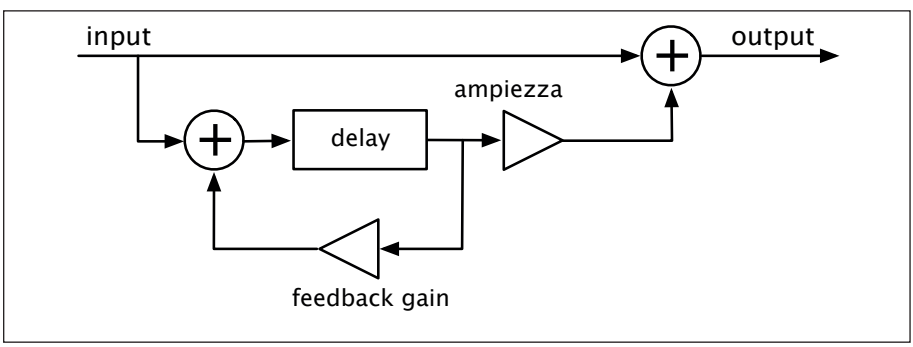


fig. 6.3: *delay* con *feedback*

ESEMPIO INTERATTIVO 6A.2 • *Eco multiple*

In questo modo si può ripetere l'effetto di eco molte volte, regolando il *gain* del *feedback*. Se il *gain* è nullo non avremo alcun *feedback* quindi la ripetizione del segnale avverrà una volta sola, man mano che aumentiamo il *gain* otteniamo un numero sempre maggiore di ripetizioni.
Tecnicamente il *gain* del *feedback* è una moltiplicazione del segnale, e il fattore di moltiplicazione varia tipicamente tra 0 e 1. Con un *gain* al 50% ad esempio il segnale viene moltiplicato per 0.5 (ovvero la sua ampiezza viene ridotta alla metà ad ogni rientro nel circuito): ad esempio, se il moltiplicatore ha valore 0.5 e l'ampiezza iniziale è 1, nei vari riscalamenti del segnale avremo valori uguali a 0.5, 0.25, 0.125 etc.
Con un *gain* allo 0% il segnale viene moltiplicato per 0 (e quindi viene annullato), con un *gain* al 100% viene moltiplicato per 1 (cioè non viene modificato). La percentuale di *gain* infatti indica il rapporto fra l'ampiezza del segnale in uscita e quella del segnale che torna in *input*, quindi, se il valore del moltiplicatore è 0.99 (pari al 99%), il suono in uscita sarà ridotto solo dell'1% ogni volta che viene rinviato all'*input*. Il valore di moltiplicazione 1 (pari al 100%) è sconsigliabile in quanto si ottiene una ripetizione virtualmente infinita alla stessa ampiezza, e se al segnale in ingresso nel frattempo vengono inviati altri segnali audio si rischia la distorsione una volta che diversi segnali entrano nel circuito del *feedback* e le loro ampiezze vengono sommate. Ovviamente valori superiori a 1 sono sconsigliati per gli stessi motivi.

L'ACCUMULO DEL *DC OFFSET*

Può capitare (fig.6.4) che la forma d'onda di un suono contenga un *DC offset* positivo o negativo (vedi par. 5.3).

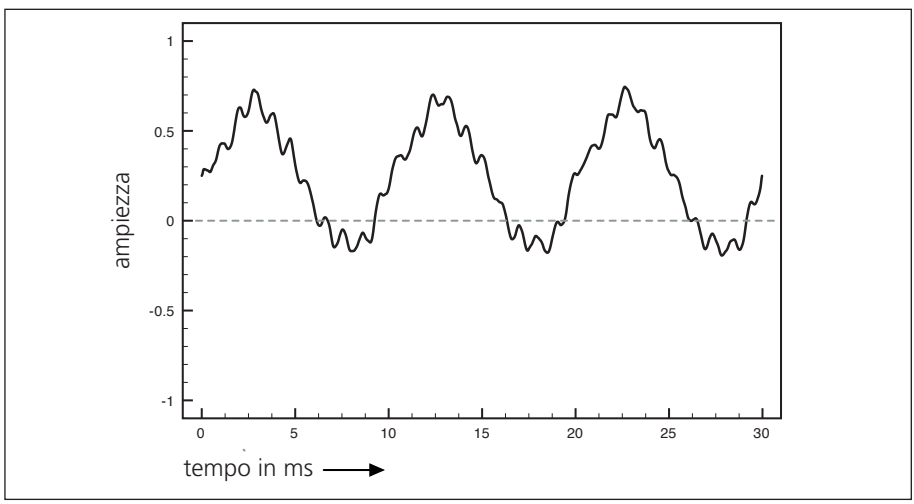


fig. 6.4: suono campionato con un sensibile *DC offset*

Questo sbilanciamento, se lieve, può non essere problematico nella riproduzione diretta del suono. Bisogna però tenere presente che nella realizzazione di un *delay* con *feedback* il *DC offset* delle diverse ripetizioni si accumula e ciò può facilmente portare il suono al di fuori del *range* di ampiezza della scheda audio. In questo caso è necessario intervenire con un filtro passa-alto con frequenza di taglio molto bassa (ad esempio 10 Hz): come abbiamo già spiegato nel par. 5.3 il *DC offset* è infatti un segnale con frequenza 0 che può essere opportunamente filtrato da un passa-alto.

SIMULAZIONE DEL TAPE DELAY E DEI DELAY ANALOGICI

Il termine ***tape delay*** fa riferimento ai vecchi sistemi di eco in cui un suono veniva registrato su nastro mediante una testina di registrazione e successivamente riletto da una testina di lettura, con possibilità di *feedback*. Il *delay time* dipendeva dalla distanza fra una testina e l'altra e dalla velocità di scorrimento del nastro. Questo tipo di eco poteva anche avere *delay time* molto lunghi (utilizzando due registratori posti a grande distanza l'uno dall'altro) oppure diverse testine di lettura per ottenere eco multiple. Alcuni sistemi usavano, al posto del nastro, dischi magnetici simili agli attuali hard disk. Il suono ripetuto risultava "filtrato" ogni volta che passava dalla testina di lettura a quella di registrazione, a causa della perdita di ampiezza delle componenti acute tipica delle registrazioni analogiche. La simulazione digitale di un *tape delay* prevede quindi, oltre al *delay* stesso, l'utilizzo di filtri che alterino il contenuto spettrale del suono ogni volta che passano per il circuito del *feedback*. Oltre al passa-basso si può anche inserire nel circuito un filtro passa-alto per evitare l'accumulo di frequenze basse: con questi accorgimenti è possibile simulare anche un altro tipo di *delay* analogico, di una generazione successiva rispetto al *tape delay*, ovvero una linea di ritardo che non si avvale di supporti magnetici ma di componenti *solid-state* (utilizzano questa tecnologia i *delay* analogici a pedale usati dai chitarristi).

ESEMPIO INTERATTIVO 6A.3 • Tape delay - delay *analogico*

(...)

il capitolo prosegue con:

6.9 PITCH SHIFT, REVERSE E DELAY VARIABILE
Delay variabile senza trasposizione

6.10 ALGORITMO DI KARPLUS-STRONG

- Esempi sonori e interattivi
- Test a risposte brevi
- Test con ascolto e analisi
- Concetti di base - glossario - discografia

6P
LINEE DI RITARDO: ECO, LOOP, FLANGER, CHORUS, FILTRI COMB E ALLPASS, PHASER, PITCH SHIFTING, REVERSE, ALGORITMO DI KARPLUS-STRONG

PREREQUISITI PER IL CAPITOLO

- Contenuti del volume 1, del capitolo 5 (teoria e pratica), dell'interludio C e del capitolo 6T

OBIETTIVI

Abilità

- Saper distinguere e gestire i diversi tipi di linee di ritardo
- Saper costruire mediante linee di ritardo effetti di eco, eco con feedback, simulazione di tape delay, slapback delay, ping-pong delay, multitap delay, multiband-multitap delay
- Saper costruire e gestire loop mediante linee di ritardo
- Saper costruire algoritmi di flanger e chorus
- Saper programmare e utilizzare algoritmi di comb filter e allpass filter di diverso tipo, anche per la costruzione di risuonatori armonici e di effetti phaser
- Saper costruire delay per gestire il pitch shifting, il reverse ed effetti di ritardo variabile, inclusi i glissandi e i cambi di delay time senza trasposizione di pitch
- Saper programmare algoriitmi di Karplus-Strong per la simulazione del suono di corda pizzicata ed altri suoni
- Saper utilizzare linee di ritardo per i messaggi Max

Competenze

- Saper realizzare una breve composizione basata sull'uso di suoni campionati, con utilizzo di loop, reverse, diversi tipi di linee di ritardo con vari tipi di elaborazione

ATTIVITÀ

- Costruzione e modifiche di algoritmi

SUSSIDI DIDATTICI

- Lista oggetti Max - Lista attributi, messaggi e argomenti per oggetti Max specifici

6.1 IL *DELAY TIME*: DAI FILTRI ALL'ECO

Per realizzare una linea di ritardo con MaxMSP possiamo usare, come abbiamo già visto nel paragrafo 3.6P del primo volume, l'oggetto **`delay~`** che ritarda il segnale di un certo numero di campioni: ricostruite la *patch* di fig. 6.1.

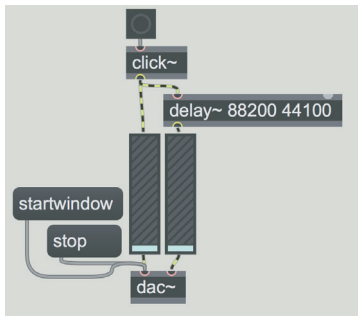


fig. 6.1: l'oggetto **`delay~`**

Il segnale generato dall'oggetto **`click~`** viene ritardato, sul canale destro, di 44100 campioni[1]: nel caso in cui la frequenza di campionamento della scheda audio sia 44100 Hz, il ritardo sarebbe quindi di 1 secondo.
È facile capire che specificare il ritardo in campioni è piuttosto scomodo se abbiamo bisogno di un tempo di *delay* espresso in secondi: il numero di campioni necessari infatti varia con il variare della frequenza di campionamento e non possiamo essere sicuri che il tempo di *delay* sarà lo stesso in ogni situazione. Generalmente l'oggetto **`delay~`** viene usato nei casi in cui abbiamo bisogno di ritardare il segnale di un numero esatto di campioni (come abbiamo visto per i filtri nel paragrafo 3.6P); tuttavia è possibile indicare il tempo di ritardo anche in altri formati, utilizzando la sintassi delle diverse unità di tempo che abbiamo visto al paragrafo IC.1. Modificate la *patch* precedente nel modo illustrato in figura 6.2.

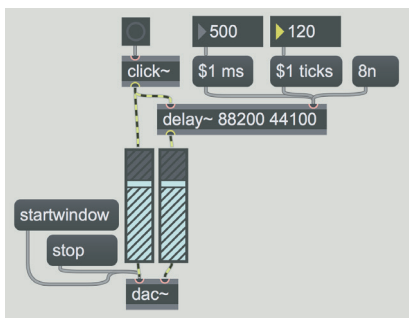


fig. 6.2: specificare il tempo di ritardo in diversi formati

[1] Ricordiamo che i due argomenti dell'oggetto rappresentano rispettivamente la quantità di memoria riservata al ritardo (ovvero il massimo ritardo che è possibile ottenere) e il ritardo effettivo: entrambe le quantità sono espresse, di *default*, in campioni.

In figura vediamo il modo di indicare il tempo di ritardo dell'oggetto **delay~** in tre diversi formati: millisecondi, *ticks* e *note value*. Il valore desiderato deve essere passato al secondo ingresso dell'oggetto e sostituisce l'argomento 44100. Fate riferimento al paragrafo IC.1 (sottoparagrafo "Valori di tempo") per l'elenco completo dei formati disponibili[2].

Nel seguito di questo capitolo ci capiterà spesso di utilizzare valori di ritardo che variano in modo continuo nel tempo. In questo caso, per evitare discontinuità, il tempo di ritardo va passato come segnale, e non come messaggio Max.
È possibile variare il tempo di ritardo dell'oggetto **delay~** tramite un segnale, ma solo utilizzando il tempo in campioni, e questo ci riporta alla situazione "scomoda" che abbiamo evidenziato sopra.
Se abbiamo bisogno di un ritardo variabile in modo continuo è preferibile usare una coppia di oggetti, **tapin~** e **tapout~**, che servono rispettivamente a scrivere e leggere (con un ritardo definito) il segnale in una determinata zona di memoria. Il tempo di ritardo viene espresso in millisecondi, e può essere modificato tramite un segnale. Inoltre con la coppia **tapin~** e **tapout~** è possibile inserire un *feedback* (retroazione) nel circuito del *delay*: è possibile cioè rimandare all'ingresso il suono ritardato, cosa che con l'oggetto **delay~** non si può fare. Un limite della coppia **tapin~ tapout~** è che non è possibile impostare un ritardo inferiore al *Signal Vector Size* (vedi par. 5.1P), mentre, come sappiamo, con l'oggetto **delay~** possiamo avere ritardi anche di un singolo campione. Gli oggetti **tapin~** e **tapout~** ci saranno molto utili nei prossimi paragrafi; cerchiamo quindi di capirne bene il funzionamento. Ricostruite la *patch* di fig. 6.3.

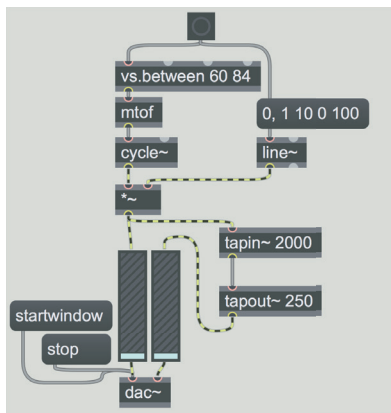


fig. 6.3: gli oggetti **tapin~** e **tapout~**

Nella parte alta abbiamo un semplice algoritmo che genera ad ogni *bang* un suono di breve durata con frequenza casuale. Questo suono viene eseguito immediatamente sul canale sinistro e ritardato, nel modo che ora vedremo, dalla coppia **tapin~ tapout~** sul canale destro.

[2] Un'altra possibilità è convertire in campioni il valore espresso in millisecondi tramite l'oggetto **mstosamps~** che abbiamo visto al paragrafo 2.4P del primo volume.

L'oggetto `tapin~` ha un argomento che indica in millisecondi (non in campioni) il massimo ritardo che sarà possibile realizzare: questo oggetto, infatti, si "appropria" di una porzione di memoria la cui lunghezza in campioni è data dall'argomento che viene diviso per 1000 (per ottenere la durata equivalente in secondi) e moltiplicato per la frequenza di campionamento. Nel nostro caso ad esempio l'argomento è 2000 (millisecondi) che diviso per 1000 ci dà 2 (secondi); se la frequenza di campionamento fosse 44100, la porzione di memoria a disposizione dell'oggetto sarebbe di 88200 campioni. Questa memoria viene usata da `tapin~` per scrivere, ovvero per registrare, il segnale che riceve: ogni volta che arriva alla fine della memoria (ogni due secondi, nel nostro caso) ricomincia da capo, cancellando ciò che aveva registrato precedentemente. Tecnicamente si dice che sta utilizzando un *buffer circolare* (vedi anche il paragrafo 6.2 della teoria), che possiamo immaginare come un nastro magnetico chiuso ad anello che scorre sotto una testina di registrazione (rappresentata da `tapin~`). L'oggetto `tapout~` invece funge da testina di lettura, e legge lo stesso *buffer* circolare ad una certa distanza (data dall'argomento) rispetto alla testina di registrazione di `tapin~`, generando quindi il ritardo desiderato. Nel caso illustrato in figura 6.3 il *buffer* circolare è lungo 2 secondi (2000 millisecondi) come indicato dall'argomento di `tapin~`, e il tempo effettivo di *delay* è 250 millisecondi, come indicato dall'argomento di `tapout~`. Vedremo nel prossimo paragrafo come variare questi argomenti.
Resta da dire del cavo che collega `tapin~` a `tapout~`: come si può vedere non si tratta di un cavo audio (dal momento che è un cavo nero), ma di un collegamento che fa sì che `tapin~` e `tapout~` condividano lo stesso *buffer* circolare. Se colleghiamo un oggetto `print` all'uscita di `tapin~` e facciamo clic sul *message box* "startwindow", possiamo vedere nella *Max Console* il messaggio "tapconnect": questo messaggio viene inviato dall'oggetto `tapin~` all'oggetto `tapout~` ogni volta che si avvia il motore DSP e serve appunto a far condividere ai due oggetti la stessa zona di memoria.
Per avere più punti di lettura in una stessa linea di ritardo è possibile collegare più oggetti `tapout~` ad uno stesso `tapin~`, oppure fornire a `tapout~` un numero di argomenti pari alla quantità di punti di lettura che desideriamo.

6.2 ECO

Per ottenere un effetto di eco con gli oggetti `tapin~` e `tapout~` ricostruite la *patch* di fig. 6.4.

