

## Sintesi Digitale: Max/MSP

Max/MSP è un ambiente integrato di programmazione per la musica orientato agli oggetti grafici. A differenza di altri linguaggi di programmazione basati sul paradigma dell'Object Oriented Programming (OOP) come SuperCollider e C-Sound, basati sulla scrittura testuale dei programmi, Max/MSP si basa su oggetti grafici. La 'scrittura' di un programma in, detto **patch** in Max/MSP, consiste nella interconnessione dei vari oggetti (**objects**) attraverso dei cavi virtuali (**patchcords**).

Max/MSP può a tutti gli effetti essere considerato un linguaggio orientato agli oggetti in quanto, come i software ai quali si è accennato sopra, è basato sulla regola "un'interfaccia, molti metodi": in pratica l'utente lavora con un numero di moduli funzionali, gli oggetti appunto, concettualmente identici. La differenza fra gli oggetti consiste in quello che fanno (funzione) e nel modo in cui il programmatore 'forza' a farlo (metodi e messaggi). La sintassi risulta così estremamente semplificata e l'ambiente di sviluppo estremamente aperto e flessibile. Max/MSP parte con una libreria di oggetti estremamente vasta, mettendo in grado l'utente di implementare praticamente qualunque algoritmo. Gli oggetti disponibili sono divisi in due categorie: quelli dedicati alla composizione assistita, alla parte di controllo e al MIDI, che costituiscono la parte MAX del software, e quelli dedicati alla generazione ed elaborazione di audio digitale, costituenti la parte MSP (Music Signal Processing) dell'ambiente. In aggiunta è stata sviluppata una terza categoria di oggetti, chiamata JITTER, nata circa due anni fa, che integrano in Max/MSP l'elaborazione e la generazione di segnali Video. I tre tipi di oggetti possono essere utilizzati contemporaneamente (e, almeno per i primi due tipi, devono essere usati contemporaneamente, come vedremo) in una patch, rendendo possibili complessi algoritmi di elaborazione video in *real-time*. Essendo Max/MSP scritto nel linguaggio di programmazione C++, la versione ad oggetti del famoso C, è stato possibile per la cycling74 rendere disponibile (il download è gratuito) un Software Development Kit (SDK) che permette di sviluppare oggetti, in questo caso chiamati **externals**. Questo ha permesso il fiorire di librerie di oggetti create da programmatori di terze parti, o addirittura indipendenti, che hanno ampliato notevolmente il range di possibilità che il software offre. Tanto per nominarne alcune fra le più famose, si possono citare le librerie *Percolate*, una raccolta di oggetti per il DSP e per la sintesi video, e *Litter*, una collezione di oggetti MAX, per la generazione di sequenze casuali di numeri.

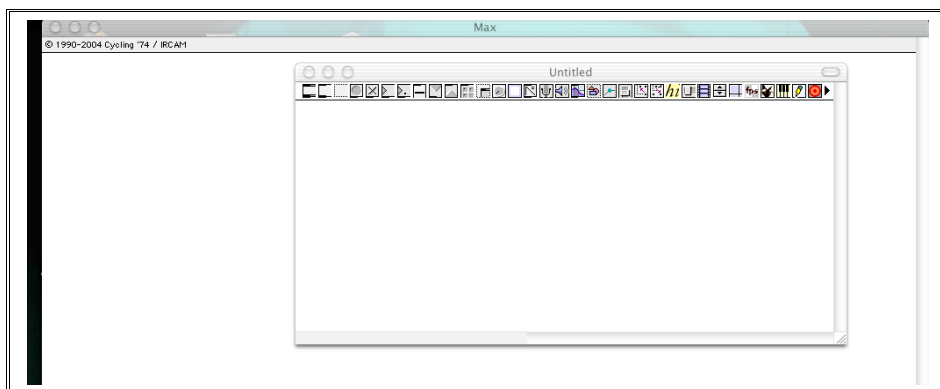
Per favorire la diffusione del software e lo scambio di patches fra gli utenti, Max/MSP viene fornito di due versioni eseguibili: la versione del software full e la versione detta **runtime**, che è disponibile per il download gratuito sul sito della cycling74. Tale versione gira indipendentemente dal software completo, ed ha due limitazioni: a) non è possibile creare patches nuove al suo interno b) non è possibile editare patches aperte al suo interno, ma ha la funzione di poter utilizzare patches create da altri utenti, senza possedere il software. Pertanto se un musicista è interessato solamente all'utilizzo di una patch creata da qualcun altro, può farlo senza acquistare il software. Questa caratteristica è comune a quasi tutti i linguaggi di programmazione dedicati alla musica di ricerca, come SuperCollider ad esempio.

Un'altra caratteristica di Max/MSP è la possibilità di creare plug-ins e applicazioni stand-alone a partire da una patch. I plug-ins che si possono creare sono compatibili con tutti i formati esistenti, ossia VST, RTAS, MAS e , nella versione per osX, anche AU (Audio Unit, il formato proprietario Apple).

La possibilità di disporre di una versione runtime del software, di sviluppare externals e di poter creare applicazioni stand-alone e plug-ins ha fatto di Max/MSP il software più utilizzato nella musica di ricerca e nell'elettronica, e l'ambiente di riferimento per la creazione di installazioni e opere intermediali.

## L'interfaccia Utente di Max/MSP

All'apertura del software, Max/MSP si presenta come in Fig.1.

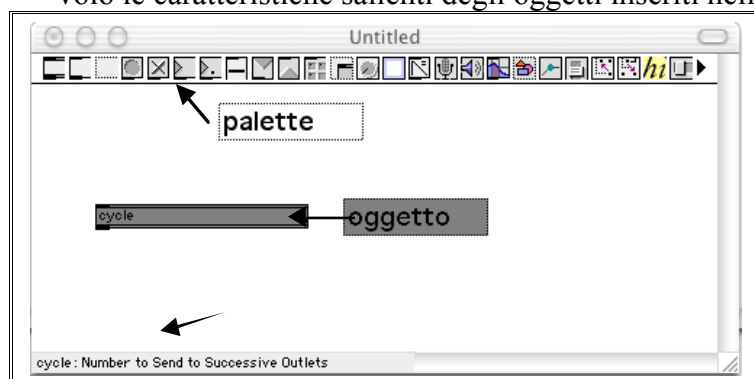


**Figura 1: Schermata di Max/MSP all'apertura del software.**

La finestra chiamata Max è una pagina che sta in back-ground, nella quale vengono visualizzati messaggi di caricamento del software, di configurazione delle interfacce audio collegate al sistema, messaggi di errore o output testuali del software. Può essere chiusa e riaperta a piacimento (Mela-M) sul sistema osX (su os9.2 deve essere mantenuta aperta) ed è utilissima per il debug delle patch.

La finestra che in fig.1 è chiamata 'untitled' (perché non ancora nominata!) è detta **patcher-window**, ed è la finestra di lavoro principale, quella in cui viene di fatto creata la patch, come dice il nome stesso. La patcher window è costituita di 3 parti fondamentali, indicate in fig.2 :

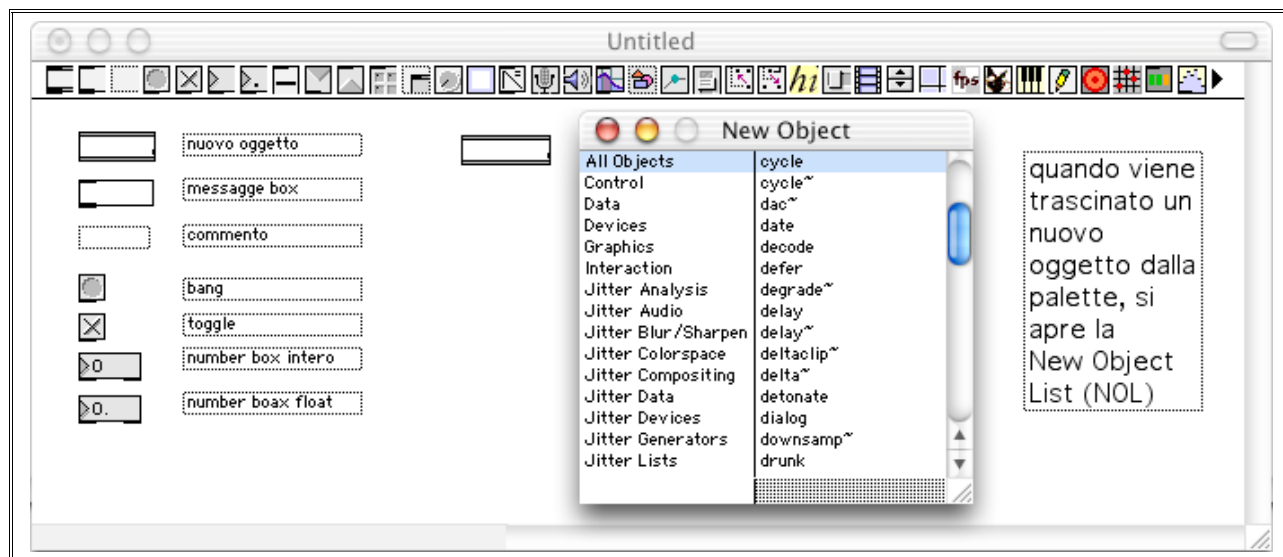
1. La **palette**: un menù grafico ad icone, che contiene tutti gli oggetti disponibili. È possibile scegliere un oggetto e inserirlo nella patch, trascinandolo con il mouse dalla palette all'area di lavoro.
2. L'**area di lavoro**: lo spazio bianco della patcher window è dedicato alla costruzione grafica della patch. Conterrà gli oggetti, collegati con i **patchcords**, e altre **sub-patch**. Vedremo infatti che una delle caratteristiche più importanti di Max/MSP è la possibilità di incapsulare patch dentro patch, organizzando concettualmente e graficamente nel migliore dei modi un algoritmo.
3. L'**assistance area**: in basso a sinistra, nella patcher window, al bordo della finestra, vi è una striscia che costituisce una sorta di utilissimo quick help testuale, che permette di conoscere al volo le caratteristiche salienti degli oggetti inseriti nella patch.



**Figura 2: Patcher Window.**

Gli **oggetti** sono costituiti da icone di diverso tipo, secondo la funzione che svolgono. I principali tipi di oggetti sono elencati in fig.3. Tutti gli oggetti, sia quelli Max che quelli Msp, sono dotati di ingressi, detti **inlets**, e di uscite, dette **outlets**, rappresentati come dei rettangolini neri nella parte superiore (inlets) e inferiore (outlets) dell'oggetto. Per conoscere quale è la funzione di un ingresso o di un uscita e che tipo di dati accetta, basta puntarlo con il mouse e leggere le informazioni

relative scritte nell'assistance area. Informazioni molto più dettagliate relative agli oggetti sono date nel *Reference Manual*.



**Figura 3: Tipi di oggetto e New Object List.**

Gli oggetti vengono interconnessi con dei cavi virtuali, i **patchcords**, che collegano gli outlets di un dato oggetto agli inlets di un altro, secondo l'algoritmo che si desidera implementare. Attraverso i patchcords gli outlets mandano informazioni agli inlets, e tali informazioni sono detti **messaggi**. Discuteremo adesso i principali tipi di oggetti MAX, mentre gli oggetti MSP saranno trattati più avanti. A parte differenze concettuali e funzionali, i messaggi MSP vengono trattati, nella costruzione di una patch, allo stesso modo degli oggetti MAX.

### **Gli Oggetti MAX**

In fig.3 sono elencati i principali tipi di oggetto MAX, ossia gli oggetti dedicati ai segnali di controllo e al MIDI. L'oggetto principale è l'**object-box**, ossia il primo da sinistra sulla palette. Se si trascina l'object-box nell'area di lavoro, viene aperta la **New Object List (NOL)**, ossia la lista di tutti gli oggetti disponibili. È possibile scegliere gli oggetti per categoria, selezionandola nella parte sinistra della NOL, selezionando l'oggetto di nostro interesse nella parte destra, che può essere fatta scorrere velocemente digitando le prime due lettere dell'oggetto. È anche possibile scrivere direttamente il nome dell'oggetto nel box: chiudendo la NOL apparirà un cursore. Il nome dell'oggetto, che indica la sua funzione, può essere costituito da un nome, come ad esempio *metro*, *makenote*, *cycle*, o da un simbolo, come > o !=.

Gli inlets e gli outlets hanno funzioni differenti a seconda dell'oggetto, un'indicazione delle quali è data sinteticamente nell'assistance area, come già detto. Alcuni oggetti possono non avere inlets o outlets, perché ricevono ingressi o mandano uscite verso interfacce esterne al software. Ad esempio *midout* non ha outlets, in quanto manda messaggi all'interfaccia midi hardware collegata al computer.

Il secondo oggetto indicato in fig.3 è detto **message-box**, ed è destinato a contenere messaggi da inviare ai vari oggetti. I messaggi costituiscono i dati sui quali operano gli oggetti di MAX.

Max/MSP è un linguaggio *tipato*, ossia i dati che elabora sono suddivisi in *tipi*, che ne caratterizzano l'entità, e oggetti diversi possono accettare alcuni tipi di dato soltanto.