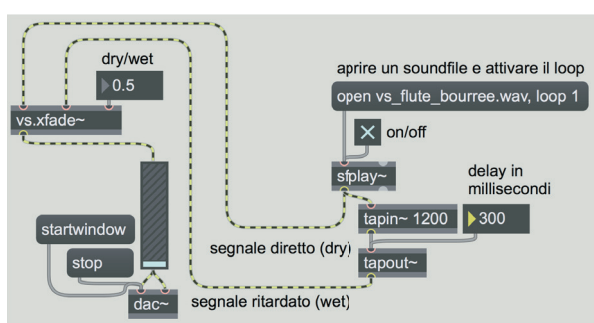


fig. 6.4: effetto eco

Notate che il *message box* a destra contiene due messaggi separati da una virgola: con il primo apriamo un file audio, con il secondo impostiamo la modalità *loop* per l'oggetto **sfplay~**. Come sappiamo il file audio vs_flute_bourree.wav è contenuto nella libreria *Virtual Sound*, e viene ritrovato da Max senza bisogno di specificare un percorso (naturalmente se avete installato correttamente la libreria!). Se volete caricare un altro suono vi ricordiamo che dovete inviare a **sfplay~** il messaggio "open", e si aprirà una finestra che vi permetterà di selezionare il file che volete. Se invece volete provare l'effetto utilizzando uno strumento o la voce, collegate un microfono alla vostra scheda audio e sostituite l'oggetto **sfplay~** con **adc~**: potete fare questa sostituzione su gran parte delle *patch* di cui parleremo nel seguito di questo capitolo, fate però attenzione al *feedback* tra casse e microfono! Soprattutto all'inizio, quando ancora dovete prendere confidenza con gli effetti, vi consigliamo di utilizzare una cuffia.
L'oggetto **vs.xfade~** sulla sinistra ci permette di miscelare il suono diretto (*dry*) proveniente da **sfplay~** con il suono ritardato (*wet*) proveniente da **tapout~**. Abbiamo già visto l'oggetto **vs.xfade~** nel paragrafo 3.5P del primo volume: ricordiamo che il numero inviato al terzo ingresso serve a modificare la proporzione tra i segnali che entrano nei primi due ingressi; con 0 viene riprodotto solo il primo segnale, con 1 solo il secondo, con 0.5 vengono miscelati i due segnali in parti eguali e così via.
Inviando un messaggio numerico a **tapout~** possiamo impostare il tempo di *delay.* Fate clic sul **toggle** collegato all'oggetto **sfplay~** e provate a variare il tempo di *delay* per sentire diversi possibili effetti di eco. Verifichiamo innanzitutto che per sentire la ripetizione del suono è necessario superare la *zona di Haas* di 25-35 millisecondi. Con il file audio specificato in figura (un suono di flauto il cui attacco è abbastanza dolce) si ha una chiara sensazione di raddoppio del suono quando si superano i 50 millisecondi. Dal momento che l'esecuzione della *bourrée* è a 100 bpm, provate anche un *delay* di 300 millisecondi (corrispondente al ritardo di una croma, ovvero un ottavo) e multipli. Con la coppia **tapin~ tapout~** è possibile introdurre il *feedback* per realizzare eco multiple (cosa che invece è impossibile con **delay~**), modificate la *patch* come illustrato in fig. 6.5.

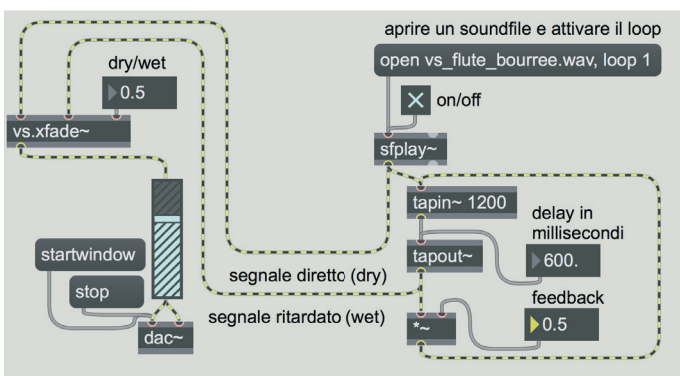


fig. 6.5: eco con *feedback*

Dall'uscita di `tapout~` viene prelevato il segnale, che viene riscalato da un moltiplicatore controllato da un *floating point number box* (sotto la scritta "feedback"). Il segnale riscalato viene rimandato all'ingresso di `tapin~` dove si somma con il nuovo segnale prodotto da `sfplay~`.
Attenzione! Per evitare distorsioni dovete immettere un limite nel *floating point number box* che regola il *feedback*; aprite l'*inspector* dell'oggetto, andate alla categoria "Value", e impostate gli attributi "Minimum" e "Maximum" rispettivamente a 0 e 0.99.
Questi attributi limitano l'intervallo di valori che è possibile produrre con il *number box*: in questo modo eviterete di impostare un *feedback* superiore o pari a 1 e sarete sicuri che le ripetizioni dell'eco si estingueranno gradualmente.
Il *feedback*, come abbiamo già detto, è realizzabile con la coppia `tapin~ tapout~` ma non con `delay~`: MSP infatti non permette che il circuito del segnale si chiuda in un anello o *loop* (detto anche *feedback loop*[3]). Osservando la figura 6.6 possiamo vedere la differenza tra un *feedback* realizzato con l'oggetto `delay~` (che blocca il motore DSP) e uno realizzato con `tapin~ tapout~`: nel primo caso il circuito audio si chiude ad anello (il circuito infatti contiene solo cavi giallo-neri), nel secondo caso il circuito audio resta aperto, perché il cavo di collegamento tra `tapin~` e `tapout~` non è un cavo audio[4].

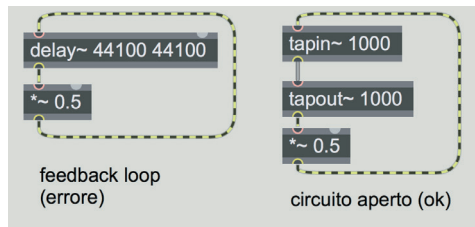


fig. 6.6: il *feedback loop*

ATTIVITÀ

Per evitare l'accumulo del *DC offset* nella *patch* di fig. 6.5, inserite un filtro passa-alto del primo ordine (vedi par. 3.4P del primo volume) tra l'oggetto `sfplay~` e l'algoritmo di *delay*. La frequenza di taglio deve essere al di sotto della banda audio (ad es. 10 Hz) per evitare di eliminare componenti significative del segnale in ingresso.

[3] Attenzione, il *feedback loop*, che costituisce un errore in MSP, non va confuso con il *loop* di un file audio di cui abbiamo parlato nel capitolo 5.
[4] Come abbiamo detto, mediante questo collegamento l'oggetto `tapin~` trasmette il messaggio "tapconnect" all'oggetto `tapout~`, e questo fa sì che i due oggetti condividano lo stesso buffer circolare.

SIMULAZIONE DEL TAPE DELAY

Per realizzare una simulazione del *tape delay* dobbiamo inserire un filtro passa-basso all'uscita di `tapout~` (fig. 6.7).

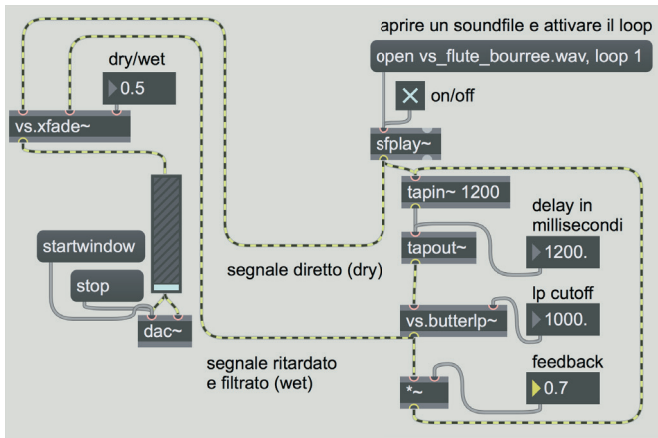


fig. 6.7: "tape delay" con filtro passa-basso e *feedback*

Qui abbiamo utilizzato un filtro di Butterworth (`vs.butterlp~`, vedi capitolo 3); provate a ricostruire la *patch* inserendo i parametri indicati in figura: sentirete che ogni copia ritardata del suono è più scura della precedente. Per sentire l'effetto da solo, fate clic sul `toggle` "on/off" per eseguire il file di suono e dopo circa un secondo di nuovo clic per arrestare il suono originale. Per evitare l'accumulo di frequenze basse, e per rendere più definite le ripetizioni, possiamo aggiungere il filtro passa-alto `vs.butterhp~` (fig. 6.8): tale filtro ci permette (come "effetto collaterale") anche di evitare l'accumularsi del *DC offset* eventualmente presente nel segnale in ingresso.

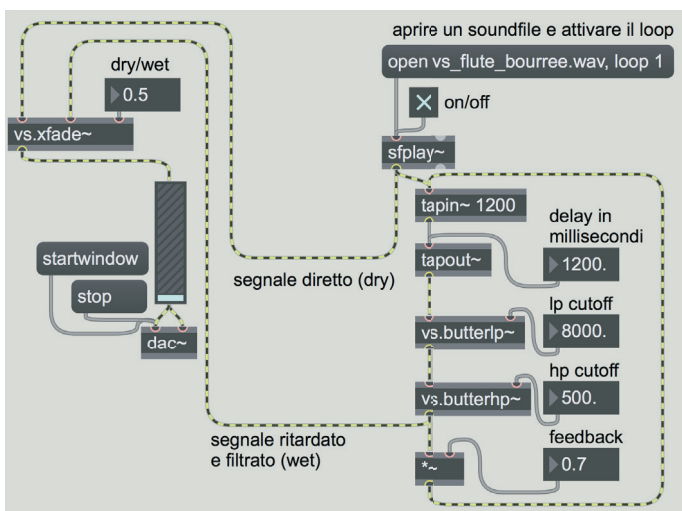


fig. 6.8: aggiunta di un filtro passa-alto al *tape delay*

Provate diverse frequenze di taglio: con una banda abbastanza stretta sulle frequenze medio-alte (*low-pass* 4000 Hz, *high-pass* 1000 Hz) è possibile ad esempio realizzare un suono "telefonico".

ATTIVITÀ

Partendo dalla *patch* di fig. 6.7 utilizzate filtri diversi, ad esempio il filtro passa-banda `vs.butterbp~`, oppure un banco di filtri tramite `fffb~` (vedi par. 3.7P) per trasformare il contenuto spettrale del segnale ritardato: in entrambi i casi fate molta attenzione ai valori di Q!, non devono assolutamente andare sotto lo 0, né essere troppo alti; partite sempre da un *feedback* = 0, e poi alzatelo cautamente.

(...)

il capitolo prosegue con:

7T
PROCESSORI DI DINAMICA

PREREQUISITI PER IL CAPITOLO

- Contenuti del volume 1 e dei capitoli 5 e 6 (teoria)

OBIETTIVI

Conoscenze

- Conoscere i diversi usi dei processori di dinamica
- Conoscere le possibilità di utilizzo degli envelope follower
- Conoscere l'utilizzo e i parametri dei compressori, dei de-esser e dei limiter
- Conoscere l'utilizzo e i parametri degli espansori e dei gate
- Conoscere gli utilizzi e le differenze fra:
  downward compression; downward expansion;
  upward compression, upward expansion e parallel compression;
  compressori multi-zona; compressori multi-banda; compressori a soglia temporale;
  external side-chain e ducking; gate e ducker adattivi, triggering gate, gate sequencer; feedback controllato in dinamica.

CONTENUTI

- Envelope follower
- Compressori e limiter
- Espansori e gate
- downward upward e parallel compression
- Downward e upward expansion
- side-chain e ducking
- Utilizzi tecnici e creativi dei processori di dinamica

ATTIVITÀ

- Esempi sonori

VERIFICHE

- Test a risposte brevi
- Test con ascolto e analisi

SUSSIDI DIDATTICI

- Concetti di base - Glossario - Discografia

Abbiamo parlato, nel par. 5.1, di **gamma dinamica** (o ***dynamic range***) come rapporto fra l'ampiezza massima e l'ampiezza minima che è possibile rappresentare all'interno di un determinato hardware, software o supporto. Questo è un fatto puramente tecnico, dovuto al numero di bit. All'interno del *range* consentito dal numero di bit possiamo operare scelte tecnico-espressive molto diverse, diverse per ogni pezzo, per ogni suono. Possiamo lavorare con un sistema che tecnicamente ci consente una gamma dinamica massima di 90 dB, ma scegliere di comporre un lavoro per la televisione in cui la gamma dinamica può essere molto ridotta, diciamo ad esempio 15 dB. Il contesto in cui si inserisce il discorso sui processori di dinamica non influirà quindi sul numero di bit, che viene deciso all'inizio del lavoro e non dovrebbe essere più mutato, ma sulle scelte tecnico-espressive che si possono operare all'interno di un sistema dato per mutare il *range* dinamico di un suono o di un pezzo. I **processori di dinamica** sono dispositivi che utilizzano tecniche di trasformazione del *range* dinamico di un suono (o una serie di suoni, o un pezzo etc.), per scopi molto diversi, sia tecnici, sia creativi. Ricordiamo che la gamma dinamica esprime *un rapporto*, o *una differenza* fra ampiezza massima e ampiezza minima, e che è concetto ben diverso dall'ampiezza assoluta. Si può avere un pezzo in cui la gamma dinamica è di 20 dB e l'ampiezza massima sia a 0 dB e un altro in cui la gamma dinamica sia identica ma l'ampiezza massima sia a –3 dB. Vedremo come lavorare su entrambi i parametri, ma la specificità dei processori di dinamica è quella di trasformare il *range* dinamico in modi diversissimi.
Ci occuperemo in particolare, in questo capitolo, degli *envelope follower/envelope shaper* (estrattori/modellatori di inviluppo), dei diversi tipi e diversi usi di compressori ed espansori, del *limiter* e del *gate*.

7.1 ENVELOPE FOLLOWER

L'***envelope follower*** (o *peak amplitude follower*, o *envelope detector*) svolge la funzione di estrazione dell'inviluppo, misurando l'ampiezza dei picchi positivi di una forma d'onda. Sulla base di questa serie di valori d'ampiezza estratti da un suono A, l'*envelope follower* produce un segnale di controllo che può, ad esempio, essere "imposto" ad un suono B come controllo dell'inviluppo d'ampiezza (semplicemente facendo un'operazione di moltiplicazione fra il suono e l'inviluppo estratto), oppure come controllo della frequenza centrale di un filtro o di altri parametri. Le applicazioni possono essere molteplici, si possono ad esempio applicare inviluppi percussivi a suoni continui, oppure applicare l'inviluppo delle onde del mare a un suono di coro, come negli esempi che seguono.
Si può persino privare un suono del suo inviluppo, dividendo il segnale del suono per l'inviluppo stesso. Se dividiamo un suono per il suo inviluppo, infatti, è come se annullassimo l'inviluppo implicito nel suono stesso, trasformandolo nella costante 1.[1] In questo modo il suono risulta privo di escursione dinamica. Una volta appiattita la dinamica di un suono, è possibile applicare un inviluppo

[1] Qualsiasi numero diviso per se stesso dà infatti come risultato 1.

tratto da un suono diverso. Il suono con un inviluppo piatto potrà cioè essere poi moltiplicato per l'inviluppo di un altro suono.
Si può anche ottenere l'inverso di un inviluppo, in modo che quando il suono originale è al suo picco massimo, l'inviluppo inverso è al suo minimo. In alcuni casi ci si riferisce a questo tipo di operazioni anche con il termine *envelope shaping*.
Un'altra funzione è quella denominata *balance* in alcuni linguaggi di programmazione per l'audio[2] (da non confondere con lo stesso termine usato per indicare il controllo della spazializzazione stereo). Questa tecnica si utilizza, ad esempio, nel caso in cui un suono filtrato risulti troppo debole (o troppo forte) rispetto allo stesso suono prima del filtraggio. Un tipico esempio si ha quando la frequenza centrale di un filtro passa-banda non è presente, o ha un'ampiezza molto bassa, nel suono da filtrare. In questo caso si può, tramite l'algoritmo illustrato, applicare al suono filtrato l'ampiezza del suono prima del filtraggio.

ESEMPIO SONORO 7A.1

a) Inviluppo del pianoforte a suono di flauto
b) Inviluppo di un rullante a un suono di tromba
c) Esempio da A.Cipriani *Aqua Sapientiae/Angelus Domini*: inviluppo di suoni di onde del mare applicato a voci in contrappunto

La misurazione dell'inviluppo può essere realizzata con sistemi diversi: uno di questi, descritto da Dodge e Jerse, è quello di utilizzare una tecnica chiamata *rectification*, cioè quella di trasformare i valori di ampiezza dei campioni di un suono in valori assoluti (cioè indipendenti dal segno + o -). I valori negativi dei campioni diventeranno così tutti positivi. Il segnale "rettificato" verrà quindi passato attraverso un filtro passa-basso (solitamente a frequenza sub-audio) che serve ad arrotondare il segnale "spigoloso". Se il filtraggio è troppo pesante, la curva risulterà troppo lontana dall'originale, se invece il filtraggio è troppo leggero, si possono avvertire le asperità e discontinuità dell'inviluppo. E' importante perciò saper decidere che tipo di filtro passa-basso applicare a seconda della complessità dell'inviluppo del suono A, e anche dell'uso che se ne vuole fare come segnale di controllo. E' bene comunque avere una definizione alta per realizzare estrazione di inviluppi complessi.
Un altro sistema è quello di calcolare una media dei valori assoluti d'ampiezza dei campioni. In questo caso il grado di definizione dell'inviluppo sarà dato anche dal numero di campioni che vengono considerati per calcolare la media: maggiore è il numero di campioni, meno accurato sarà il profilo dell'inviluppo.
Il segnale di controllo generato dall'*envelope follower* può essere applicato, come detto, anche ai filtri. Si può decidere per esempio che tale segnale,

[2] Cfr. Dodge and Jerse 1997, p. 181

opportunamente modificato, controlli la frequenza centrale di un filtro passa-banda o la frequenza di taglio di un filtro passa-basso. Questi filtri possono agire su un secondo suono, o per elaborare il suono di partenza. (Attenzione! E' importante in questo caso non confondere la funzione di arrotondamento del primo filtro passa-basso sul segnale di controllo in uscita dall'*envelope follower* rispetto al secondo filtro passa-basso che invece è controllato da tale segnale e che agisce su un suono). Con questo sistema si può anche simulare un risultato simile a quello del VCF di un sintetizzatore (descritto nel par.3.5 della teoria), cioè quello in cui la frequenza del filtro dipende dall'inviluppo d'ampiezza, ovvero può seguirne il profilo, restituendo un suono più brillante quando l'ampiezza è massima e più scuro quando l'ampiezza diminuisce, comportandosi quindi come buona parte degli strumenti acustici.

ESEMPIO SONORO 7A.2

a) *Envelope follower* come controllo per il filtraggio di un secondo suono (passa banda)
b) *Envelope follower* come controllo per il filtraggio di un secondo suono (passa-basso)
c) *Envelope follower* come controllo per il filtraggio dello stesso suono da cui è estratto l'inviluppo (passa-banda)
d) *Envelope follower* come controllo per il filtraggio dello stesso suono da cui è estratto l'inviluppo (passa-basso).