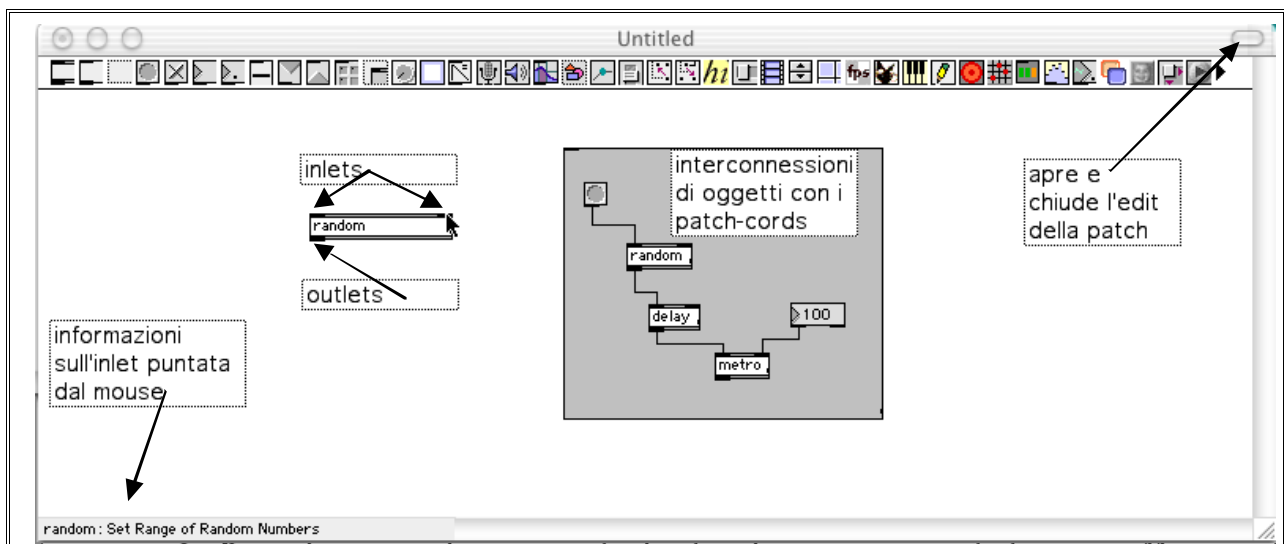
I *tipi di dato* in Max/MSP sono:

- **Int**: numeri interi (ad esempio: 5).
- **Float**: numeri in floating point, ossia con virgola decimale (ad esempio: 3.256).
- **List**: lista di due o più numeri, separati da uno spazio. Max riconosce come lista ogni sequenza di numeri separati da uno spazio.

- **Bang**: il messaggio bang è un particolare tipo di messaggio, importantissimo per Max: quando inviato ad un oggetto lo forza a fare quello che è preposto a fare! Ad esempio se è ricevuto dall'oggetto **random**, esso invierà al suo outlet un numero casuale.
- **Symbol**: un symbol è una sequenza di caratteri non numerici. Un messaggio simbolo serve principalmente a inviare comandi particolari ad oggetti. Ad esempio l'oggetto **groove** se riceve nel suo inlet di sinistra un messaggio *'loop 1'*, legge in loop l'audio file al quale è collegato; se invece riceve il messaggio *'loop 0'* esce dal loop.
- **Anymessage**: un messaggio può essere costituito di una serie di numeri e parole. Alcuni oggetti possono ricevere messaggi di qualunque tipo.

Si noti in fig.4 il tasto in alto a destra nella patcher window: premendolo è possibile aprire o chiudere in **edit mode** la patch, in modo da rendere possibile o meno l'editing. La stessa cosa può essere fatta attraverso lo short-cut Mela-E o con Mela-click in un punto dell'area di lavoro.

Il terzo oggetto in fig.3 è il **comment-box**, ossia un riquadro nel quale è possibile inserire testo per commentare la patch. Il commento è buona norma in ogni linguaggio di programmazione, ed ha il duplice scopo di rendere comprensibile la patch ad utenti diversi dal programmatore e quello di rendere possibili il debug ed ulteriori modifiche alla patch, anche a distanza di tempo, da parte del programmatore stesso. Può sembrare strano, ma aprire una patch che non si usa da mesi e capirci qualcosa può essere impresa ardua, anche se siamo stati noi a crearla.....



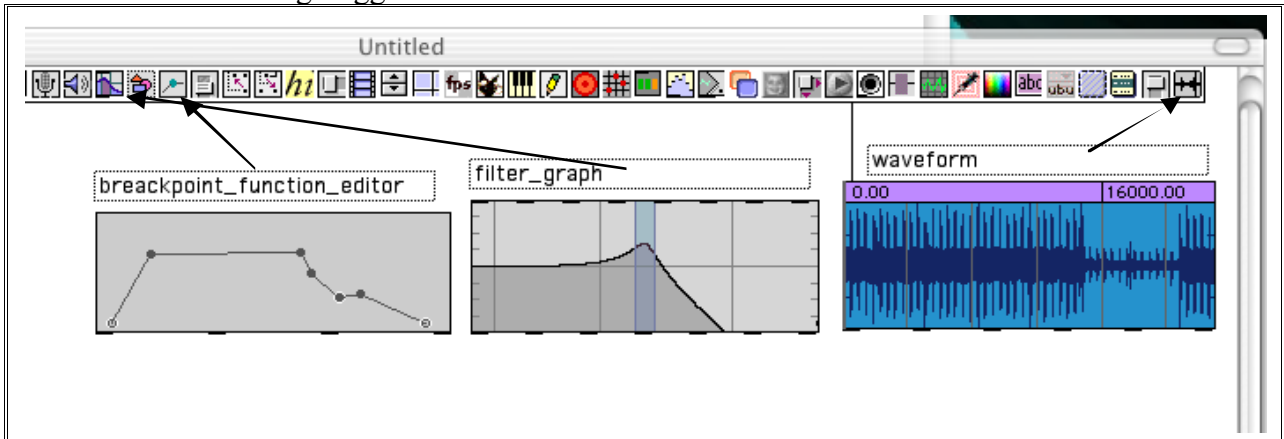
**Figura 4: Interconnessioni, inlets e outlets.**

Il quarto oggetto è il **bang**, fondamentale nella costruzione di ogni patch in Max/MSP. Tale oggetto manda un messaggio di tipo bang, come descritto sopra, e fa parte, assieme agli altri tre tipi di oggetto che lo seguono in figura, di un insieme di oggetti dedicati all'interfaccia utente della patch. Infatti, la possibilità di comunicare con la macchina, ossia di immettere dati che il nostro algoritmo deve elaborare, è offerta proprio da tale tipologia di oggetti. I **number-box**, sia di tipo **int** che **float**, sono dedicati all'inserimento di dati di tipo numerico da parte dell'utente, mentre l'oggetto **toggle** è usato per inviare messaggi di tipo on/off. Vedremo più avanti delle applicazioni di questi oggetti, e avremo modo di capirne l'importanza. I number-box sono molto importanti anche per fare il debug di una patch, in quanto permettono di visualizzare il risultato ottenuto dall'outlet di molti oggetti.

Un'ulteriore categoria di oggetti dedicati all'interfaccia utente è costituita dagli **user interface objects**, oggetti di vario tipo, presenti nella palette e trascinabili nell'area di lavoro, che servono a rappresentare e manipolare graficamente dati e grandezze in uso nella patch. Ad esempio l'oggetto **waveform**, vedi fig.5, serve a rappresentare la forma d'onda di un file audio contenuto in un oggetto **buffer** (vedi esempio sul playback ed il recording di audio file), ma serve anche a

modificare vari parametri relativi alla sua lettura da parte dell'oggetto **groove**, con il quale comunica, come inizio, fine, lunghezza del loop etc.

Sempre in fig.5 è mostrato l'oggetto **breackpoint function editor**, utilizzato per la creazione e l'editing di involuipi. Altri utilissimi user interface objects sono il **filter graph**, che permette di editare i parametri e la risposta di un filtro, e lo **scope**, un oscilloscopio che può diventare correlatore di fase e analizzatore di spettro. Negli esempi applicativi che faremo più avanti, utilizzeremo tutti e tre gli oggetti citati.



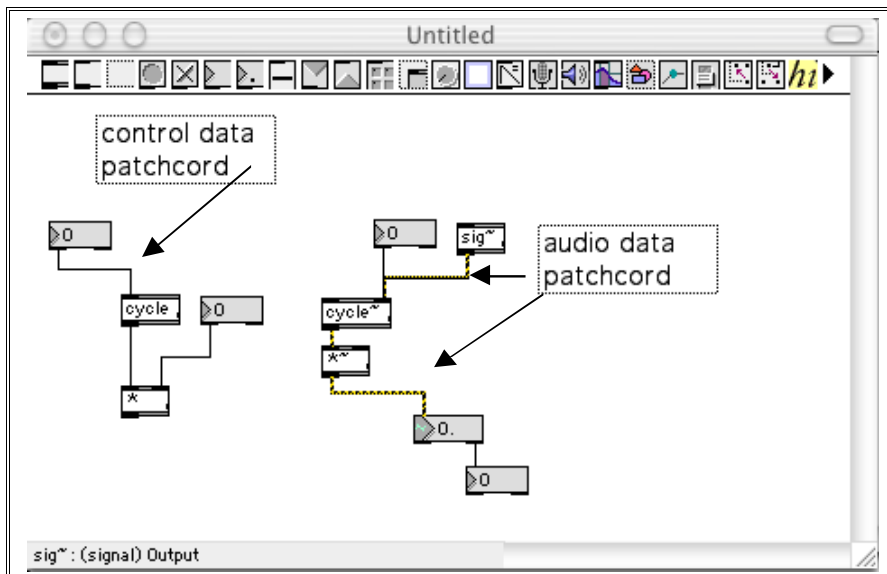
**Figura 5:Alcuni User Interface Objects.**

### ***Gli oggetti MSP***

Nell'introduzione abbiamo raccontato brevemente la storia dello sviluppo e dell'evoluzione di Max/MSP. Abbiamo visto che la parte MSP del software, dedicata al Digital Signal Processing in real-time, è stata sviluppata in un secondo momento. Gli oggetti MSP sono del tutto simili a quelli finora trattati per quanto riguarda il loro inserimento nella patch, basta infatti trascinare l'oggetto box nell'area di lavoro e scegliere l'oggetto dalla NOL oppure scriverne il nome nel box.

Vi sono però alcune importanti differenze, intrinseche alla natura degli oggetti:

1. **Nome:** per poter essere distinti dagli oggetti MAX, tutti gli oggetti MSP hanno il nome formato da una parola seguita dal simbolo ~, detto tilde. Per ottenerlo si deve usare la combinazione di tasti **alt+5**. Molti oggetti hanno due versioni, una MAX ed una MSP. Ad esempio l'oggetto **cycle**, che genera una sinusoide, ha sia la versione **cycle** che quella **cycle~**.
2. **Patchcords:** i cavi virtuali che collegano inlets e outlets di oggetti MSP sono graficamente diversi da quelli che collegano oggetti MAX, infatti i segnali che attraversano i due tipi di patchcords sono differenti. Nel secondo caso si tratta di messaggi relativi al controllo ed al MIDI, nel secondo si tratta di dati audio. In fig. 6 si può osservare la differenza dal punto di vista grafico: mentre i patchcords per gli oggetti MAX sono neri, quelli che collegano oggetti MSP sono costituiti da un filo nero ed uno giallo (il colore può essere cambiato) intrecciati.
3. Il percorso di segnale che la rete di connessioni fra oggetti di tipo MSP crea, viene vista da Max/MSP come una sorta di una operazione matematica, valutata *in ogni istante* dal processore (le operazioni che il processore deve compiere dipendono dagli oggetti interconnessi e, naturalmente dal modo in cui sono collegati). Questa precisazione è fondamentale perché costituisce la differenza principale fra le due tipologie di oggetto. Gli oggetti MAX, infatti, nel loro collegamento, creano un semplice percorso per le informazioni, che sono processate dagli oggetti solamente quando tali oggetti sono 'stimolati' dall'utente o da altri oggetti. Inoltre, mentre la velocità alla quale gli oggetti MSP processano il segnale è pari alla frequenza di campionamento (scelta nella pagina *DSP Status* come frequenza operativa del software), definita **audio rate**, gli oggetti MAX controllano gli inlets e generano dati agli outlets ad una frequenza di lavoro molto più bassa, detta **control rate**, pari al millesimo di secondo (comunque sufficiente per i dati MIDI e di controllo).



**Figura 6: Patchcords per oggetti MAX e MSP.**

Si deve notare una cosa importante: tutti gli oggetti MSP ricevono messaggi di controllo, ad esempio in fig.6 il number box collegato a cycle~ serve ad impostare un parametro; ma nessun oggetto MAX può ricevere dati audio. Per far comunicare i due mondi esistono degli oggetti dedicati, come **sig~** e **number~**, in fig.6. In pratica tali oggetti trasformano messaggi che viaggiano alla control rate in messaggi più veloci, alla audio rate, e viceversa.

### ***La regola del “right to left order” e lo Sviluppo Verticale***

Le patches si sviluppano in verticale, dall’alto verso il basso, con gli ingressi in alto e le uscite in basso, assecondando la disposizione di inlets e outlets. Il fatto che Max/MSP elabori i dati molto velocemente, crea l’illusione che gli eventi avvengano *contemporaneamente*. Ma niente può avvenire in contemporanea su una macchina, poiché essa dovrà per forza seguire un principio gerarchico nell’esecuzione delle istruzioni.

In Max/MSP questa limitazione, che risulta del tutto trasparente all’utente nella maggior parte dei casi, è stata incanalata nella regola del **right to left order**. In pratica, tutti i dati relativi agli oggetti MAX vengono valutati, all’interno della patch, da destra a sinistra. Questo significa che:

- Se un oggetto MAX ha più outlets, i risultati delle operazioni svolte dall’oggetto saranno mandati in uscita partendo dall’outlet di destra, proseguendo poi verso sinistra.
- La regola vale anche per la disposizione spaziale degli oggetti all’interno della patch: gli oggetti più a destra processeranno i dati prima di quelli a sinistra.

Questa regola, vale la pena sottolinearlo, è importantissima: non tenerne conto significa commettere errori grossolani, molto difficili da scovare. Facciamo un esempio in fig.7.