7.2 COMPRESSORI E DOWNWARD COMPRESSION

IL COMPRESSORE

Il **compressore**[3] è un processore di dinamica che serve a ridurre la gamma dinamica di un suono. Gli usi tecnici e creativi del compressore sono molteplici, si tratta di un dispositivo molto importante nella catena elettroacustica. Proviamo, prima di parlare degli usi possibili di questo dispositivo, a descrivere qualcosa di simile a ciò che succede in un compressore semplice. Immaginiamo di avere un amplificatore con un potenziometro del volume. Abbiamo un suono in ingresso che varia in modi imprevedibili. Vogliamo che tale suono,

[3] Attenzione! La parola compressione può creare ambiguità, dal momento che viene usata sia per la riduzione dei dati (come nel caso di cui abbiamo trattato nel par. 5.3) sia per la compressione intesa come riduzione del *range* dinamico di un suono (di cui tratteremo in questo paragrafo). Per questo nel cap. 5 ci siamo riferiti alla riduzione dei dati chiamandola compressione dei dati, lasciando, in questo capitolo, il termine semplice "compressione" per indicare una riduzione del *range* dinamico.

mantenga in uscita un livello di pressione sonora alto, ma non superi una certa intensità. Cosa facciamo? Non appena ci accorgiamo che il suono supera una certa soglia, provvediamo immediatamente a diminuire il guadagno (cioè "abbassiamo il volume"), e quando il suono torna sotto la soglia riportiamo il guadagno alla posizione iniziale.

- Qual'è la soglia oltre la quale "abbassiamo il volume"?
- Di quanto lo abbassiamo?
- A che velocità ruotiamo il potenziometro per abbassare il volume?
- Con quale velocità ruotiamo il potenziometro per rialzare il volume?

Rispondendo a queste domande ci avviciniamo ad alcuni concetti fondamentali che riguardano i parametri di un compressore:
la soglia (threshold) ovvero: oltre quale soglia in dB entra in funzione l'azione di compressione?
il rapporto di compressione (***ratio*** oppure ***slope***) ovvero: come viene riscalata l'ampiezza del segnale quando supera la soglia?
il tempo d'attacco (*attack*) ovvero: a partire dal momento in cui il segnale in ingresso supera la soglia, in quanto tempo, ad ogni aumento di ampiezza, il compressore raggiunge il *ratio* stabilito mediante la diminuzione del guadagno?
il tempo di rilascio (*release*) ovvero: in quanto tempo il compressore, ad ogni diminuzione di ampiezza raggiunge il *ratio* stabilito mediante l'aumento del guadagno?
È importante notare che molta letteratura su questo tema insiste sulla falsa nozione che il *release* abbia luogo solo quando il suono rientra al di sotto della soglia o che l'attacco abbia luogo solo nel momento in cui si supera la soglia. In realtà il *release* avviene ad ogni diminuzione dell'ampiezza anche quando il suono rimane sopra la soglia. Solo l'ultimo *release* ha luogo in corrispondenza del rientro dell'ampiezza del suono al di sotto della soglia.[4] Allo stesso modo, finché il suono rimane sopra la soglia l'attacco ha luogo anche ogni volta in cui ci sia un aumento di ampiezza.

Alcuni degli usi di un compressore sono:

- **rendere più comprensibile la voce di uno speaker** in un documentario: immaginiamo ad esempio un passaggio di un documentario in cui ci sia musica e suoni d'ambiente, e immaginiamo di voler sovrapporre ad essi la voce di uno speaker senza diminuire troppo il livello degli altri due segnali, ma lasciando la piena comprensibilità della voce. Può succedere che alcuni fonemi dello speaker vengano mascherati dalla musica e altri siano invece comprensibili, cioè abbiano l'intensità giusta: un compressore può servire proprio per attenuare le parti più forti senza che le parti più deboli del segnale vengano attenuate per poter poi, una volta livellato il segnale, aumentarne l'intensità globale e rendere la voce sempre comprensibile.
- **ottenere effetti musicali**, ben noti nel *rock*, come per esempio la compressione molto accentuata di un suono di chitarra elettrica, che al limite fa scomparire l'attacco della nota, togliendole la caratteristica di corda

[4] A questo proposito si veda Izhaki, R., 2012, pp. 280-1

pizzicata oppure la forte compressione di una voce, che comprimendo i picchi vocali lascia trasparire meglio tutti i suoni meno evidenti di una voce come quello della saliva e del respiro.

- comprimere la gamma dinamica e successivamente poter **aumentare il livello complessivo del segnale**. Se ad esempio un dato pezzo musicale ha alcuni picchi a 0 dB, ma la gran parte dei suoni nel pezzo ha una intensità molto più bassa, ci troviamo nell'impossibilità di aumentare l'ampiezza di tutto segnale perché altrimenti i picchi, che già sono a 0 dB, verrebbero distorti. A questo punto applichiamo una compressione tale da ridurre a –3 dB tutti i picchi, lasciando inalterati tutti i suoni a intensità bassa. Una volta applicata la compressione, potremo a quel punto aumentare di 3 dB il livello complessivo del segnale compresso, riportando perciò i picchi a 0 dB ma avendo aumentato il livello di tutti gli altri suoni presenti nel segnale. Se questa funzione viene applicata su un file audio intero (ad esempio su un intero pezzo musicale, cioè in fase di mastering) anziché su un suono o una serie di suoni, si preferisce fare uso di un *limiter* (di cui parleremo più avanti in questo capitolo).
- **Livellare la pressione sonora diseguale di alcuni strumenti**, ad esempio i fiati, i quali tendono a produrre un'intensità maggiore sulle frequenze acute rispetto a quelle gravi (in questo caso, come vedremo, si può utilizzare un compressore multibanda).
- **Ridurre le sibilanze di una voce**. Ciò è possibile tramite una combinazione speciale di un filtro passa-banda con un compressore. Esiste, a questo scopo, uno speciale compressore, che ha il nome di *de-esser* (utile, ad esempio, quando uno speaker o un cantante ha *s* troppo sibilanti), di cui parleremo alla fine di questo paragrafo.

ESEMPIO SONORO 7B.1 • *Esempi di compressione*
(esempi prima e dopo la compressione)

a) Voce di uno speaker resa più comprensibile tramite compressione
b) Compressione come mezzo per il mutamento del timbro (es. chitarra elettrica)
c) Uso della compressione per aumentare il livello complessivo del segnale
d) Uso della compressione per livellare la pressione sonora diseguale di alcuni strumenti
e) Uso del *de-esser* per la riduzione delle sibilanze di una voce

PARAMETRI DEL COMPRESSORE E *DOWNWARD COMPRESSION*

Si possono distinguere una compressione verso il basso, o ***downward compression*** (che attenua i picchi sopra la soglia, è la compressione di cui abbiamo parlato finora) e una compressione verso l'alto, o ***upward compression*** (che aumenta il livello delle zone con bassa intensità, ne parleremo più avanti).

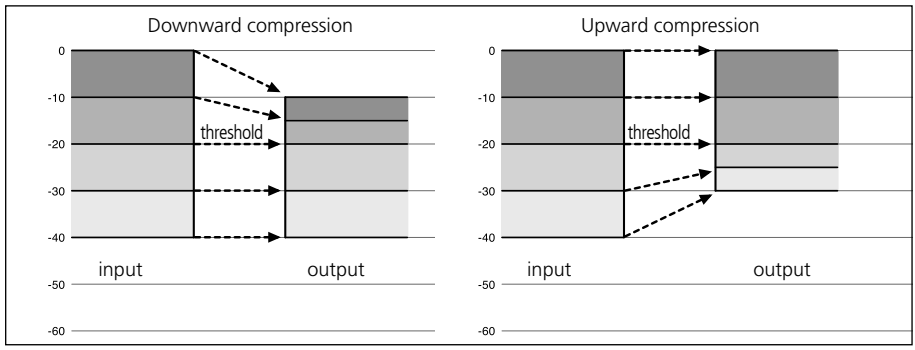


fig. 7.1: *downward* e *upward compression*[5]

(...)

[5] Immagine adattata da Izhaki, R. - 2012 - pag. 263

il capitolo prosegue con:

Triggering gate
Gate-sequencer (live slicing)
Feedback controllato in dinamica

- Concetti di base - Glossario - Discografia

7P
PROCESSORI DI DINAMICA

PREREQUISITI PER IL CAPITOLO

- Contenuti del volume 1, dei capitoli 5 e 6 (teoria e pratica), dell'interludio C e del capitolo 7T

OBIETTIVI

Abilità

- Saper applicare l'inviluppo di un suono a un altro suono o ad altri parametri.
- Saper costruire algoritmi per gli envelope follower, downward e upward compression, parallel compression, de-esser, limiter e live normalizer.
- Saper applicare i valori dei parametri adatti per qualsiasi processore di dinamica.
- Saper costruire algoritmi per il gate, downward o upward expander, per il side-chain e il ducking.
- Saper usare i processori di dinamica sia per scopi tecnici, sia per fini creativi.

CONTENUTI

- Envelope follower
- Compressori, limiter, live normalizer e de-esser
- Espansori e gate
- Realizzazioni di downward e upward compression ed expansion
- Realizzazioni di parallel compression
- Realizzazioni di side-chain esterni e ducking
- Realizzazioni di gate e ducker adattivi, triggering gate, gate-sequencer e feedback controllati in dinamica

ATTIVITÀ

- Costruzione e modifiche di algoritmi

SUSSIDI DIDATTICI

- Lista oggetti Max - Lista comandi, attributi e parametri per oggetti Max specifici

7.1 ENVELOPE FOLLOWER

Ci sono diversi modi per realizzare un *envelope follower* in Max: come vedremo ogni metodo ha caratteristiche proprie che possono renderlo più o meno utile nelle diverse applicazioni. Analizzeremo i diversi metodi e ne discuteremo gli usi più efficaci.
Il primo oggetto che presentiamo è **`average~`**: ricostruite la semplice *patch* di figura 7.1.

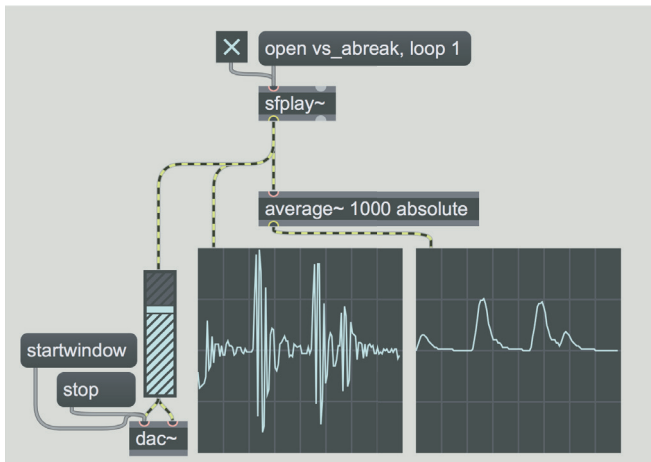


fig. 7.1: *envelope following* con `average~`

Questo oggetto genera un segnale che rappresenta una media (in inglese *average*) dei valori dei campioni in entrata. Come vediamo in figura, il segnale in uscita segue il profilo dinamico, ovvero l'inviluppo, del segnale in ingresso. Il primo argomento di `average~` indica il numero di campioni in ingresso da considerare per il calcolo della media (in figura 1000), il secondo argomento (in figura è "absolute") specifica il modo in cui viene calcolata la media.
I modi possono essere tre:

1) *bipolar* (modalità di *default*), viene semplicemente calcolata la media dei campioni in ingresso; ovvero si somma il valore dei campioni e si divide il risultato per il numero dei campioni. Nel caso di segnali audio tale media, quando è calcolata su un grande numero di valori (pari almeno ad un ciclo della forma d'onda in ingresso), risulta molto vicina allo zero, in quanto i valori positivi e quelli negativi di un segnale bipolare tendono ad equivalersi, annullandosi a vicenda.
2) *absolute*, la media viene calcolata sul valore assoluto (cioè sempre positivo) dei campioni in entrata. È la modalità usata in figura.
3) *rms* (*root mean square*). In questa modalità i campioni in ingresso vengono elevati al quadrato (e diventano perciò tutti positivi), viene poi calcolata la media, e successivamente la radice quadrata di tale media. Questo processo, che è il più dispendioso per la CPU, è quello che dà i risultati più accurati. La modalità *absolute* è comunque sufficiente in molti casi.

È possibile modificare il modo di calcolo inviando all'ingresso di **average~** i messaggi "bipolar", "absolute" o "rms". È anche possibile modificare il numero di campioni da utilizzare per il calcolo della media inviando all'oggetto un valore intero. Tale valore però non può essere superiore all'argomento specificato.
Quanti campioni devono essere utilizzati per ottenere un buon *envelope following*? Non è possibile indicare un valore buono per tutte le situazioni; notate comunque che all'aumentare del numero di campioni il segnale risultante tenderà ad appiattire gli accenti del suono in ingresso, mentre diminuendo il numero dei campioni l'inviluppo seguirà sempre più fedelmente l'ampiezza di picco, fino a che, utilizzando un solo campione per la "media", verrà riprodotto l'andamento istantaneo della forma d'onda in entrata.

Possiamo utilizzare l'inviluppo ricavato da **average~** applicandolo ad un segnale ad ampiezza costante: modificate la *patch* nel modo indicato in figura 7.2.

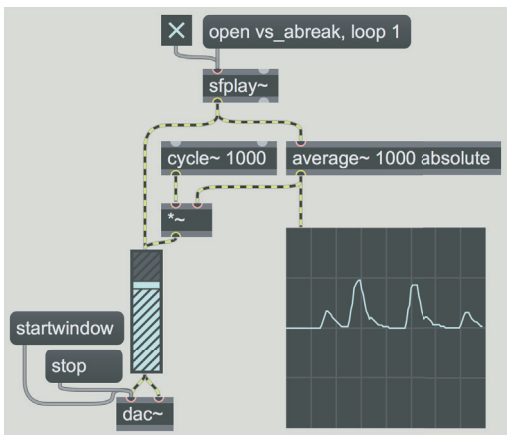


fig. 7.2: inviluppo applicato ad un oscillatore

In questo caso, a un oscillatore sinusoidale a 1000 Hz viene applicato l'inviluppo del *loop* di batteria. Se avviate la *patch* noterete che l'inviluppo applicato all'oscillatore appare ritardato rispetto al suono diretto della batteria: questo avviene perché ogni campione generato dall'oggetto **average~** è il risultato della media dei 1000 campioni precedenti. Una soluzione semplice consiste nel ritardare il segnale diretto (quello che va dall'oggetto **sfplay~** all'oggetto **gain~**) finché i due suoni non sembrino sincronizzati: provate ad esempio ad inserire un *delay* di 200-300 campioni.
Modificate poi il numero di campioni utilizzati da **average~** per calcolare la media collegando un *number box* all'oggetto. Cosa succede con 100 campioni? E con 10? E con 1?

Possiamo anche utilizzare l'inviluppo ricavato dall'envelope follower per controllare alcuni parametri di elaborazione del suono. Modificate la *patch* come indicato in fig. 7.3.

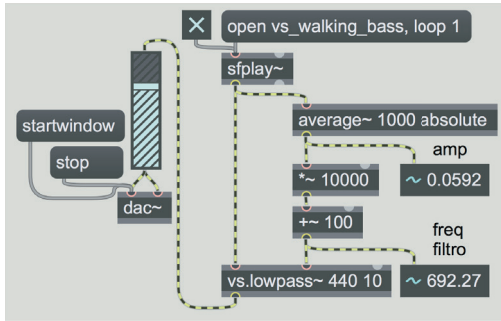


fig. 7.3: inviluppo che controlla un filtro

Il suono caricato è vs_walking_bass.wav, e l'inviluppo calcolato dall'oggetto `average~` viene utilizzato per controllare un filtro passa-basso applicato al soundfile. Come potete notare, la frequenza del filtro segue il profilo dinamico del suono. L'ampiezza dell'inviluppo viene moltiplicata per 10000, viene poi sommato il valore 100 e il risultato viene usato come frequenza di taglio. In realtà, dato che in questo caso il segnale generato dall'*envelope follower* raramente supera il valore 0.1, la frequenza di taglio massima sarà all'incirca di 1100 Hz (10000 • 0.1 + 100).
Provate a cambiare i valori del moltiplicatore, del sommatore, del fattore Q (secondo argomento del filtro) e il tipo di filtro (ad esempio usate un filtro passa-alto).

Vediamo come possiamo controllare parametri diversi tramite l'*envelope follower*: aprite la *patch* **07_01_envfollow.maxpat** (fig. 7.4).

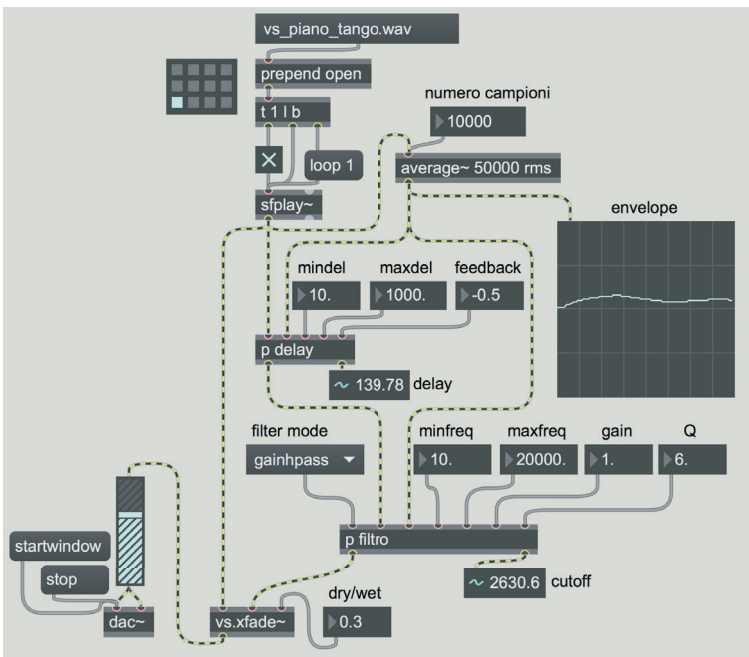


fig. 7.4: file **07_01_envfollow.maxpat**

In questa *patch* l'*envelope follower* serve a controllare contemporaneamente un tempo di *delay* e la frequenza di taglio di un filtro. Il segnale prodotto da **sfplay~** viene infatti inviato ad un oggetto **average~** che genera l'inviluppo (in modalità *rms*) e lo trasmette a due *subpatch*. La prima *subpatch*, [p delay], riceve l'inviluppo al secondo ingresso e lo usa per controllare il tempo di ritardo del suono che arriva all'ingresso di sinistra; il tempo di ritardo può variare tra un minimo e un massimo liberamente impostabili (vedi i due *flonum* "mindel" e "maxdel"), è inoltre possibile impostare un *feedback*. Il segnale ritardato viene generato all'uscita di sinistra della *subpatch*, mentre all'uscita di destra viene trasmesso il tempo effettivo di *delay* (visualizzato da un oggetto **number~**). Il contenuto della *subpatch* è abbastanza semplice (fig. 7.5).

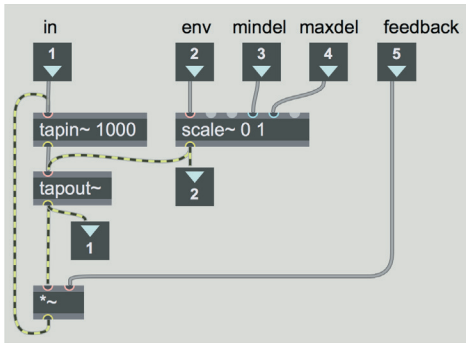


fig. 7.5: *subpatch* [p delay]

Il segnale generato da **average~** (inlet "env") viene passato ad un oggetto **scale~** che trasforma l'intervallo 0-1 dell'inviluppo nell'intervallo mindel-maxdel impostato nella *patch* principale[1]. Il tempo di ritardo viene trasmesso al circuito **tapin~/tapout~** con *feedback* visibile sulla sinistra.
Il segnale ritardato passa poi alla *subpatch* [p filtro], che utilizza l'inviluppo per modulare la frequenza di taglio di un filtro. Anche in questo caso il parametro viene impostato entro un minimo e un massimo ("minfreq" e "maxfreq") ed è inoltre possibile regolare l'ampiezza del segnale filtrato ("gain") e il fattore Q.
Si può cambiare il tipo di filtro utilizzando l'oggetto **umenu** "filter mode". I filtri disponibili sono tre: *gainlpass*, *gainhpass* e *gainbpass*; ovvero, filtro passa-basso, passa-alto e passa-banda, tutti e tre con regolazione del *gain*. Vediamo il contenuto della *subpatch* (fig. 7.6).

(...)

[1] C'è da notare che è molto improbabile che l'inviluppo calcolato da **average~** si avvicini al valore massimo 1 e che si ottenga quindi il massimo ritardo. Il parametro "maxdel" va perciò impostato tenendo conto di questo fatto.

il capitolo prosegue con:

Interludio D

GESTIONE AVANZATA DEI PRESET, BPATCHER E ARGOMENTI VARIABILI, GESTIONE DATI E PARTITURE

PREREQUISITI PER IL CAPITOLO

- Contenuti del volume 1, dei capitoli 5, 6 e 7 (teoria e pratica) e dell'interludio C

OBIETTIVI

Abilità

- Saper gestire sistemi di memorizzazione complessa
- Saper gestire la visualizzazione parziale o totale di abstraction e subpatch all'interno di una patch
- Saper gestire argomenti variabili e locali in abstraction o subpatch
- Saper gestire insiemi di dati e realizzare algoritmi per la gestione di "partiture"

CONTENUTI

- Sistemi avanzati di gestione e di interpolazione fra preset
- Sistemi di visualizzazione di abstraction e subpatch all'interno di una patch
- Gestione dei dati in Max

SUSSIDI DIDATTICI

- Lista oggetti Max - Lista attributi, argomenti, messaggi e comandi per oggetti Max specifici - Glossario

ID.1 GESTIONE AVANZATA DEI PRESET

Finora per gestire i parametri presenti in una *patch* abbiamo utilizzato l'oggetto `preset`, che permette una facile memorizzazione dei valori contenuti in oggetti interfaccia come il *number box* o il `multislider`.
In figura ID.1 ricapitoliamo rapidamente le caratteristiche dell'oggetto.

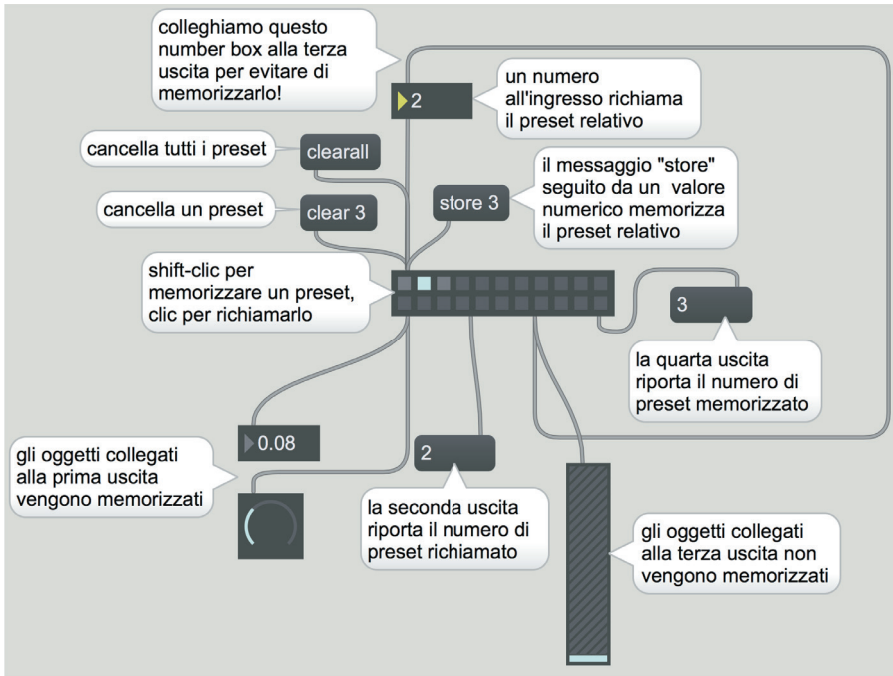


fig. ID.1: l'oggetto `preset`

È possibile includere o escludere determinati oggetti dalla memorizzazione nei *preset* tramite la prima e la terza uscita; cerchiamo di capire come funziona il meccanismo.
- Quando l'oggetto `preset` non ha nessun oggetto collegato alla prima e alla terza uscita, memorizza tutti i valori presenti negli oggetti interfaccia della *patch*.
- Quando alcuni oggetti sono collegati alla prima uscita, vengono memorizzati solo i valori contenuti in tali oggetti, tutti gli altri vengono ignorati.
- Quando alcuni oggetti sono collegati alla terza uscita (ma nessuno alla prima), i valori di tali oggetti vengono ignorati e tutti gli altri memorizzati (abbiamo utilizzato spesso la terza uscita per escludere dalla memorizzazione oggetti che regolano il volume dell'audio in uscita, come `gain~` o `live.gain~`, poiché tale regolazione deve dipendere dal sistema di ascolto che si ha a disposizione).
- Se ci sono oggetti collegati sia alla prima sia alla terza uscita, tali oggetti vengono rispettivamente memorizzati e ignorati, come nei casi precedenti, e eventuali oggetti non collegati vengono ignorati.

Come sappiamo per memorizzare una configurazione (o *preset*) bisogna fare shift-clic con il mouse su uno dei "pallini" di `preset`, mentre con un clic semplice richiamiamo la configurazione.
È inoltre possibile richiamare una configurazione inviando un numero intero all'ingresso di `preset`: il valore 1 corrisponde al primo "pallino" e così via.
Per memorizzare un configurazione si può anche mandare all'ingresso di `preset` il comando "store" seguito da un numero. Se vogliamo cancellare un *preset* possiamo inviare all'oggetto il comando "clear" seguito da un numero; per cancellarli tutti possiamo inviare il comando "clearall".
Quando un *preset* viene richiamato, il numero corrispondente viene riportato alla seconda uscita, quando un *preset* viene memorizzato, il numero corrispondente viene riportato alla quarta uscita.
Notate che nella *patch* in figura abbiamo escluso dalla memorizzazione il *number box* collegato all'ingresso di `preset`: sapreste dire perché?

Per quanto semplice e funzionale, il sistema di gestione dei parametri tramite `preset` è abbastanza limitato: non ci permette ad esempio di memorizzare lo stato di oggetti interfaccia contenuti in *subpatch*, né di interpolare i valori tra due configurazioni. Vedremo quindi, nel seguito di questo paragrafo, un sistema più raffinato (e naturalmente più complesso) di gestione dei *preset*.

L'OGGETTO PATTR

L'oggetto **`pattr`** è un contenitore universale di dati (può infatti memorizzare valori numerici, liste o simboli) in grado di condividere il proprio contenuto con gli oggetti interfaccia di Max. È inoltre l'oggetto che ci permetterà di muovere i primi passi nel sistema avanzato di gestione dei *preset*. Vediamone alcune caratteristiche nelle *patch* di figura ID.2.

Nonostante siano molto semplici, vi raccomandiamo di ricreare tutte e sei le *patch* contenute in figura, poiché l'oggetto `pattr` per alcune caratteristiche si discosta dagli oggetti standard di Max.
Nella *patch* numero 1 vediamo che un oggetto `pattr` può ricevere qualunque messaggio (numeri, stringhe o liste), e tali messaggi vengono immediatamente passati all'uscita di sinistra. Un *bang* trasmette di nuovo l'ultimo messaggio ricevuto.
È possibile "legare" (in inglese *to bind*) un oggetto interfaccia ad un `pattr` collegando la seconda uscita di quest'ultimo all'oggetto che si desidera legare (*patch* numero 2). Qualunque valore generato dall'oggetto legato viene anche memorizzato in `pattr` e passato alla sua uscita sinistra. Inoltre, come si vede dalla *patch* 3, un valore inviato a `pattr` viene trasmesso anche all'oggetto legato.
Il primo argomento di `pattr` diventa il suo *scripting name*. L'oggetto `pattr` deve avere obbligatoriamente uno *scripting name*, in assenza di un argomento lo *scripting name* viene assegnato da Max (*patch* numero 4).
Il secondo argomento di `pattr`, se corrisponde allo *scripting name* di un oggetto interfaccia, crea un legame senza bisogno di un collegamento: per ricostruire la *patch* numero 5 dovete, tramite l'*inspector*, assegnare lo *scripting name* "guitar" al *floating point number box*.

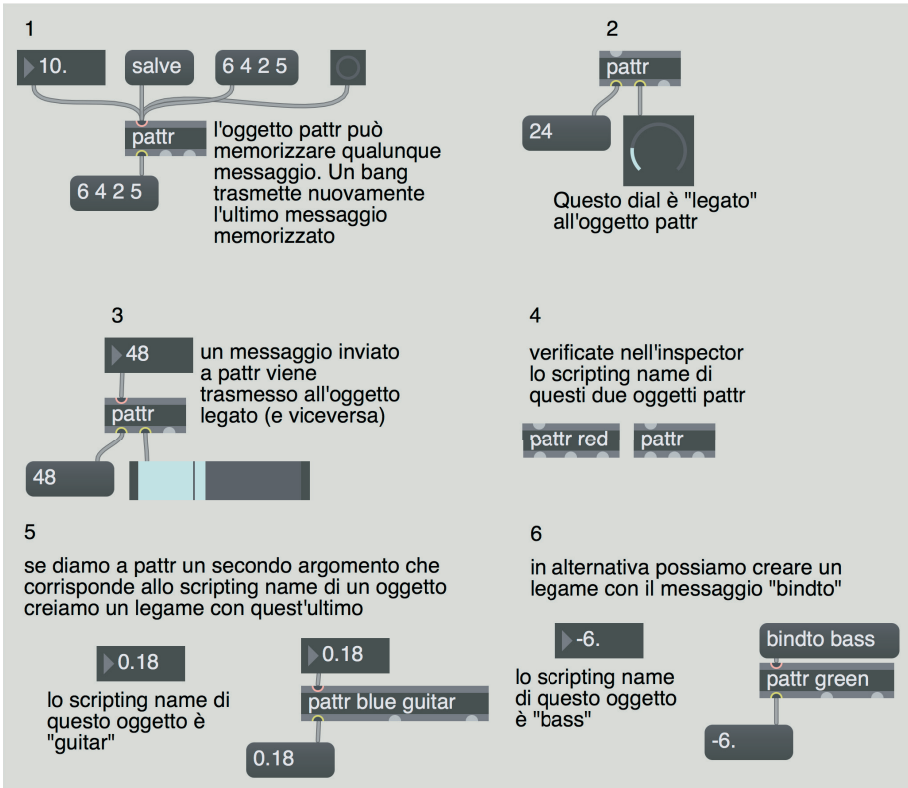


fig. ID.2: l'oggetto `pattr`

Affinché l'oggetto venga legato è necessario assegnare prima lo *scripting name* e poi creare l'oggetto [`pattr` blue guitar]. Notate che abbiamo in questo caso due *scripting name*: uno per l'oggetto `pattr` e uno per il `flonum`. Non sarebbe possibile assegnare lo stesso nome all'oggetto `pattr` e al `flonum` perché lo *scripting name* deve essere unico per ogni oggetto[1].
Possiamo legare "al volo" un oggetto interfaccia ad un `pattr` inviando a quest'ultimo il messaggio "bindto" seguita dallo *scripting name* dell'oggetto interfaccia (*patch* 6).
Quando si salva una *patch* che contiene un `pattr`, i dati eventualmente contenuti nell'oggetto vengono richiamati ogni volta che si carica la *patch*: questa caratteristica è gestita dall'attributo di `pattr` "autorestore" che di *default* è attivato. Per verificarlo provate a impostare a metà corsa lo `slider` della *patch* 3, poi salvate e ricaricate la *patch*: il cursore dello `slider` apparirà nella stessa posizione in cui lo abbiamo salvato, e non a 0 come avverrebbe per un oggetto non legato.

[1] Ci sono, come sappiamo, alcuni oggetti che condividono uno stesso nome, come ad esempio `buffer~` e `groove~`; ma in quel caso non si tratta dello *scripting name* degli oggetti, ma del nome dato allo spazio in memoria che contiene i dati condivisi.

L'OGGETTO PATTRSTORAGE

A parte la funzione *autorestore*, l'oggetto **pattr** non sembra introdurre particolari novità rispetto a quanto possiamo fare, ad esempio, con la coppia di oggetti **send/receive** o con l'oggetto **pvar**. Se però affianchiamo a **pattr** l'oggetto **pattrstorage** possiamo scoprire alcune novità interessanti. Aprite la *patch* **ID_01_pattrstorage.maxpat** (fig. ID.3).

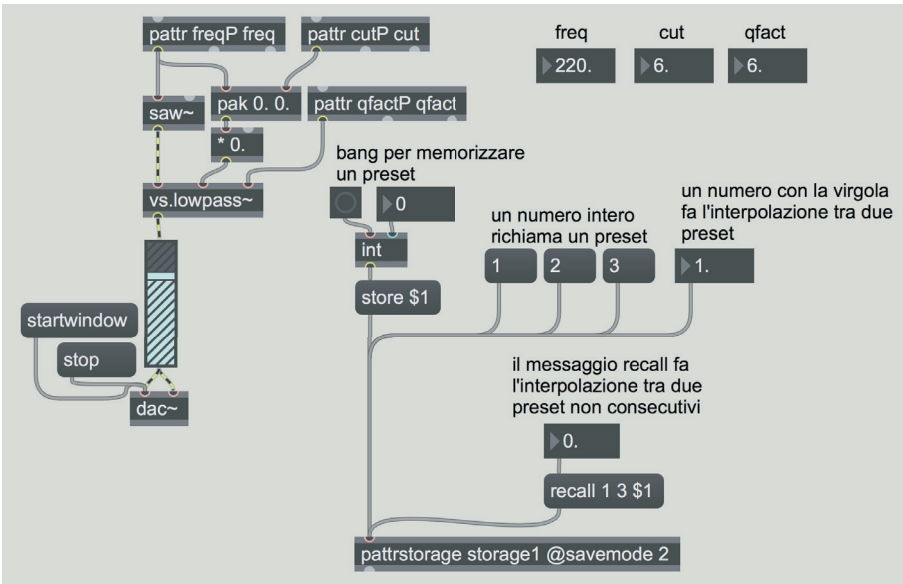


fig. ID.3: file **ID_01_pattrstorage.maxpat**

L'oggetto **pattrstorage** è il cuore del sistema evoluto di gestione dei *preset*. È con questo oggetto infatti che vengono memorizzati e richiamati i valori contenuti negli oggetti **pattr**.
I tre **flonum** in alto a destra hanno gli *scripting name* "freq", "cut" e "qfact" e di conseguenza sono legati ai tre oggetti **pattr** visibili a sinistra. Notate, infatti, che i tre oggetti **pattr** hanno due argomenti: il primo è lo *scripting name* del **pattr** stesso, il secondo è lo *scripting name* di uno dei tre *number box* in alto a destra. Attivate il "motore" DSP, e fate i clic sui tre *message box* al centro contenenti i numeri 1, 2 e 3: vengono selezionati tre *preset* che sono stati memorizzati con **pattrstorage**.