L’oggetto **note-in**, che riceve un messaggio di nota midi dall’interfaccia collegata al computer, manderà per primo il messaggio dall’outlet di destra, poi da quello centrale e per ultimo, da quello di sinistra.

Quando un oggetto ha più di un inlet, esso si aspetta il primo messaggio nell’inlet più a destra collegato ad un altro oggetto. Se due oggetti ricevono da uno stesso outlet, e sono allineati in verticale, l’oggetto più in basso sarà il primo a ricevere il segnale.

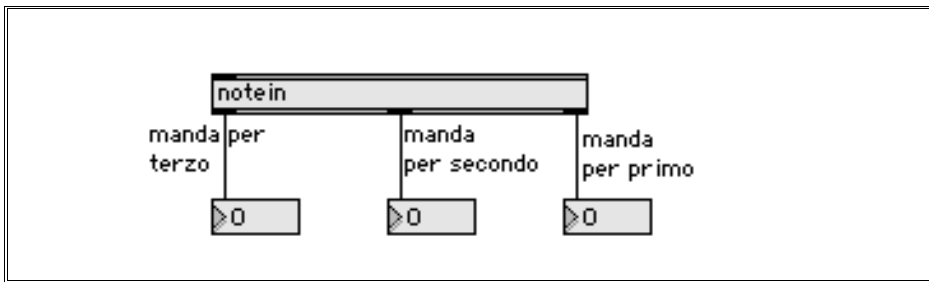


Figura 7: Regola del "right to left order".

### Audio I/O e DSP Status

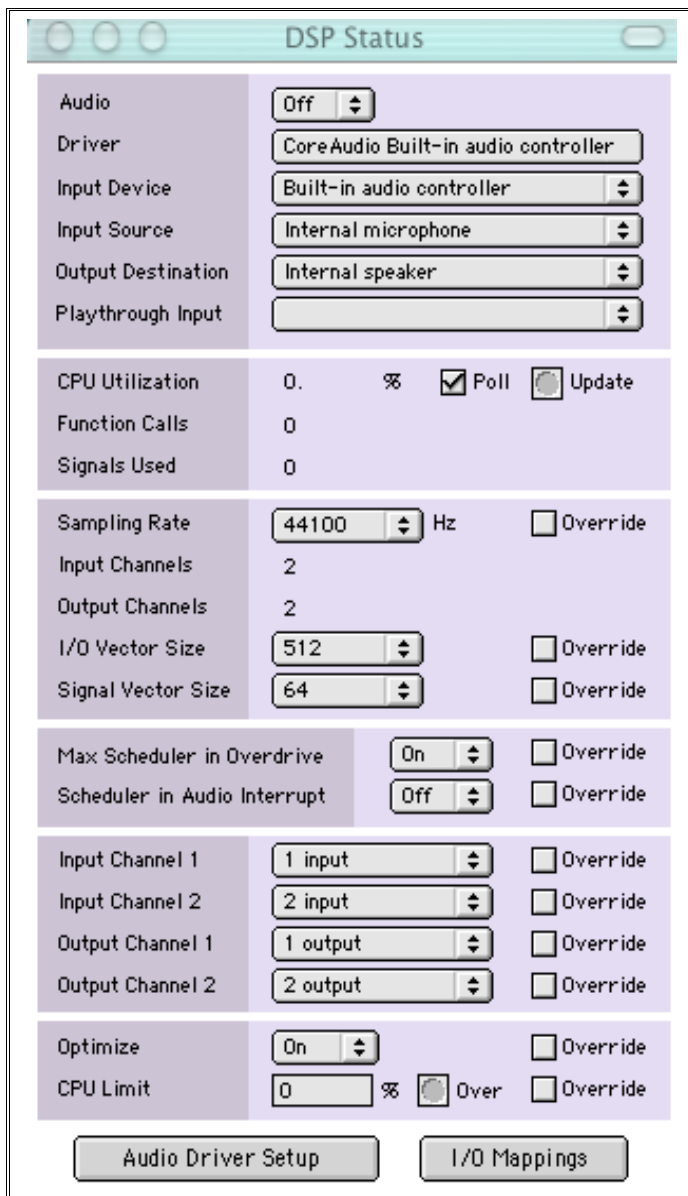
MSP si interfaccia con le schede audio installate sul computer attraverso degli oggetti dedicati: **adc~**, **dac~** (rispettivamente ingressi e uscite) per l'I/O multicanale e gli equivalenti (definiti 'facili') **ezdac~** e **ezadc~** per la gestione di ingresso e uscita stereo. In assenza di schede multicanale (su os9.2 viene supportato il protocollo **ASIO**, mentre su osX è utilizzato l'**Audio Core**) MSP utilizza di default il driver Sound Manager della Apple, che fornisce un I/O stereo full duplex.

La finestra **DSP Status** è dedicata alle configurazioni globali del software. Come si può osservare in fig. 8, nella parte superiore della finestra si sceglie la scheda audio da utilizzare, fra quelle installate sul sistema e dotate di driver 'visibili' dal software. Più in basso è possibile scegliere la frequenza di campionamento che Max/MSP utilizzerà nell'elaborazione dell'audio digitale (le scelte possibili sono imposte dall'hardware in uso).

L'**I/O Vector Size** è la taglia, in campioni, del buffer di ingresso e di uscita della scheda, e pertanto determinerà la latenza del sistema. Più è grande, meno carico graverà sul processore, maggiore sarà la latenza del sistema. Il **Signal Vector Size** è, invece, il numero di campioni elaborati in una volta dal software, durante il processing del segnale.

La finestra del DSP Status si può aprire in due modi: con doppio-click sull'oggetto **adc~** o **dac~**, oppure dal menù Option.

Max/MSP gestisce fino a 512 canali **logici** di ingresso e uscita, che vengono assegnati nel DSP Status ai canali fisici disponibili sulla scheda audio. Questo permette di poter usare MSP in applicazioni che utilizzano la tecnologia **ReWire**, sia come **client** che come **host**. L'assegnazione dei canali virtuali a quelli fisici viene fatta aprendo la finestra **I/O Mapping**, e, per i primi due canali, direttamente dal Dsp Status.



**Figura 8: DSP Status.**

### ***Help on-line e Documentazione***

Max/MSP è dotato di un potente sistema di help on-line, a vari livelli. Innanzitutto, come già visto, l'**assistance area**, situata in basso a sinistra nella patcher window, costituisce un mezzo veloce per avere informazioni su inlet e outlet di un oggetto.

L'**help on-line** vero e proprio si ottiene clickando sull'oggetto tenendo premuto **alt**: apparirà una vera e propria patch dimostrativa dell'oggetto, che può essere aperta ed editata a sua volta. Questo sistema consente di imparare bene ed in fretta, ed è spesso fonte di idee e spunti per la creazione di nuove patch.