(...)

il capitolo prosegue con:

8T
L'ARTE DELL'ORGANIZZAZIONE DEL SUONO: PROCESSI DI MOVIMENTO

PREREQUISITI PER IL CAPITOLO

- Contenuti del volume 1 e dei capitoli 5, 6 e 7 (teoria)

OBIETTIVI

Conoscenze

- Conoscere varie modalità possibili di movimento dei suoni
- Conoscere le interrelazioni fra diversi tipi di movimento
- Conoscere i movimenti che mettono in evidenza il passaggio nella percezione da un parametro all'altro
- Conoscere diverse possibilità di movimento all'interno del timbro
- Conoscere limiti e ambiguità delle categorizzazioni dei movimenti

CONTENUTI

- I movimenti semplici, complessi e composti
- Le sequenze di movimenti
- Modalità di movimenti all'interno del timbro

ATTIVITÀ

- Esempi sonori e interattivi

VERIFICHE

- Test a risposte brevi di ascolto e analisi (massimo 30 parole)

SUSSIDI DIDATTICI

- Concetti di base - Glossario

Il campo nel quale la musica prende corpo è una temporalità fortemente spazializzata. Intendiamoci: non diventa visiva, la musica. Essa è e rimane nell'orecchio. Tuttavia la sua organizzazione, le sue connessioni logiche provengono alla nostra mente dal mondo visivo, dal mondo dello spazio.(…) Ci troviamo di fronte al respiro della materia…
(Salvatore Sciarrino, 1998)

La nostra metafora principale per la composizione musicale deve cambiare da quella dell'architettura a quella della chimica (…) Questa mutazione è la più radicale possibile: da un insieme finito di proprietà archetipiche scelte con cura e governate da principi "architettonici" tradizionali, si passa a un continuum di eventi sonori unici e la possibilità di espandere, plasmare e trasformare quel continuum in qualsiasi modo scegliamo, per costruire nuovi mondi di concatenazioni musicali.
(Trevor Wishart, 1994)

Non comporre più con le note ma con i suoni
Non comporre più solo suoni, ma la differenza che li separa
Agire su quelle differenze…controllare l'evoluzione (o non-evoluzione) del suono e la velocità della sua evoluzione
(Gerard Grisey, 1996)

8.1 COSA SONO I PROCESSI DI MOVIMENTO

INTRODUZIONE

Finora abbiamo realizzato suoni singoli o sequenze di suoni senza una particolare intenzione di esplorare l'organizzazione dei suoni. Da ora cominciamo a lavorare sulle possibili articolazioni dei suoni e sul loro movimento. In questo capitolo (e nel corrispettivo di pratica) la conoscenza della teoria e le abilità pratiche sviluppate sinora (sia nel campo dell'analisi all'ascolto del suono, sia nel campo della programmazione) saranno valorizzate ancora di più chiamando in campo la creatività del lettore e la sua capacità di costruire processi di movimento. Come già scritto nell'introduzione a questo volume, qui ci limiteremo ad articolazioni sonore che non superano un minuto: esploreremo un livello intermedio fra la micro-forma dei suoni singoli e la macro-forma di un'intera creazione sonora[1]. I processi di movimento dei suoni, in questo capitolo saranno quindi costruiti in astratto, al di fuori di una dimensione e di un contesto formale più ampio, cioè quello di una creazione sonora vera e propria, di cui parleremo nel prossimo volume. Ciò non significa che il percorso creativo debba seguire questa successione (dai singoli suoni, ai processi di movimento, alla forma generale): questo ordine ha uno scopo puramente didattico.

[1] Intendiamo, con il termine composizione, un'attività e un'esperienza che va oltre la composizione musicale in senso stretto, ma comprende la composizione di lavori di sound art o di sound design, o di colonne sonore di lavori audio-visivi, una soundscape composition etc.

Nella pratica, ogni artista del suono segue il proprio modo di costruire una forma nel tempo e/o nello spazio, anche, ad esempio, a partire dal progetto generale verso una specificazione degli aspetti sonori, lavorando contemporaneamente su tutti i livelli, o addirittura affidando al computer il compito di operare alcune scelte formali.
Nelle creazioni di installazioni interattive (anche in rete), poi, le forme assumono contorni non determinati dall'inizio, e lo scopo dei compositori/artisti è quello di creare un ambiente interattivo in cui la forma è sempre rideterminata dalle scelte degli utenti, all'interno di quell'ambiente.
Infine, un sound designer che voglia costruire il suono per immagini in movimento, o giochi interattivi o altro, si troverà a fare scelte formali in relazione all'ambiente creato da altri, o con altri soggetti.
Questo capitolo sarà utile a prescindere dalla forma finale del lavoro creativo che si sta compiendo con il suono. Quando si compone costruendo ed elaborando i timbri fino al più piccolo dettaglio, infatti, è importante saper creare e gestire i vari livelli (dal micro al macro) sulla base del proprio progetto creativo, e quindi anche saper ascoltare e valutare i risultati "zoomando" fuori e dentro i livelli.
Horacio Vaggione a questo proposito parla di "*loop* del *feedback* azione/percezione"[2]. "Come un pittore che lavora direttamente su una tela deve indietreggiare e distanziarsi per percepire i risultati della sua azione, valutandola in diverse prospettive spaziali, così il compositore ha a che fare con differenti misure del tempo[3]. Da ciò ricaviamo che, oltre al *loop* del *feedback* azione/percezione, dobbiamo aggiungere una nuova categoria, cioè una specie di "ascolto mobile", che consente di valutare su diversi piani temporali i risultati delle operazioni svolte. Alcuni di questi livelli temporali non si possono ascoltare direttamente, e bisogna perciò valutarli percettivamente mediante gli effetti spostandosi su altri livelli (più alti)." (Vaggione, 2001, p.60)[4]

Uno di questi spostamenti è appunto quello dal livello dei singoli suoni a quelli dei movimenti dei suoni stessi. In questo modo osserveremo gli effetti dei suoni osservandoli da un piano leggermente più elevato, ma non elevato abbastanza da parlare già di creazione di un'opera completa, in quanto questa presuppone un'idea e un progetto formale e livelli superiori di tempo, che affronteremo nel prossimo volume. In ogni caso questo capitolo di teoria e pratica serve anche a far sviluppare un "ascolto mobile" e la capacità di mettere in moto un *loop*

---

2 *The meaning of any compositional technique, or any chunk of musical knowledge, arises from its function in support of a specific musical action, which in turn has a strong bearing on the question of how this action is perceived. Action and perception lie at the heart of musical processes, as these musical processes are created by successive operations of concretization having as a tuning tool—as a principle of reality— an action/perception feedback loop.* (Vaggione, 2001, p.61)

3 Noi aggiungeremmo "e dello spazio".

4 *As a painter who works directly on a canvas must step back some distance to perceive the result of his or her action, validating it in a variety of spatial perspectives, so must the composer dealing with different time scales. This being so, a new category must be added to the action/perception feedback loop, a kind of "shifting hearing" allowing the results of operations to be checked at many different time scales. Some of these time scales are not audible directly and need to be validated perceptually by their effects over other (higher) time scales.* (trad. Cipriani)

di *feedback* azione/percezione, nel rapporto con la propria conoscenza, abilità e creatività.

I PROCESSI DI MOVIMENTO

I processi di movimento sono processi di evoluzione del suono che possono avvenire mediante la variazione dello spettro, o dello spazio o mediante l'interazione fra questi ed altri parametri[5].

Si possono avere movimenti plurimi e anche contrastanti o ambigui all'interno di una stessa sequenza sonora, ed è molto importante imparare a gestirli e a creare sequenze di movimenti complessi ripartendo dalla combinazione organica di quelli semplici, inventando veri e propri mondi sonori.
In questo capitolo (e in quello di pratica) indicheremo al lettore diversi percorsi di tipo tecnico per costruire sequenze sonore esemplificative dei movimenti che descriviamo.
Attenzione, però! Potrete trovarvi di fronte ad esempi in cui la definizione di alcuni movimenti ha un senso dal punto di vista tecnico e un senso diverso dal punto di vista percettivo: questo non deve rappresentare un problema, ma un'occasione per osservare come, nell'atto dell'organizzazione del suono[6], ci si trovi continuamente a contatto con contraddizioni e ambiguità e che queste vanno conosciute e accettate come elemento di ricchezza e polisemanticità.

---

5 Questo termine è mutuato da un articolo di Denis Smalley (Smalley 1986), fondamentale per la comprensione dei processi compositivi della musica elettronica e non solo. I processi di movimento per Smalley sono processi di evoluzione del suono che possono avvenire mediante la variazione dello spettro, o dello spazio o mediante l'interazione fra questi ed altri parametri. La prospettiva da cui Smalley osserva i movimenti del suono, tuttavia, non è di tipo tecnologico; l'elettroacustica e le tecniche usate nella produzione della musica non vengono considerate. La concentrazione quindi è focalizzata sull'ascolto e su ciò che del suono emerge dall'ascolto stesso. La terminologia con cui Smalley descrive le varie possibilità di movimento del suono è utile, in assenza di partiture, a comprendere meglio e ad ascoltare (nonché a comporre, se se ne hanno le competenze) con una maggiore consapevolezza un pezzo di musica elettronica. C'è però una importante differenza tra la concezione di Smalley e la nostra: nella nostra prospettiva, infatti, i processi di movimento vengono considerati dal punto di vista tecnico-creativo e non solo da quello dell'ascolto. La conseguenza è che l'idea di movimento unidirezionale, ad esempio, non riguarda nel nostro modo di intenderlo, solo un'ascesa o discesa riconoscibile all'ascolto nello spazio delle frequenze, ma in genere un aumento da valori minori a valori maggiori (o viceversa) in diversi campi percettivamente rilevanti, come quello delle durate, quello dell'ampiezza, etc. In questo modo si intende il movimento unidirezionale in modo più ampio, trasversale rispetto a parametri che invece Smalley descrive separatamente. Il lavoro di separazione dei parametri, utilissimo nell'ascolto analitico di un pezzo, deve lasciare spazio, nel lavoro di creazione sonora, a una prospettiva di complessità e di interrelazione che svilupperemo ulteriormente nel capitolo di avvio alla creazione sonora del prossimo volume. Lo scopo tecnico-compositivo di questo libro ci porta, in sostanza, ad operare un processo diverso da quello spettromorfologico di Smalley. Non si parte, cioè, dall'ascolto per arrivare alla definizione delle caratteristiche del suono e delle sue possibili evoluzioni, ma si parte dall'ascolto e dalle definizioni verso la creazione di movimenti sonori.
6 Questo è il termine più ampio per parlare di creazione sonora mediante l'uso delle nuove tecnologie, definito da Leigh Landy in un suo importante testo (Landy, 2007), anche in relazione al concetto di "suono organizzato" elaborato da Edgar Varèse.

Il nostro intento pertanto non è quello di costruire una teoria regolare e coerente, ma un percorso didattico aperto, interattivo, a volte problematico, dove il lettore però possa realmente e praticamente imparare a camminare, avventurarsi e fare le sue scelte e le sue scoperte.
Naturalmente è importante capire che in una prospettiva creativa, i nostri scopi, metodi e modi di osservare i parametri e le proprietà del suono sono ben diversi anche da chi si occupa di suono dal punto di vista fisico.
Ad esempio, durante la creazione musicale si può passare in modo 'disinvolto' dal dominio del tempo al dominio delle frequenze. Inoltre, nella percezione e quindi anche nella musica si mescolano parametri che hanno nature molto diverse, si passa attraverso diversi concetti di tempo e di spazio, o si esplorano zone di confine fra una proprietà e un'altra del suono. Di conseguenza, per poter parlare di arte dell'organizzazione del suono, dovremo fondere percorsi che per un fisico sarebbero inconciliabili (in quanto i necessari salti logici non potrebbero essere inquadrati o verificati mediante un metodo scientifico) e ci troveremo a rivedere alcuni concetti che abbiamo spiegato nei capitoli precedenti mediante una visione più ampia, forse meno rigorosa dal punto di vista scientifico, ma più vicina ai percorsi che i compositori, gli artisti del suono e i sound designer hanno bisogno di attraversare. In particolare considereremo frequenza, ampiezza e durata parametri basilari, la cui organizzazione nel tempo dà luogo ad altre proprietà più complesse del suono, come il timbro, il ritmo e la spazializzazione. Ci troveremo spesso a passare dagli uni agli altri senza soluzione di continuità, dando per scontato che questo capitolo è pensato come un percorso esperienziale da attraversare, e persino le categorie che introdurremo serviranno solo come punti di riferimento di passaggio per osservarne meglio i limiti e i confini.

CATEGORIE DI MOVIMENTO

Abbiamo scelto di utilizzare poche categorie, organizzate in ordine di complessità:
-**movimenti semplici**
cioè movimenti dei valori di un solo parametro in un suono
-**movimenti complessi** (che contengono movimenti semplici)
ossia movimenti dei valori di più parametri in un suono
-**movimenti composti** (che possono contenere movimenti semplici e/o complessi)
cioè movimenti dei valori di parametri in più suoni
-**sequenze di movimenti** (che possono contenere qualunque tipo di movimento)
successione di più movimenti in relazione fra loro
Queste categorie non sono assolute, possono esserci zone di ambiguità, e in generale sono basate su convenzioni che abbiamo adottato. Vediamo alcuni esempi di ambiguità:

1) Ambiguità della divisione fra i movimenti in un suono (movimenti semplici e complessi) e movimenti in più suoni (movimenti composti). Cosa è un suono singolo? Dipende dalla percezione: ci sono suoni che possono essere considerati sia come suoni singoli (il suono del mare) sia come composti da singole unità autonome (i suoni delle onde del mare). Dipende anche dalle intenzioni di chi lavora con il suono: si può far muovere un suono con molte componenti come oggetto singolo, oppure muovere i parametri delle sue componenti.

Il suono di un file audio o proveniente da un nostro algoritmo può contenere anche movimenti composti ma se il movimento si concentra su quel suono come elemento unico e ne aumentiamo l'intensità globale, quel movimento sarà semplice, perché variamo un solo parametro di un suono. Viceversa se interverremo separatamente sui vari suoni che compongono il suono composto, allora stiamo realizzando un nuovo movimento composto[7].

2) Ambiguità fra movimento semplice e complesso in un sistema che è a sua volta complesso. Prendiamo l'esempio dei filtri: per convenzione abbiamo inserito nella prima categoria un movimento ascendente della frequenza di taglio di un filtro. Tuttavia, questo movimento è semplice solo dal punto di vista dell'utente, il quale opera su un solo parametro. Ovviamente ciò che succede all'interno del filtro quando l'utente effettua questo movimento è qualcosa di più complesso.
La stessa cosa avviene in un sistema di spazializzazione stereo: il movimento da un estremo all'altro del parametro panning fra valori 0 e 1 è di tipo semplice in quanto lo possiamo implementare con un solo segmento di retta che fa muovere il suono da una parte all'altra. Ma è semplice solo se non osserviamo cosa c'è all'interno di un panner, cioè una doppia regolazione inversamente proporzionale delle intensità del canale destro e del canale sinistro. Tuttavia lo abbiamo inserito, per motivi pratici e logici, nella prima categoria.

Vaggione osserva che: "alcuni tipi di rappresentazione, che sono validi se applicati ad un livello, non mantengono la stessa pertinenza quando sono trasposti su un altro livello. Perciò quando si opera ad un multi-livello non si può escludere che avvengano fratture, distorsioni e discordanze fra i vari livelli. Per affrontare queste discordanze, bisogna "comporre" una strategia multi-sintattica" (ibidem)[8].
In questo senso, paradossalmente, abbiamo diviso i movimenti in categorie anche per poterne mostrare le zone grigie, le zone di sovrapposizione, che si trovano ad ogni passo nella pratica della musica elettronica e del sound design. Le riflessioni che seguiranno non hanno pertanto intenti normativi, né pretese di esaustività, praticamente impossibile nel campo della ricerca sonora, ma sono un viaggio nel suono e nei suoi movimenti, utile per acquisire una consapevolezza più profonda, premessa indispensabile per chi voglia avventurarsi nell'arte dell'organizzazione del suono.
Ora che abbiamo descritto brevemente i nodi più spinosi e i limiti relativi alla formalizzazione dei livelli, possiamo cominciare a descrivere i vari processi di movimento.
(...)

7 Anche il concetto di oggetto sonoro, teorizzato da Pierre Schaeffer, ci è sembrato passibile comunque della stessa ambiguità, e abbiamo preferito non introdurlo in questa fase.
8 *"some types of representation that are valid on one level cannot always retain their pertinence when transposed to another level. Thus, multi-level operations do not exclude fractures, distortions, and mismatches between the levels. To face these mismatches, a multi-syntactical strategy is "composed."* (trad. Cipriani)

il capitolo prosegue con:

Gesti e tessiture
Regolarità delle sequenze ritmiche e loro funzioni
Poliritmie e irregolarità ritmiche
Relazione figura/sfondo
Processi di moltiplicazione

- Esempi sonori e interattivi
- Test a risposte brevi di ascolto e analisi (massimo 30 parole)
- Concetti di base - Glossario

8P
L'ARTE DELL'ORGANIZZAZIONE DEL SUONO: PROCESSI DI MOVIMENTO

PREREQUISITI PER IL CAPITOLO

- Contenuti del volume 1, dei capitoli 5, 6 e 7 (teoria e pratica), degli interludi C e D e del capitolo 8T

OBIETTIVI

Abilità

- Saper prefigurare e programmare modalità diverse di movimento dei suoni e di direzioni di movimenti all'interno del timbro
- Saper costruire algoritmi per le interrelazioni fra diversi tipi di movimento e per sequenze di movimenti
- Saper applicare i valori dei parametri adatti per consentire all'ascoltatore di passare, nella percezione del suono elaborato, da un parametro all'altro

CONTENUTI

- Pratica dei processi di movimento semplici complessi e composti
- Pratica dei processi di movimento all'interno del timbro
- Pratica di costruzione di sequenze di movimenti

ATTIVITÀ

- Costruzione e modifiche di algoritmi
- Attività di reverse engineering

SUSSIDI DIDATTICI

- Lista oggetti Max

8.1 I PROCESSI DI MOVIMENTO

Dal momento che i processi di movimento sono descritti dettagliatamente nel capitolo 8T, la parte pratica consisterà per lo più in una serie di attività e di esercizi di *reverse engineering* e di analisi da svolgere parallelamente alla lettura della parte teorica. Notate che in questo caso il *reverse engineering* non deve essere volto alla ricostruzione del suono, ma del *movimento* a cui viene sottoposto il suono.
Tutte le tecniche apprese nei capitoli precedenti potranno essere usate per realizzare dei piccoli studi di musica elettronica e di sound design.
Il capitolo è breve ma ci sarà molto da lavorare: cominciamo senz'altro con la prima categoria di movimento.

8.2 MOVIMENTI SEMPLICI

MOVIMENTI UNIDIREZIONALI - FREQUENZA

Applicare un movimento semplice alla frequenza vuol dire essenzialmente realizzare un glissando: gli oggetti che ci servono in questo caso sono naturalmente `line~` e `curve~`: si tratta di tecniche che conosciamo benissimo e sulle quali non ci dilungheremo.

Un movimento semplice che proceda per gradi, ovvero che non sia continuo, può essere realizzato in diversi modi. Ricostruite la *patch* di figura 8.1.