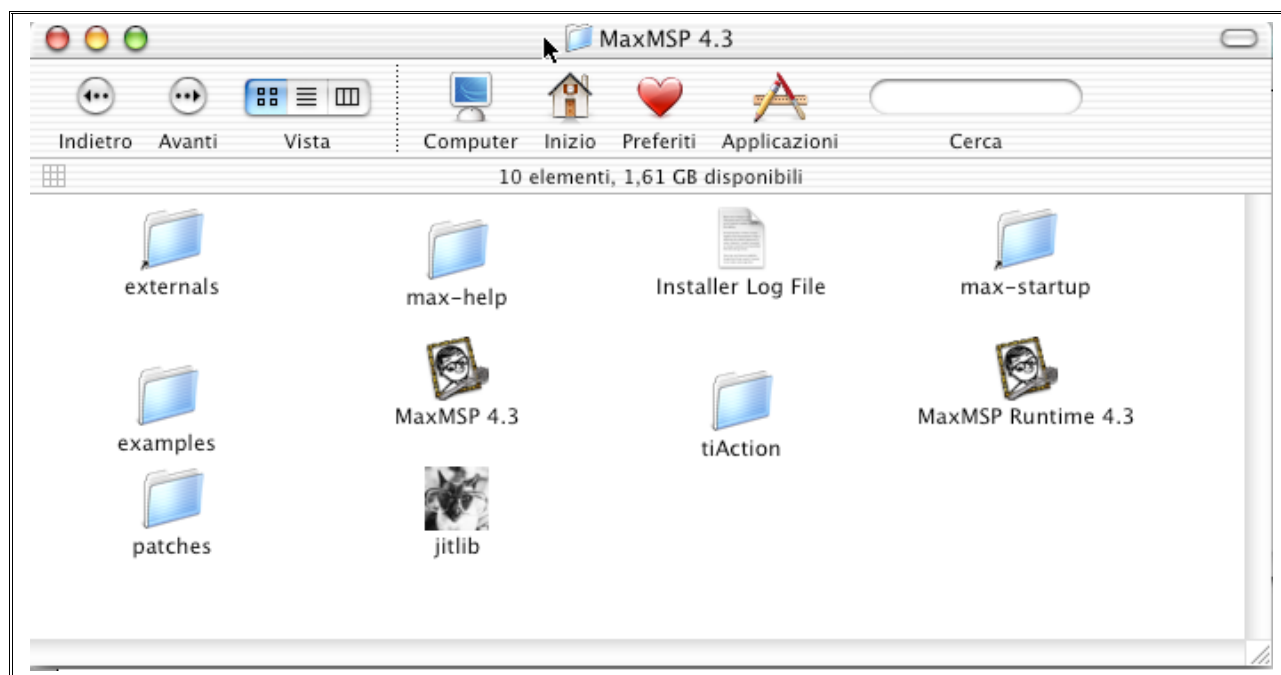
Naturalmente la documentazione in formato **.pdf** è insostituibile, data la completezza e la qualità delle spiegazioni, che vanno ben oltre l'uso del software, toccando e approfondendo argomenti quali vari tipi di sintesi, FFT, digital signal processing, MIDI, campionamento etc.etc.

Esistono due manuali, uno per MAX ed uno per MSP, corredati di tutorial sottoforma di patches, una per ogni capitolo del manuale. In aggiunta vi è un manuale **Getting Started**, che introduce all'audio digitale, al DSP e a varie altre interessanti questioni, e due **Reference Manual**, uno per MAX e uno per MSP, che raccolgono le descrizioni dettagliate di ogni oggetto.

Chi sviluppa oggetti **externals**, solitamente fornisce anche una **help patch**, da inserire nell'apposita **folder** dentro il folder principale di Max/MSP, in modo da poter aprire l'help on-line con le stesse modalità di un qualsiasi altro oggetto.

### **Il Folder di Max/MSP**

In fig. 9 si può osservare il folder di Max/MSP, e le sotto-cartelle in esso contenute. Naturalmente all'interno del folder troviamo i file eseguibili, che come abbiamo visto sono due **Max/MSP x.xx** e **Max/MSP Runtime x.xx** (le x indicano la versione del software, che, naturalmente può cambiare!).



**Figura 9: Il folder di Max/MSP.**

Le varie cartelle sono:

- **Max-help**: cartella che contiene le help-patch, utilizzate per l'help in linea, come visto. Per convenzione vengono nominate con estensione **.help**.
- **Externals**: cartella che contiene gli oggetti creati da terze parti, attraverso il Software Development Kit.
- **Max-startup**: contiene gli oggetti che saranno caricati all'avvio del software e messi nella palette, per poter essere trascinati nell'area di lavoro.
- **Patches**: contiene altre cartelle, che a loro volta contengono patches di utility nell'utilizzo del software, come gli **extras** (patches di utilizzo frequente, customizzabile dall'utente), **editors** (le pagine DSP Status, floating inspectors e altre sono normali patches, raccolte in questa cartella) e altre.

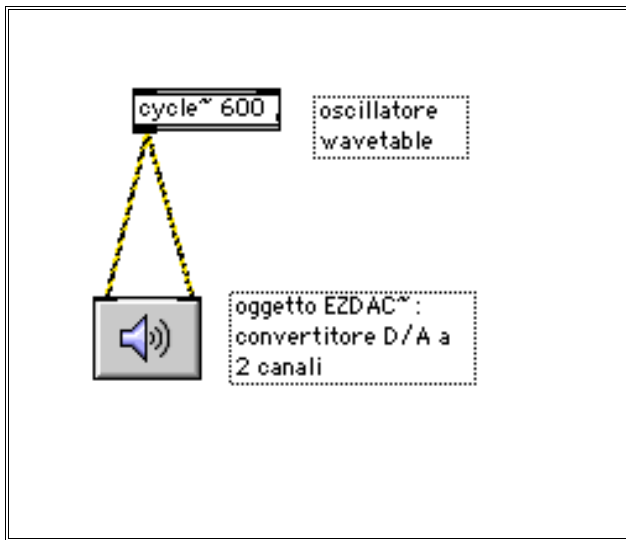
### **Esempi di Sintesi in Max/MSP**

Un modo abbastanza semplice e veloce di imparare ad usare Max/MSP è la realizzazione di piccole patches, mettendo in pratica le nozioni di sintesi analogica apprese nella prima parte del corso. La curva di apprendimento del software non è immediata, e questo è normale data la versatilità e le potenzialità enormi del software. Ma seguendo passo dopo passo i tutorials e i manuali forniti dalla stessa casa madre, si può imparare presto e bene.

### **La patch "hello world!" di Max/MSP: l'oscillatore sinusoidale.**

Ogni linguaggio di programmazione ha un programmino chiamato "hello world!", introduce all'ambiente di programmazione, senza ulteriori pretese. Per chi ha avuto un C64, si ricorda che in quel caso il programma in videoBasic più semplice consisteva nello scrivere sullo schermo "ciao, mondo!"....

Nel nostro caso creeremo una patch consistente in un oscillatore sinusoidale a frequenza fissa, molto semplice, ma che introduce già un paio di elementi importanti. Si osservi la figura che segue.