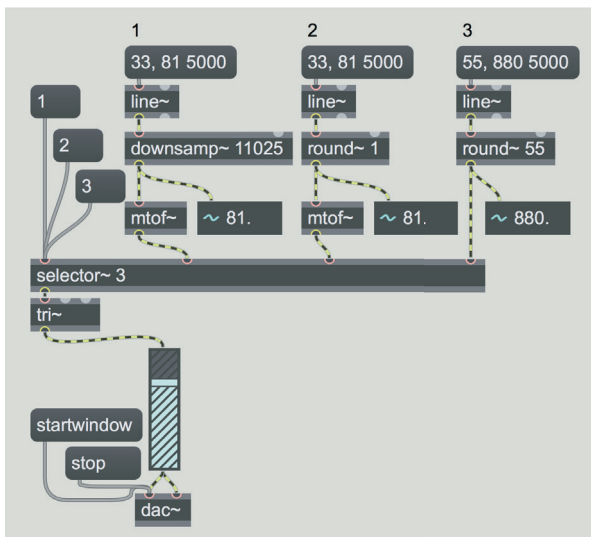


fig. 8.1: tre diversi modi di quantizzare un glissando

In tutti e tre i casi abbiamo un movimento unidirezionale semplice dal LA0 al LA4. Il primo "glissando quantizzato" è realizzato tramite l'oggetto `downsamp~`. Questo oggetto effettua un sottocampionamento del segnale in ingresso tramite *sample and hold*; l'argomento indica il periodo di campionamento espresso in campioni. In figura quindi campioniamo (e manteniamo) un

valore ogni 11025 campioni; ovvero, supponendo un *sample rate* di 44100 hz, ogni quarto di secondo. Il glissando è effettuato sui valori MIDI, che vengono convertiti in frequenza dopo il *sample and hold*.
Nel secondo esempio usiamo l'oggetto [**`round~`** 1], che arrotonda il segnale secondo un certo intervallo. L'argomento indica appunto l'intervallo dell'arrotondamento: 1 arrotonda al numero intero più vicino, 2 arrotonda ai multipli di 2, 3 ai multipli di 3 e così via. È anche possibile arrotondare i decimali: l'argomento 0.5 ad esempio produrrebbe valori multipli di 0.5. La quantizzazione, nel caso illustrato in figura, produce una serie di semitoni ascendenti.
Nel terzo esempio realizziamo un glissando di frequenze (non di note MIDI) e arrotondiamo ai multipli di 55, in pratica realizziamo un glissando di armoniche.

ATTIVITÀ

- Trovate almeno altre tre tecniche per quantizzare un glissando.
- Fate il *reverse engineering* dell'Esempio sonoro 8A.1, utilizzando un suono campionato a vostra scelta.

MOVIMENTI UNIDIREZIONALI – DURATE E RITMO

Per questo tipo di movimento possiamo usare la tecnica illustrata nella *patch* 05_13_blocks_tech_accel.maxpat (vedi paragrafo 5.4P).
Per introdurre una irregolarità nel ritmo possiamo servirci dell'oggetto `vs.randmetro` (vedi paragrafo IC.3).

ATTIVITÀ

- Fate il *reverse engineering* degli Esempi sonori 8A.2, utilizzando un suono campionato a vostra scelta.
- Create un movimento unidirezionale del ritmo che vada da una scansione regolare ad una irregolare.
- Create un movimento di accorciamento progressivo del suono mantenendo fisso il ritmo di generazione.
- Create degli accelerando e dei rallentando tramite *delay* multipli (prendete ispirazione dalla *patch* 06_02_multitap2.maxpat, usando almeno 32 *tap*).

(...)

il capitolo prosegue con:

9T
MIDI

PREREQUISITI PER IL CAPITOLO

- Contenuti del volume 1 e dei capitoli 5, 6, 7 e 8 (teoria)

OBIETTIVI

Conoscenze

- Conoscere il protocollo MIDI
- Conoscere la struttura e l'utilizzo dei messaggi di canale e di sistema
- Conoscere gli utilizzi di base di *controller* MIDI

CONTENUTI

- Protocollo MIDI: connessioni e messaggi
- Moduli trasmettitori e moduli ricevitori: i percorsi del segnale MIDI
- Struttura e utilizzo dei Channel Voice Message e dei Channel Mode Message
- Struttura e utilizzo dei System Real Time Message
- I controller MIDI: dalle interfacce pseudo-strumentali alle superfici di controllo
- I controller MIDI avanzati: dall'utilizzo dei MIDI Data Glove al Gesture Mapping

VERIFICHE

- Test a risposte brevi (massimo 30 parole)

SUSSIDI DIDATTICI

- Concetti di base - Glossario

9.1 LO STANDARD MIDI

Lo scambio di informazioni tra strumenti musicali elettronici, sistemi di controllo e computer è spesso attivato tramite il *protocollo* MIDI, uno standard creato agli inizi degli anni '80 e ancora molto diffuso. Il termine MIDI è un acronimo per *Musical Instrument Digital Interface*.
Il protocollo MIDI è usato per applicazioni molto diverse fra loro. In questo capitolo tratteremo, del MIDI, solo ciò che è essenziale per i nostri scopi, unitamente ad alcune informazioni di base.

I dispositivi MIDI possono essere di due tipi:
- *dispositivi di controllo* (*controller*), che servono a generare messaggi MIDI
- *dispositivi sonori* (*sound module*), che utilizzano i messaggi MIDI ricevuti per produrre o modificare suoni.

Questi ultimi possono essere a loro volta divisi in due tipi:
- strumenti, come i sintetizzatori e i campionatori, che generano suoni,
- moduli di elaborazione del suono come *delay*, riverberi ed altri effetti che invece servono a modificare fonti sonore esterne.

Uno stesso dispositivo, ad esempio un computer, può comportarsi, a seconda dei casi, da trasmettitore o da ricevitore. Va prima ricordato però che un computer non può comunicare in MIDI con l'esterno se non è dotato di un collegamento ad esempio un'interfaccia MIDI (o MIDI *Interface Card*). In realtà la gran parte degli strumenti MIDI dispongono di un collegamento digitale diretto con il computer (ad esempio USB) con il quale ricevere e trasmettere dati MIDI senza dover utilizzare un'interfaccia MIDI apposita.
Nella figura è riprodotto il frontalino di un'interfaccia MIDI con due prese MIDI Out, due MIDI In e la presa USB per il collegamento con il computer.

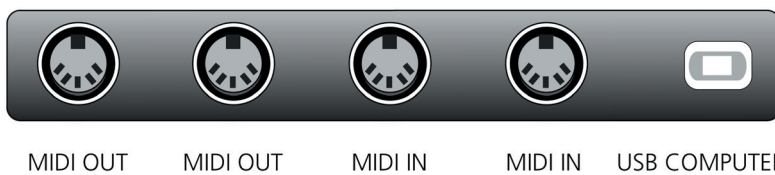


fig. 9.1: frontale interfaccia MIDI

Oggi le applicazioni del MIDI sono numerosissime e fortemente intersecate con quelle legate al campo audio professionale. Questo protocollo, infatti, è in continua evoluzione. Oggi il MIDI è usato nei computer, in internet, negli apparecchi di telefonia mobile, nel controllo di sistemi complessi per il suono, di sistemi luci, sistemi multimediali e quant'altro.
Esistono diversi altri protocolli di comunicazione, anche più veloci e flessibili del MIDI (come ad esempio OSC), i quali però sono attualmente non diffusi sugli strumenti musicali elettronici (come tastiere, chitarre MIDI etc.). Il MIDI rimane perciò in uso a causa della sua semplicità d'uso e per la sua forte penetrazione nel campo della produzione commerciale di strumenti musicali.

9.2 MESSAGGI MIDI

Diciamo subito che i messaggi MIDI non contengono alcun suono, bensì il *modo* in cui un determinato suono, residente in una memoria, deve essere suonato; in altre parole qual è la nota da suonare, quando tale nota deve iniziare e quando deve terminare, con quale dinamica etc. È molto importante dunque distinguere un file audio (il quale contiene la forma d'onda del suono, e con essa la durata, l'inviluppo, il timbro, la frequenza etc.) da un messaggio MIDI che fornisce solo le informazioni relative alle modalità con cui un dato suono può essere suonato, e non la forma d'onda. Quest'ultima infatti è residente in una memoria che è separata dal messaggio MIDI. Ad esempio, una tastiera trasmette i dati MIDI che attivano un modulo di generazione sonora che li riceve. Tale modulo può avere, residenti in memoria, varie forme d'onda o circuiti per la generazione del suono che vengono "azionati" dai messaggi MIDI che provengono dalla tastiera. Fin qui tutto è chiaro: la tastiera manda istruzioni MIDI al modulo sonoro il quale le riceve, genera un suono secondo le istruzioni che arrivano in ingresso e lo manda in uscita mediante la propria scheda audio. Il motivo per cui spesso si genera confusione è che esistono due tipi di tastiere:

1) le tastiere mute o *Master Keyboard*, le quali svolgono la funzione di semplice trasmettitore, cioè la funzione di MIDI *Keyboard Controller*. Questo tipo di tastiere genera esclusivamente codici MIDI che vengono inviati a ricevitori esterni tramite il MIDI Out.
2) le tastiere con funzione di campionatore o sintetizzatore, cioè quelle che non generano solo codici MIDI, ma contengono anche un modulo di generazione sonora. Questo tipo di tastiere, come i computer, possono funzionare sia da trasmettitori (tramite il MIDI Out) sia da ricevitori (tramite il MIDI In), e inoltre hanno un collegamento interno che fa comunicare la parte che trasmette messaggi MIDI con la parte di generazione sonora. Quando premiamo un tasto su una tastiera di questo tipo, attiviamo messaggi MIDI che possono essere mandati sia ad un modulo esterno (tramite il MIDI Out) sia al modulo interno il quale genera suoni che escono dall'uscita audio della tastiera. Anche in questo caso i codici MIDI trasmessi dalla tastiera al modulo sonoro esterno o a quello interno *non contengono alcun suono*; il suono è generato dal modulo sonoro esterno o da quello interno.

Come vedremo nel prossimo paragrafo, oltre le tastiere, fra i *controller* di tipo strumentale troviamo anche chitarre MIDI, strumenti ad arco e a fiato MIDI, percussioni e batterie MIDI, pedaliere MIDI (o *Key Pedalboard*, a una o due ottave, utilizzate per i suoni gravi, come quelle dell'organo) etc.
Si tratta di *controller* che utilizzano forme e modalità simili a quelle dei rispettivi strumenti acustici ma, ancora una volta, esse servono ad inviare solo messaggi MIDI ad un modulo sonoro, che può essere anche esterno; tale modulo potrà contenere suoni coerenti con l'interfaccia (ad esempio suoni di chitarra attivati da un MIDI *Guitar Controller*) oppure no (ad esempio un suono di flauto o clacson di automobile attivato da un clarinetto MIDI).

Chiariti questi concetti generali su cosa *non contiene* un messaggio MIDI, cerchiamo di capire ora *cosa può contenere*. Iniziamo con un esempio: un modulo sonoro risponde a un messaggio di tasto premuto (*Note On*) e di tasto rilasciato (*Note Off*) proveniente da una tastiera muta, ed anche all'informazione di dinamica associata alla pressione del tasto. Questa viene chiamata *Key Velocity*, perché viene misurata come la velocità con la quale il tasto viene abbassato, che a sua volta è proporzionale alla dinamica con la quale si suonerebbe una nota sul pianoforte.
Poiché in MIDI tutto è espresso mediante numeri, bisogna conoscere la convenzione che viene adottata. Questa convenzione (come sappiamo dal par. 1.4T) prevede che a ogni nota (o meglio, a ogni tasto) corrisponda un numero (Key Number): il DO centrale è uguale a 60, Il DO# a 61, il RE a 62, il RE# a 63 etc.
Le note ammesse sono quelle da 0 a 127, anche normalmente gli strumenti MIDI non sono in grado di produrre l'intera gamma: una tastiera MIDI con l'estensione di un pianoforte ad esempio può produrre le note da 21 a 108.[1]
Analogamente, a ogni dinamica corrisponde un numero, variabile anche qui fra 0 e 127. La variazione nel dispositivo sonoro è di solito di tipo logaritmico, in modo tale da rispettare la percezione dell'orecchio, e da fare sì che alla stessa differenza di numero corrisponda un'uguale variazione dinamica.
Il protocollo MIDI consente anche, entro certi limiti, di superare il sistema temperato[2], sia utilizzando, negli strumenti appositamente predisposti, temperamenti e intonazioni differenti dal temperamento equabile (ma ciò è strettamente dipendente dallo strumento utilizzato), sia inviando, prima di ciascuna nota, un'informazione che specifica la variazione di intonazione rispetto alla frequenza nominale. Questa informazione è detta in inglese *Pitch Bend*, e ha lo stesso effetto di una variazione del comando fisico presente su quasi tutti gli strumenti MIDI, detto *Pitch Bend Wheel* o *Pitch Bender*. Questo controllo, in genere in forma di leva o di ruota, consente di alterare momentaneamente l'intonazione.

Fig. 9.2: *pitch bender*

[1] Nel caso in cui si abbia a disposizione un *controller* con un numero di ottave minore rispetto ai suoni disponibii nel sound module l'utente può modificare il posizionamento del suono assegnandolo ad un'ottava più alta o più bassa fino a che non rientri nell'estensione effettiva dello strumento.
[2] Sul sistema temperato vedi par. 1.4 del volume di teoria.

Inoltre è possibile scegliere fra diversi tipi di suoni (timbri o *program*) differenti, memorizzati nello strumento stesso. Anche per questo vi è un apposito messaggio MIDI per selezionare il tipo di suono (*Program Change*).

Analizziamo più da vicino le due tipologie di *MIDI Message*:
- messaggi di canale (*Channel Message*)
- messaggi di sistema (*System Message*).

MESSAGGI DI CANALE
Se a uno stesso trasmettitore (per esempio un computer) è collegato più di uno strumento, è importante poter stabilire un modo per indirizzare messaggi MIDI a uno strumento in particolare, altrimenti tutti suonerebbero la stessa "parte". Perciò a un messaggio MIDI è associato un *numero di canale*, e ogni strumento prende in considerazione solamente quei messaggi che lo riguardano, cioè quei messaggi che hanno lo stesso numero di canale per il quale lo strumento è stato predisposto dall'utente. I messaggi di canale, quindi, sono indirizzati specificamente ad un canale, e verranno ricevuti (ed eseguiti) solo dai dispositivi che ricevono su quel particolare canale. I *Channel Message* sono di due tipi: i *Channel Voice Message*, che riguardano le modalità e i tempi di esecuzione e i *Channel Mode Message*, che indicano il modo in cui risponde il dispositivo ricevente. Vediamo dapprima i principali **Channel Voice Message**: *Note On*, *After Touch di Canale*, *After Touch* Polifonico, *Note Off*, *Program Change*, *Pitch Bend*, *Control Change*.

NOTE ON = tasto premuto
Corrisponde all'abbassamento di un determinato tasto della tastiera, o comunque all'attivazione di una nota tramite uno dei *controller* di cui abbiamo parlato; il messaggio comprende anche l'informazione riguardante il numero di canale, il numero di nota o *Key Number* e la *Key Velocity* (in questo caso ci riferiamo all'*Attack Velocity* cioè alla velocità con cui viene premuto il tasto).
Ad esempio un messaggio MIDI per suonare il DO centrale sarà composto di tre informazioni: 144 60 120. Il primo numero (detto *Status Byte*) è 144 e indica "*Note On*" cioè il messaggio di tasto premuto; il secondo numero (detto *First Data Byte*) è 60 e indica il numero di nota o *Key Number* (il DO dell'ottava centrale); il terzo numero (detto *Second Data Byte*[3]) è 120 e si riferisce all'*Attack Velocity* (il cui massimo sarebbe 127). E il canale MIDI? L'informazione del canale MIDI si desume dal primo numero. Per attivare un *Note On*, infatti, si può scegliere un numero fra 144 e 159. Se il primo numero è 144, come in questo caso, la nota viene trasmessa solo sul canale MIDI 1, se il primo numero è 145, la nota viene trasmessa solo sul canale MIDI 2, e così via fino al numero 159, che implica una trasmissione solo sul canale 16.

[3] Da notare che in alcuni casi ci si riferisce allo *Status Byte* come 1° Byte, al primo *Data Byte* come secondo Byte, e al secondo Data Byte come terzo Byte. La sostanza è identica, dipende se si dividono i Byte in due gruppi (*Status Byte* e *Data Byte*), oppure se si numerano i Byte in generale secondo il loro ordine.

MESSAGGIO NOTE ON			
	Byte di stato	**1° Byte di dati**	**2° Byte di dati**
Messaggio trasmesso	Note ON + Canale MIDI	Numero di Nota	Attack Velocity
Range di valori trasmesso	144 – 159	0 – 127	0 – 127
Contenuto del messaggio	144 = Note ON canale MIDI 1 145 = Note ON canale MIDI 2 etc..... .....fino a 159 = Note ON canale MIDI 16	0 = Do prima ottava[4] 1 = Do# prima ottava etc..... .....fino a 127= Sol ultima ottava	0 = nessun attacco 1 = ampiezza minima etc..... .....fino a 127= ampiezza massima

(...)

[4] Esistono diversi modi di numerare le ottave, a seconda degli standard. Qui si intende per prima ottava quella più grave indicata in MIDI come ottava –2, e per ultima ottava quella più acuta (8). Dalla –2 alla 8 (che nel MIDI non è completa ma arriva solo fino al sol, data la limitazione a 128 note) ci sono 10 ottave e mezza. Il Do centrale della tastiera di un pianoforte è indicato in MIDI come Do dell'ottava 3. Nel MIDI le note sono indicate secondo il sistema americano, perciò DO=C, DO#=C#, RE=D etc. L'indicazione A3, ad esempio, significa il LA sopra il DO centrale (440 Hz).

il capitolo prosegue con:

Channel pressure o aftertouch di canale
After touch polifonico o polyphonic key pressure
Note off
Program change
Pitch bend
Control change
Messaggi di sistema

9.3 I CONTROLLER MIDI
Controller semplici
Superfici di controllo
Sensori MIDI, MIDI data glove e motion tracking

- TEST A RISPOSTE BREVI (MASSIMO 30 PAROLE)
- CONCETTI DI BASE
- GLOSSARIO

9P
MIDI E CONTROLLO IN TEMPO REALE

PREREQUISITI PER IL CAPITOLO

- Contenuti del volume 1, dei capitoli 5, 6, 7 e 8 (teoria e pratica), degli interludi C e D e del capitolo 9T

OBIETTIVI

Abilità

- Saper gestire i flussi di segnali MIDI con Max all'interno di un sistema di dispositivi virtuali
- Saper gestire i flussi di segnali MIDI (anche in polifonia) fra dispositivi MIDI hardware e software mediante Max.