**Figura 10: "hello, world!" patch in Max/MSP.**

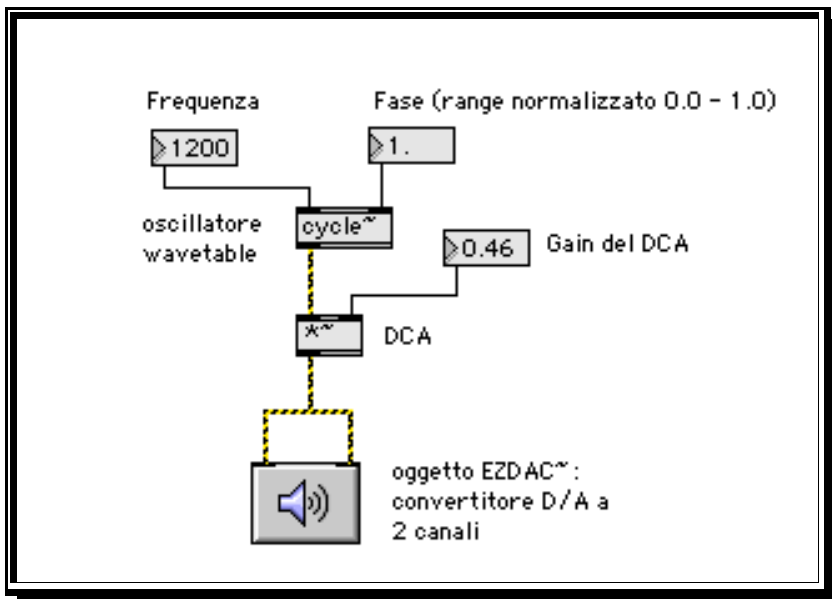
L'oggetto `ezdac~` è il **convertitore D/A** più semplice che Max/MSP mette a disposizione. Esso manda alle uscite fisiche 1 e 2 della scheda quello che gli arriva, rispettivamente, alle inlet di sinistra e di destra. È il convertitore migliore da usare se si dispone di una scheda stereo full-duplex (come quella integrata nei mac o nei pc portatili). La caratteristica principale di `ezdac~` è che esso attiva il processing audio se premuto, come un normale bottone, senza necessità di ulteriori messaggi.

L'oggetto `cycle~` è un oscillatore wavetable, che genera di default, senza ulteriori argomenti, una sinusoide, la cui frequenza può essere specificata come argomento (vedi il numero 600 scritto dopo il nome dell'oggetto, in figura). Appena attiviamo il processing, premendo sull'`ezdac~`, sentiremo una sinusoide pura a 600Hz, al livello massimo. Si noti che in Max/MSP le ampiezze hanno un range che va da 0.0 a 1.0, e sono dunque specificate come numeri float. Se imponiamo un'ampiezza di valore maggiore di 1.0, ad esempio 2.5, il risultato sarà una forma d'onda distorta.

La figura 11 mostra un miglioramento della patch di base: sono stati aggiunti due number box connessi a `cycle~`, per specificare frequenza e fase della sinusoide, ed un moltiplicatore di segnali con relativo number box. Difatti `cycle~` ha due inlet: quella di sinistra riceve la frequenza dell'oscillatore, quella di destra la fase, specificata come frazione del ciclo, con un float da 0.0 a 1.0.

Ad esempio: 0.5 corrisponderà a  $180^\circ$ , 0.75 a  $270^\circ$  e 0.25 a  $90^\circ$ .

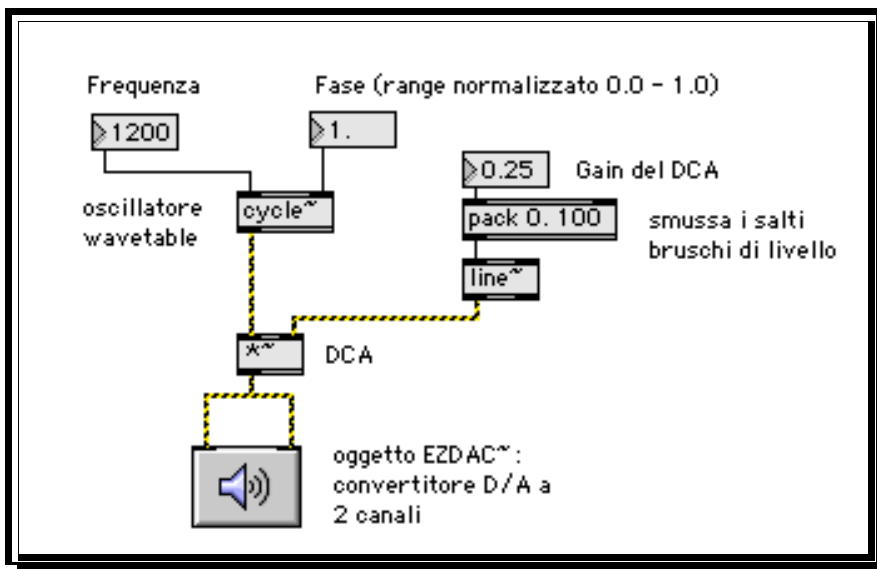
L'oggetto moltiplicatore (simbolo `*~`) moltiplica tutti i campioni del segnale che riceve nella sua inlet di sinistra per il float (naturalmente inferiore a 1.0, per non avere clip del segnale...) che riceve nella sua inlet di destra. Potrebbe anche moltiplicare due segnali, ottenendo così un **ring modulator**. L'oggetto moltiplicatore può essere immaginato come un **DCA (Digital Controlled Amplifier)**.



**Figura 11: Frequenza, Fase e livello attraverso number box e DAC.**

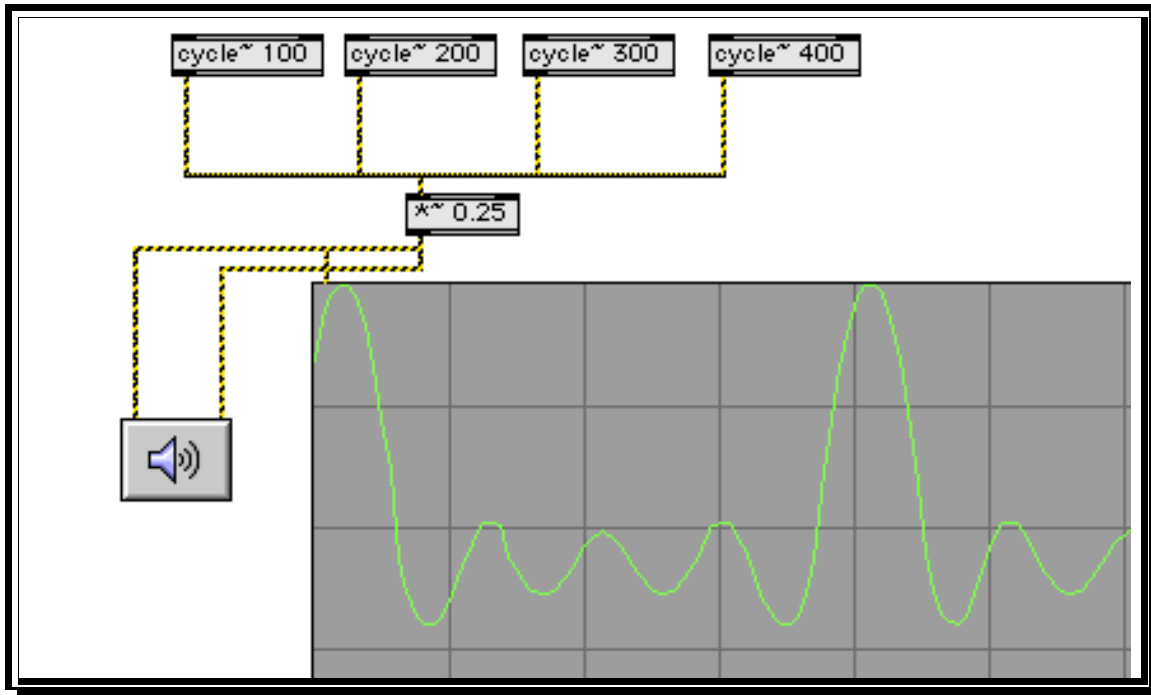
Se tentiamo di cambiare il livello del segnale dal number box indicato con gain in fig.11, udiremo dei glitch, dovuti al fatto che il livello subisce sbalzi improvvisi. Questo può essere eliminato con l'accorgimento della fig.12.