CONTENUTI

- Oggetti MIDI di Max e loro funzioni nella gestione dei messaggi
- Gestione avanzata della polifonia in MIDI fra Max e dispositivi hardware MIDI esterni

SUSSIDI DIDATTICI

- Lista oggetti Max - Lista attributi per oggetti Max specifici

9.1 MIDI E MAX

Finora, nei capitoli di pratica, abbiamo solo sfiorato l'argomento del MIDI. In questo capitolo approfondiremo solo alcuni aspetti del rapporto fra MIDI e Max, per dare indicazioni fondamentali sulla loro gestione combinata.
Oltre ai messaggi di nota (di cui abbiamo accennato nel par. IB.1), è possibile inviare via MIDI messaggi diversi che possono servire a modificare i parametri dello strumento (o dell'effetto) che li riceve.
Finora abbiamo usato il MIDI per comunicare con il sintetizzatore interno al sistema operativo, ma è naturalmente possibile utilizzare qualunque dispositivo MIDI.

Dal menu Options potete richiamare la voce *MIDI Setup*: apparirà una finestra contenente l'elenco di tutti i dispositivi MIDI reali e virtuali connessi al computer (fig. 9.1).

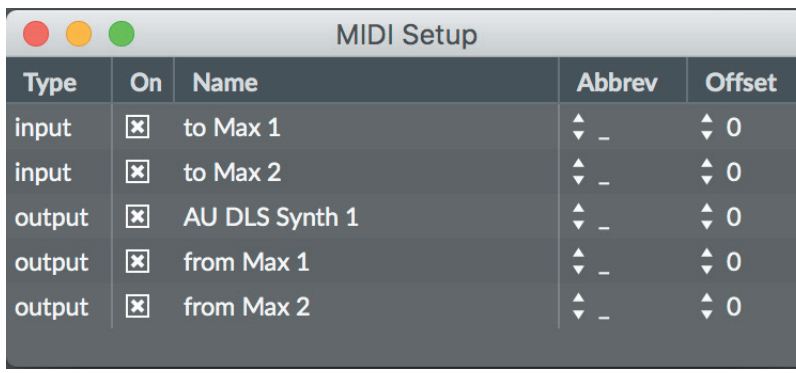


fig. 9.1: finestra *MIDI Setup*

In questa finestra abbiamo l'elenco dei dispositivi di *input* (che possono cioè mandare messaggi MIDI a Max) e di *output* (che possono ricevere messaggi MIDI da Max): come si vede alcuni dispositivi possono funzionare sia da *input* sia da *output*; è il caso ad esempio di un'interfaccia MIDI fisica (non virtuale) connessa al computer e che disponga almeno di una connessione MIDI IN e una MIDI OUT.

Ciascun dispositivo può gestire generalmente 16 canali di comunicazione per i dati MIDI. Un dispositivo, infatti, può controllare più strumenti contemporaneamente, alcuni dei quali possono essere politimbrici, cioè possono produrre contemporaneamente suoni con timbri diversi: i canali servono ad indirizzare i messaggi ai diversi strumenti (o alle diverse sezioni di uno strumento politimbrico). Se ad esempio ad un particolare dispositivo MIDI colleghiamo un pianoforte digitale, un sintetizzatore politimbrico e un riverbero, potremmo decidere di mandare i messaggi MIDI per il pianoforte digitale sul canale 1, quelli per il sintetizzatore sui canali 2, 3, 4 e 5 (dove avremo programmato quattro timbri diversi), e quelli per il riverbero sul canale 6. L'oggetto `noteout`, che abbiamo usato spesso in questi due volumi, dispone di tre ingressi, il primo per il valore di nota MIDI, il secondo per la *velocity* e il terzo per il canale MIDI.

Ogni dispositivo ha un nome, e tramite la finestra *MIDI Setup* è possibile definire un'abbreviazione e un *offset* di canale[1]. Gli oggetti Max MIDI possono riferirsi ad un particolare dispositivo tramite uno di questi parametri (utilizzati normalmente come argomenti dell'oggetto).
Nella finestra *MIDI Setup* è possibile attivare o disattivare i dispositivi facendo clic sul relativo checkbox (colonna "*On*") e modificare l'abbreviazione (una singola lettera minuscola) e l'*offset* di canale.
Notate il dispositivo "AU DLS Synth 1" che è il sintetizzatore virtuale di Mac OSX (l'analogo Windows si chiama Microsoft DirectMusic DLS Synth) che abbiamo usato finora negli esempi MIDI.
Una parola sui dispositivi "to Max 1", "to Max 2", "from Max 1" e "from Max 2": si tratta di collegamenti virtuali tra Max e altri programmi all'interno dello stesso computer. Questi dispositivi sono disponibili solo in Mac OSX. Se dopo aver avviato Max lanciate un programma che utilizza il MIDI (ad esempio un *sequencer*), vedrete, tra le interfacce riconosciute dal programma anche i dispositivi virtuali di Max: potrete così mandare dei messaggi MIDI a Max usando la porta "to Max 1" (o 2) e ricevere messaggi MIDI da Max dalla porta "from Max 1" (o 2). Per ottenere la stessa funzionalità in Windows è necessario utilizzare software di terze parti, come LoopBe1 (http://www.nerds.de) o LoopMIDI (http://www.tobias-erichsen.de).

9.2 GESTIONE DEI MESSAGGI MIDI

Oltre al già noto oggetto `noteout` che serve ad inviare messaggi di nota MIDI, esiste l'oggetto `notein` che riceve i messaggi di nota da un dispositivo esterno (ad esempio una tastiera hardware). Inoltre sono disponibili in Max altri oggetti in grado di gestire messaggi MIDI; ne vediamo alcuni in fig. 9.2.

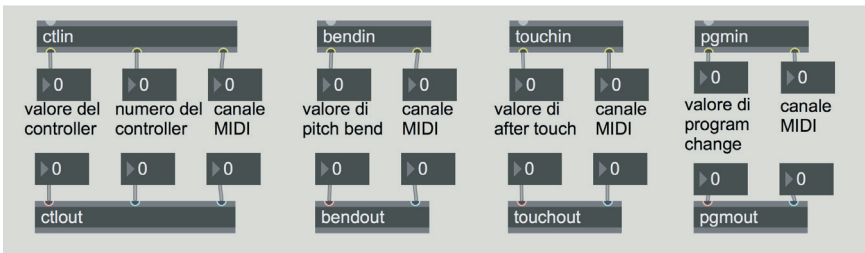


Fig. 9.2: altri oggetti MIDI

[1] I canali MIDI, come abbiamo detto, sono 16, e sono numerati da 1 a 16. Tramite l'offset è però possibile utilizzare valori superiori: in un dispositivo, infatti, il valore di offset si aggiunge al numero di canale. Se ad esempio, come appare in figura, il dispositivo *b* ha un offset pari a 16, significa che i suoi canali 1-16 vengono visti da Max come canali 17-32. Questo fa sì che quando inviamo (o riceviamo) in Max un messaggio di nota al canale 17, lo stiamo in realtà inviando (o ricevendo) al canale 1 del dispositivo *b*. In un oggetto MIDI possiamo specificare, come argomenti, il nome del dispositivo e il canale, oppure solo il canale con l'eventuale offset. In altre parole, sempre con riferimento alle impostazioni di figura 9.1, gli oggetti [`noteout` b 2] e [`noteout` 18] inviano il messaggio allo stesso canale, ovvero il secondo canale del dispositivo *b*.

Gli oggetti **ctlin** e **ctlout** ricevono e inviano i messaggi di *control change*: questi messaggi vengono solitamente generati da meccanismi (*controller*) in grado di trasmettere una serie continua di valori numerici, come ad esempio le rotelle di modulazione (*modulation wheel*) che troviamo in molte tastiere MIDI (spesso utilizzate per gestire il vibrato del suono), oppure pedali, cursori etc. (vedi anche cap. 9.3T). Per ogni canale MIDI sono disponibili 128 *controller* (numerati da 0 a 127) ciascuno dei quali può trasmettere o ricevere valori compresi tra 0 e 127. Il *control change* viene usato per modificare le caratteristiche sonore di uno strumento (aggiungendo ad esempio un vibrato al suono, o modificando la frequenza di taglio di un filtro) o di un effetto (in un distorsore, come quelli usati per la chitarra elettrica, si può usare ad esempio per regolare la quantità della distorsione).
Gli oggetti **bendin** e **bendout** ricevono e inviano i messaggi di *pitch bend*: cioè valori (compresi tra 0 e 127) che determinano l'alterazione dell'altezza di una nota, utile per simulare i piccoli glissandi di strumenti a corda come la chitarra, o a fiato come clarinetto e sax. Il relativo meccanismo di controllo presente nelle tastiere MIDI è spesso una rotella.
Gli oggetti **touchin** e **touchout** ricevono e inviano i messaggi di *after touch*: questi messaggi sono generati, in una tastiera MIDI, dalla pressione sul tasto dopo che è stata suonata una nota. Vengono utilizzati in genere per alterare il suono, ad esempio modificando il volume, o la frequenza di taglio del filtro, etc.: hanno quindi un uso simile a quello del *control change*. Anche questi valori sono compresi tra 0 e 127. Il messaggio di *after touch* è globale per tutta la tastiera: ovvero viene inviato lo stesso messaggio qualunque sia il tasto sottoposto a pressione. Esiste un altro messaggio, *polyphonic key pressure* (non illustrato in figura), che invia un valore distinto per ogni tasto premuto: si tratta di una funzione che si trova raramente nelle tastiere, quasi sempre dotate del solo *after touch globale*.
Gli oggetti **pgmin** e **pgmout** ricevono e inviano i messaggi di *program change*: si tratta dell'equivalente di un cambio di *preset*. Inviando un messaggio di *program change* (un valore compreso tra 0 e 127) ad un sintetizzatore (reale o virtuale) è ad esempio possibile cambiare il timbro. Notate che in Max questi valori sono compresi tra 1 e 128, e vengono convertiti in valori compresi tra 0 e 127 (vengono cioè diminuiti di 1) prima di essere inviati al dispositivo.
Gli oggetti MIDI possono avere come argomento il nome del dispositivo[2], il numero del canale o entrambi (vedi nota 1).

[2] Il nome del dispositivo può essere l'abbreviazione costituita da una singola lettera o il nome esteso come appare nella colonna "Name" della finestra *MIDI Setup*; quando il nome esteso è costituito da più parole separate da spazi va racchiuso tra virgolette. Con riferimento alla figura 9.1, quindi, possiamo usare per il primo dispositivo l'abbreviazione a oppure il nome esteso "DDMB2P Port 1".

Nel caso degli oggetti che ricevono messaggi MIDI, se il numero di canale viene specificato come argomento, l'*outlet* relativo al canale scompare: vedi fig. 9.3.

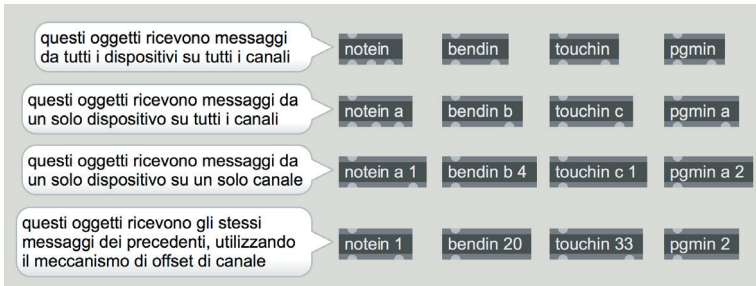


Fig. 9.3: argomenti negli oggetti che ricevono messaggi MIDI

In questa figura abbiamo un esempio per ogni combinazione di argomenti: notate che le ultime due righe di oggetti hanno un *outlet* in meno, quello corrispondente al canale MIDI, perché il canale è specificato nell'argomento. Impostando nella finestra *MIDI Setup* le abbreviazioni dei dispositivi e gli *offset* di canale come illustrato in figura 9.1, gli oggetti dell'ultima riga riceveranno il messaggio dallo stesso dispositivo e sullo stesso canale dei corrispondenti oggetti della penultima riga.
Gli oggetti **`ctlin`** e **`ctlout`** hanno una gestione degli argomenti leggermente diversa: i possibili *controller* per canale MIDI sono infatti 128, numerati da 0 a 127, e ciascuno può generare valori tra 0 e 127. Se nell'oggetto è presente un solo argomento numerico, si tratta in questo caso del numero di *controller*. Questo argomento può essere preceduto dal nome del dispositivo, oppure seguito dal numero di canale, o da entrambi. Se vogliamo specificare un canale senza specificare un *controller* l'argomento relativo al *controller* deve essere -1 (fig. 9.4).

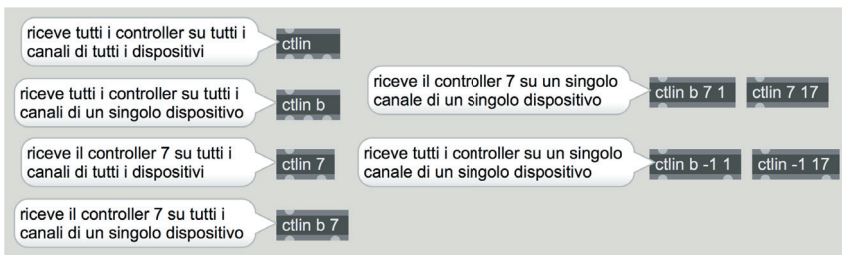


Fig. 9.4: argomenti nell'oggetto **`ctlin`**

La prima uscita corrisponde sempre al valore generato dal *controller*. Nelle combinazioni che presentano due sole uscite, la seconda uscita corrisponde, nel caso degli oggetti [**`ctlin`** 7] e [**`ctlin`** b 7], al numero di canale MIDI, mentre nel caso degli oggetti [**`ctlin`** b -1 1] e [**`ctlin`** -1 17] corrisponde al numero di *controller*: riflettete un momento e spiegate perché.

Se facciamo doppio clic, in modalità *performance*, su un oggetto MIDI, apparirà un menù contestuale da cui è possibile selezionare uno dei dispositivi disponibili, che potrà sostituire l'eventuale dispositivo indicato come argomento.

Vediamo ora una *patch* che ci permette di modificare il suono di un sintetizzatore software o hardware collegato via MIDI. Aprite il file **9_01_MIDI_synth.maxpat** (fig. 9.5).

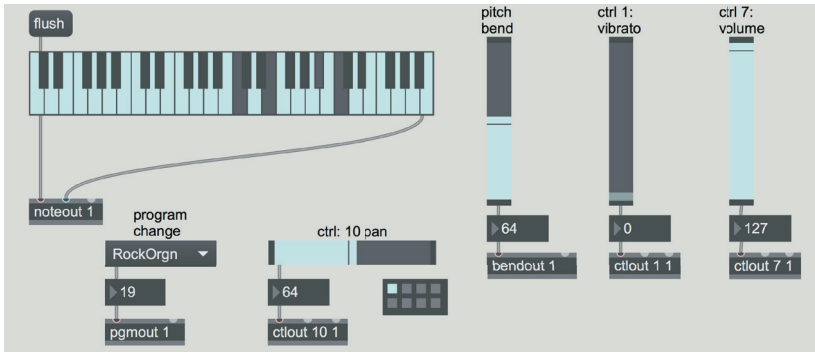


Fig. 9.5: file **9_01_MIDI_synth.maxpat**

Questa *patch* si collega di *default* al sintetizzatore interno del nostro computer, ma modificando le impostazioni nella finestra *MIDI Setup* è possibile utilizzare un altro dispositivo.

(...)

il capitolo prosegue con:

Interludio E
MAX FOR LIVE

PREREQUISITI PER IL CAPITOLO

- Contenuti del volume 1, dei capitoli 5, 6, 7, 8 e 9 (teoria e pratica) e degli interludi C e D
- Conoscenza delle funzioni principali del programma Ableton Live

OBIETTIVI

Abilità

- Saper creare device Max for Live
- Saper controllare l'ambiente Live tramite le Live API

CONTENUTI

- Costruzione di device Audio e MIDI tramite Max for Live
- Costruzione di Strumenti Virtuali tramite Max for Live
- Uso delle Live API
- La struttura gerarchica del Live Object Model

ATTIVITÀ

- Costruzione e modifiche di algoritmi

SUSSIDI DIDATTICI

- Lista oggetti Max - Lista attributi, messaggi e azioni per oggetti Max specifici - Glossario

IE.1 INTRODUZIONE A MAX FOR LIVE

Premessa: per poter seguire questo Interludio è necessario possedere le licenze d'uso di Max for Live e di Ableton Live.[1] Le informazioni qui contenute riguardano quasi esclusivamente l'uso di Max for Live e non sono necessarie per la comprensione dei capitoli successivi. Chi non possiede le suddette licenze può quindi saltare questo Interludio senza problemi.

Che cos'è Ableton Live

Ableton Live (o più semplicemente Live), è un'applicazione DAW (*Digital Audio Workstation*); ovvero un software che permette di registrare, manipolare e riprodurre tracce audio. Oltre a ciò, gestisce sequenze MIDI con cui può controllare strumenti hardware esterni o strumenti virtuali all'interno dell'applicazione stessa.
La caratteristica peculiare di Live è la possibilità di gestire le tracce audio e MIDI in modo non lineare. Mentre la gran parte dei sistemi DAW e dei sequencer infatti pone le sequenze lungo una timeline che scorre, appunto, linearmente, in Live è possibile attivare le sequenze indipendentemente l'una dall'altra, e realizzare quindi una sorta di arrangiamento in tempo reale, liberamente modificabile durante l'esecuzione.
Dal momento che questo testo non è un manuale di Live, daremo per scontata la conoscenza delle principali funzioni del programma: in particolare la struttura del Live set e delle sue sezioni, inclusa la sezione di *Help* e le lezioni di Live, l'uso delle *clip* audio e MIDI nelle modalità "Session View" e "Arrangement View", l'uso dei *device* Live standard, l'uso della funzione *Group* (*Raggruppa*) per creare *Audio Rack* contenenti *device* multipli, l'uso delle diverse catene (*chain*) di *device* all'interno dell'*Audio Rack*, l'uso delle automazioni e l'uso degli inviluppi nelle *clip*.

Che cos'è Max for Live

Max for Live è un'estensione di Live che permette la creazione, in linguaggio Max, di nuovi *plug-in* (detti *device*, in italiano dispositivi, nel gergo Live)[2] utilizzabili all'interno dell'applicazione.
Tramite Max for Live è inoltre possibile controllare diverse funzioni di Live, come ad esempio gestire il volume o il pan di una traccia, far partire o arrestare una *clip* o modificare i parametri di altri *device*.
Per utilizzare Max for Live (d'ora in poi M4L)[3] non è necessario possedere una licenza completa di Max, è sufficiente avere la sola licenza d'uso di M4L, oltre a quella di Live.

Chi ha M4L ma non Max può comunque utilizzare le *patch* contenute in questo

[1] A partire dalla versione 9 di Live, Max for Live è incluso nei pacchetti disponibili con Live Suite.

[2] Abbiamo accennato ai *plug-in* nel paragrafo 3.8T del primo volume. Si tratta di componenti software che vengono "ospitate" all'interno di un programma e ne ampliano le funzionalità. Tipici esempi di *plug-in* audio sono ad esempio compressori, delay, equalizzatori, o strumenti virtuali come sintetizzatori e campionatori.

[3] M4L è comunemente usato come abbreviazione di "Max for Live": *four* (4) in inglese ha un suono simile a *for*.

testo e nel precedente volume. Tenete però presente che in questo caso Max non è in grado di gestire autonomamente l'input e l'output audio e MIDI; questo significa che tutti i segnali audio e i messaggi MIDI devono passare attraverso Live.

IE.2 FONDAMENTI - CREARE UN AUDIO EFFECT CON M4L

Innanzitutto vi raccomandiamo di leggere tutti i paragrafi in ordine! In altre parole, anche se volete realizzare un *Instrument*, e non un *Audio Effect* con M4L, non saltate questo paragrafo perché contiene informazioni essenziali per la comprensione dei paragrafi successivi.
Prima di continuare assicuratevi di aver installato nel vostro computer Max for Live e tutti i pacchetti M4L disponibili nel vostro account presso il sito www.ableton.com.
Nella prima colonna del *Live Device Browser* (ovvero l'area che si trova alla sinistra delle tracce, nella finestra principale di Live), selezionate la categoria *Max for Live*; nella seconda colonna del *Browser* appariranno 3 cartelle: *Max Audio Effect*, *Max Instrument* e *Max MIDI Effect*. Queste tre cartelle raccolgono rispettivamente gli effetti audio, gli strumenti virtuali e gli effetti MIDI realizzati con M4L.
Aprite la cartella *Max Audio Effect*, localizzate il *device* "Max Chorus" (sono elencati in ordine alfabetico) e trascinatelo su una traccia audio (fig.IE.1).

fig. IE.1: *device* "Max Chorus"

Il *device*, visibile nella parte bassa della figura, è stato scritto in Max, e per vedere la *patch* relativa è necessario fare clic sulla prima delle tre piccole icone circolari che si trovano in alto a destra nella barra del titolo del *device* (fig. IE.2).

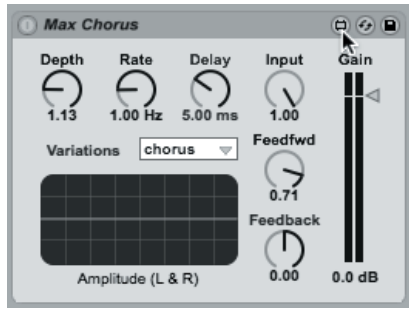


fig. IE.2: aprire la *patch* di un *device* M4L

Verrà lanciata l'applicazione Max e sarà possibile vedere e modificare la *patch*. Inizialmente la *patch* si apre in modalità *presentation*: facendo clic sull'icona che attiva e disattiva la modalità (la piccola lavagna nella parte bassa della *patcher window*) è possibile, come sappiamo, passare in modalità *patching*.
La *patch* relativa al *device* "Max Chorus" è visibile in figura IE.3; dal momento che è molto semplice non la descriveremo.

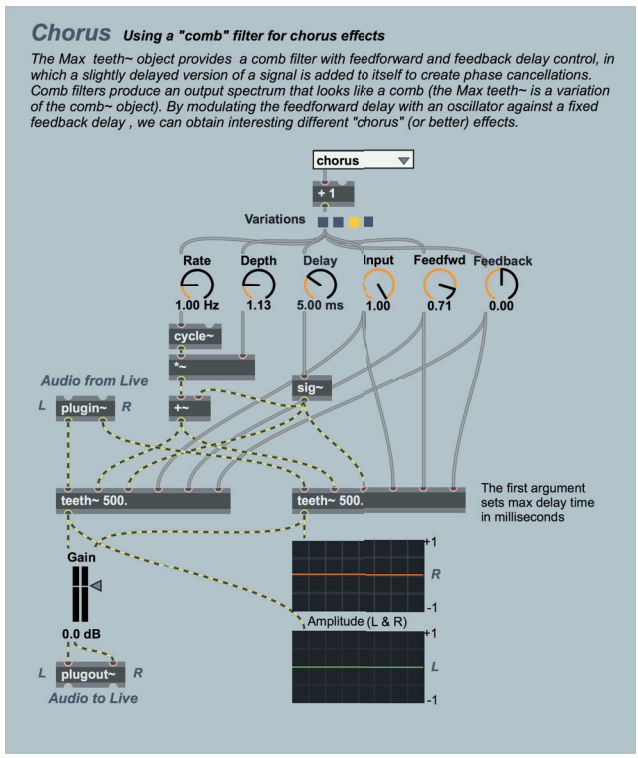


fig. IE.3: la *patch* del *device* "Max Chorus"

Notate solo la coppia di oggetti **`teeth~`** nella metà bassa della *patch*: si tratta semplicemente di filtri *comb* in cui (a differenza dell'oggetto `comb~`) il *delay* di *feedforward* è regolabile indipendentemente dal *delay* di *feedback*.
Osservate inoltre il titolo della *patcher window*: "Max Chorus.amxd". Il suffisso di una *patch* che viene usata come *device* Live è quindi *amxd*, e non *maxpat*.

Aggiungete un'audio clip alla traccia e provate il *device*. Provate poi altri *device* di effetti audio M4L, aprendo le *patch* relative per vederne il funzionamento. Buon divertimento!

Vediamo ora come si crea un nuovo *device* in M4L. Notate innanzitutto che il primo *device* elencato all'interno della cartella *Max Audio Effect* si chiama semplicemente "Max Audio Effect" e ha un'icona diversa da quella degli altri *device*. Si tratta in effetti di un *template* a partire dal quale possiamo realizzare i nostri *device*: trascinatelo su una traccia audio, o alternativamente fate doppio clic sulla sua icona; apparirà il *device* di *default* sulla traccia selezionata (fig. IE.4).

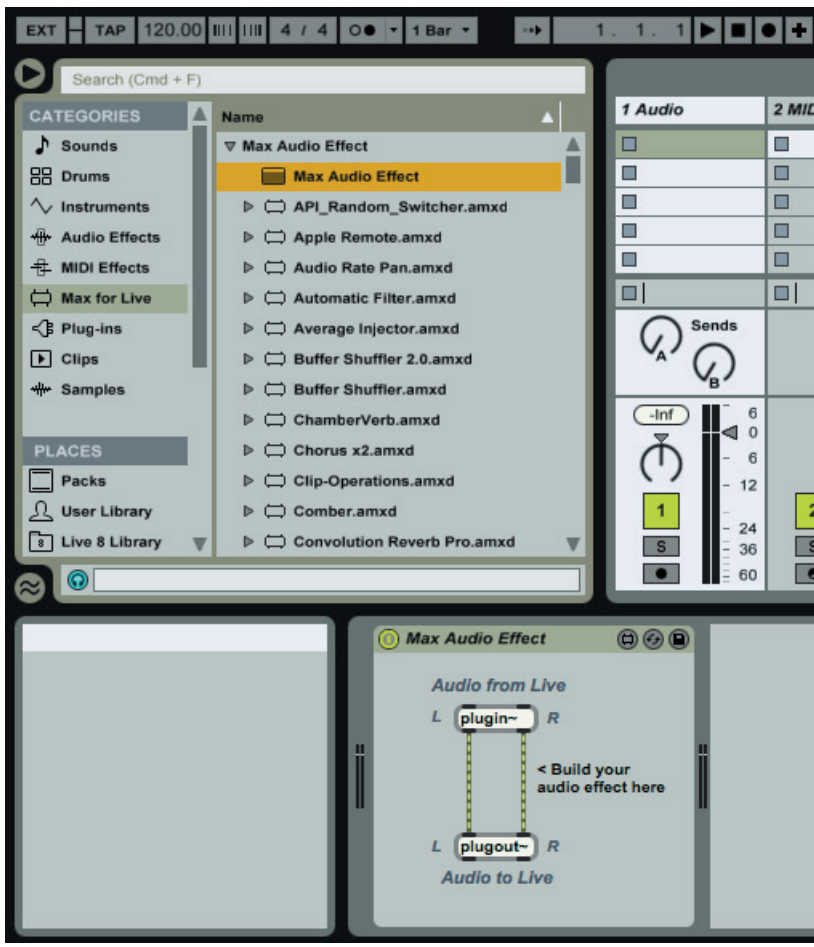


fig. IE.4: il *device* di *default* per gli effetti audio

Aprendo questo *device* è possibile modificarlo e salvarlo con un nuovo nome. M4L vi proporrà di salvarlo all'interno della cartella *Max Audio Effect* che si trova nella vostra "User Library", vi consigliamo di non cambiare questo percorso, e di creare eventualmente una sotto-cartella in cui raccogliere tutti i vostri *device*.

Il *device* di *default* contiene due soli oggetti: **`plugin~`** e **`plugout~`**. Il primo riceve il segnale audio dalla traccia (o dall'eventuale *device* precedente) il secondo trasmette l'audio all'uscita della traccia (o all'eventuale *device* successivo): in pratica questi due oggetti sostituiscono gli oggetti **`adc~`** e **`dac~`** delle normali *patch* Max.
Proviamo a costruire uno *slapback delay* (vedi cap. 6.2P): modificate il *device* come da figura IE.5, e salvate la *patch* con il nome "My Slapback Delay.amxd".

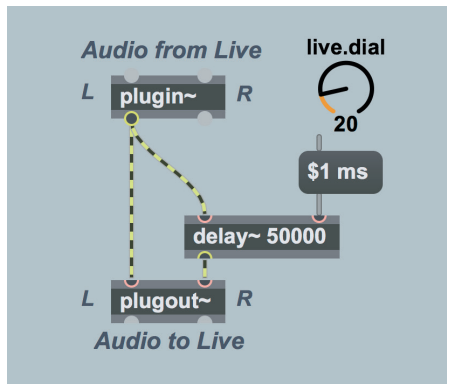


fig. IE.5: il nostro primo *device*, uno *slapback delay*

Notate che in *edit mode* appare una linea orizzontale indicata come "Device vertical limit". Solo gli oggetti che si trovano al di sopra di tale linea saranno visibili nel *device* caricato in Live.
Come si può vedere, il nostro *device* non è in modalità *presentation*: esattamente come per le normali *patch* Max è possibile aprire un *device* automaticamente in modalità *presentation* attivando l'attributo "Open in Presentation" nel *patcher inspector* (vedi anche Interludio D, paragrafo ID.2). Naturalmente è anche necessario includere nella modalità *presentation* tutti gli oggetti che desideriamo utilizzare.
L'oggetto circolare visibile in alto a destra è un **`live.dial`**, la "versione Live" dell'oggetto **`dial`** che già conosciamo[4]. Questo oggetto di *default* gestisce valori numerici compresi tra 0 e 127; vedremo tra breve altre caratteristiche degli oggetti di tipo "live.*".[5] Caricate una *clip* audio nella traccia e provate il *device*.

[4] Potete richiamare il **`live.dial`**, e tutti gli altri oggetti Max for Live, facendo clic sulla palette corrispondente all'ottava icona della *Toolbar* Superiore. Se non lo trovate potete comunque richiamarlo digitando "l" ("L" minuscolo) in una *patch* in modalità *edit*: apparirà un *object box* generico con un menù di *text completion* contenente i nomi di tutti gli oggetti della categoria Live.
[5] Con il simbolo "live.*" intendiamo qualsiasi oggetto Max della categoria Live e il cui nome comincia per "live."

I PARAMETRI DEGLI OGGETTI "LIVE.*"

Soffermiamoci un momento sull'oggetto `live.dial`, che presenta importanti caratteristiche comuni a tutti gli oggetti di tipo "live.*".
Come abbiamo detto l'oggetto trasmette di *default* valori tra 0 e 127. Il formato e l'intervallo dei valori può essere naturalmente modificato: se aprite l'inspector dell'oggetto noterete che presenta una serie di attributi, sotto la categoria "Parameter", che sono caratteristici degli oggetti di tipo "live.*".
Individuate nell'inspector gli attributi "Type", "Range/Enum", "Unit Style" e "Steps" (fig. IE.6).

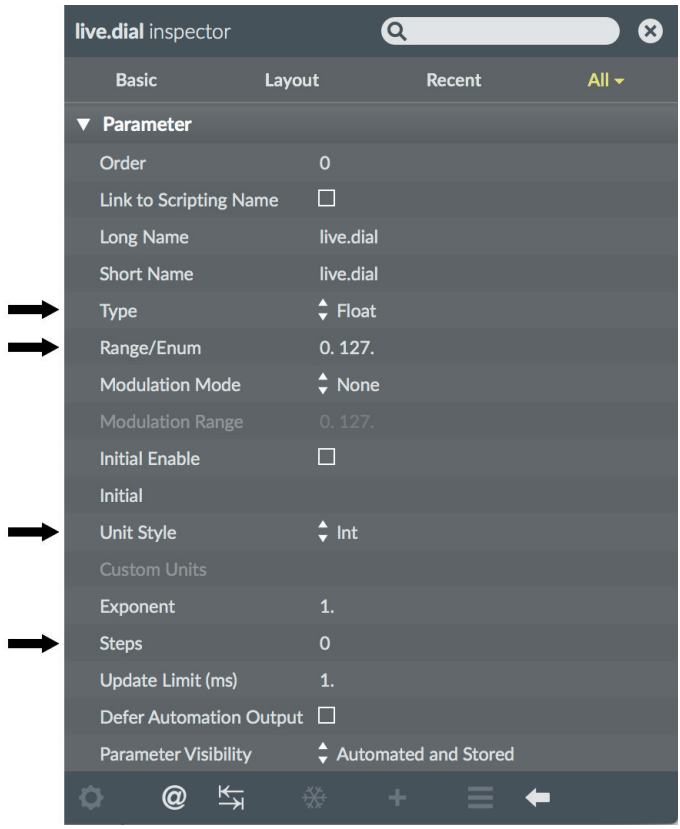


fig. IE.6: l'*inspector* di `live.dial`

(...)

il capitolo prosegue con:

Creare un device in modalità presentation
I nomi degli oggetti "live.*" e la parameters window
Automazione dei parametri in M4L
Modulazione dei parametri in M4L
Sincronizzare un device con il live transport
Usare oggetti interfaccia Max in M4L
Send-receive in un device M4L
Annotation e hint
Freeze device
Preset e pattrstorage in M4L

IE.3 VIRTUAL INSTRUMENT CON M4L

IE.4 MAX MIDI EFFECT

IE.5 LIVE API E LIVE OBJECT MODEL (LOM)
Definire un percorso (path)
Impostare ed osservare le variabili (proprietà)
Modulare i parametri con live.remote~
Una mappa per le live API: il Live Object Model (LOM)
Come individuare la traccia corrente
Le funzioni di una classe
Indirizzare un elemento con il mouse
Ottenere informazioni tramite live.object e live.observer
Abstraction M4L e altre risorse
Controllo automatico di tutti i parametri di un device

- LISTA OGGETTI MAX - LISTA ATTRIBUTI, ARGOMENTI E AZIONI PER OGGETTI MAX SPECIFICI - GLOSSARIO

Alessandro Cipriani • Maurizio Giri

Musica Elettronica e Sound Design

Teoria e Pratica con Max 7 • volume 2

Argomenti trattati

Audio digitale e suoni campionati: decimazione, tecnica dei blocchi, slicing, scrubbing, tempo e polifonia - Linee di Ritardo: eco, loop, flanger, chorus, filtri comb e allpass, phaser, pitch shifting, reverse, delay variabili, algoritmo di Karplus-Strong - Usi tecnici e creativi dei processori di dinamica: envelope follower, compressori, limiter, live normalizer, espansori, gate, side-chain, ducking - L'arte dell'organizzazione del suono: processi di movimento semplici, complessi e composti, movimento all'interno del timbro, gestione algoritmica dei movimenti, sequenze di movimenti - MIDI e gestione dei messaggi MIDI in Max - Max for Live: audio effects, virtual instruments, MIDI effects, Live API e Live Object Model.

"Il libro di Alessandro Cipriani e Maurizio Giri costituisce uno dei primi corsi di musica elettronica che integra esplicitamente percezione, teoria e pratica, usando esempi di sintesi in tempo reale che si possono manipolare e personalizzare. Secondo il mio parere, Cipriani e Giri hanno compiuto un lavoro magistrale consentendo all'esperienza e alla conoscenza teorica di rinforzarsi l'una con l'altra. Questo libro funziona sia come testo all'interno di un corso, sia come mezzo per l'autoapprendimento. In aggiunta, il libro include un'introduzione completa all'elaborazione digitale dei segnali con Max e costituisce una splendida introduzione ai concetti di programmazione con questo software. Spero che trarrete vantaggio dagli eccellenti esempi di Max creati dagli autori: sono insieme divertenti e illuminanti, e suonano abbastanza bene da poter essere utilizzati anche sul palco. Vale la pena esaminarli come modelli o per estenderli in modi sempre nuovi." (dalla prefazione di **David Zicarelli**)

Questo è il secondo volume di un sistema didattico organico in tre volumi sulla sintesi e l'elaborazione digitale del suono comprendente una corposa sezione online composta da centinaia di esempi sonori e interattivi, programmi scritti in Max e una libreria di oggetti Max espressamente creata per questo volume.

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 e insegna tecniche di programmazione con Max nei Conservatori di Latina 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 978-88-99212-06-3

www.contemponet.com

€ 47.00 iva inclusa