L'oggetto **pack** crea una lista di due valori, numero pari alle sue inlets, e la comunica all'oggetto **line~**. Tale oggetto genera una rampa continua, una funzione a forma di retta, interpolando fra due valori in un certo tempo, il cui valore in millisecondi deve essere fornito nella sua inlet di destra o come secondo argomento di una lista ricevuta nella inlet di sinistra (100ms nel nostro caso).



**Figura 12.**

Come ulteriore esempio possiamo realizzare una patch additiva. Il numero di moduli sinusoidali che abbiamo a disposizione è virtualmente illimitato, ed inoltre è disponibile un controllo molto preciso su frequenza, ampiezza e, soprattutto, fase di ogni componente. In fig.13 si vede la somma di 4 sinusoidi a frequenze armoniche, la forma d'onda risultante è visualizzata tramite l'oggetto **scope~**.



**Figura 13: Esempio di Sintesi Additiva. Il DAC con gain 0.25, ossia 1/4, serve ad evitare clip del segnale.**

Nell'esempio di fig.13 si nota come collegando più segnali o controlli alla stessa inlet, essi si sommano.

### ***Altre Forme d'Onda***

#### ***SawTooth***

L'onda a dente di sega si ottiene con l'oggetto **phasor~**. Esso genera un segnale a dente di sega che assume valori da 0.0 a 1.0. Per ottenere un segnale bipolare, dobbiamo sottrarre 0.5 e moltiplicare per 2.0. si veda la fig.14.

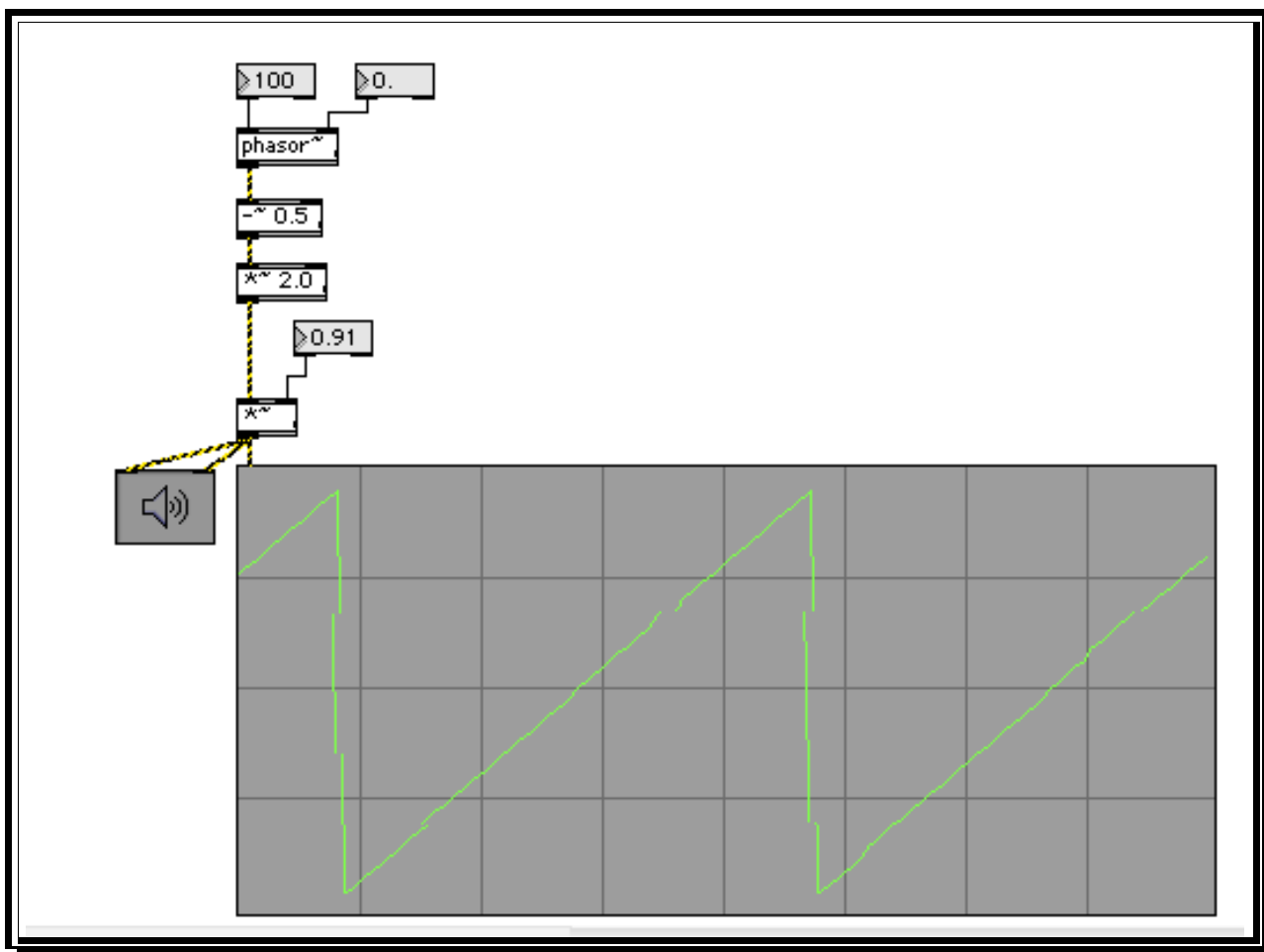


Figura 14: SawTooth WaveForm.

### ***Onda Triangolare***

La forma d'onda triangolare si può ottenere dall'unione di **phasor~** e **triangle~**, un wavetable che genera forme d'onda triangolari.

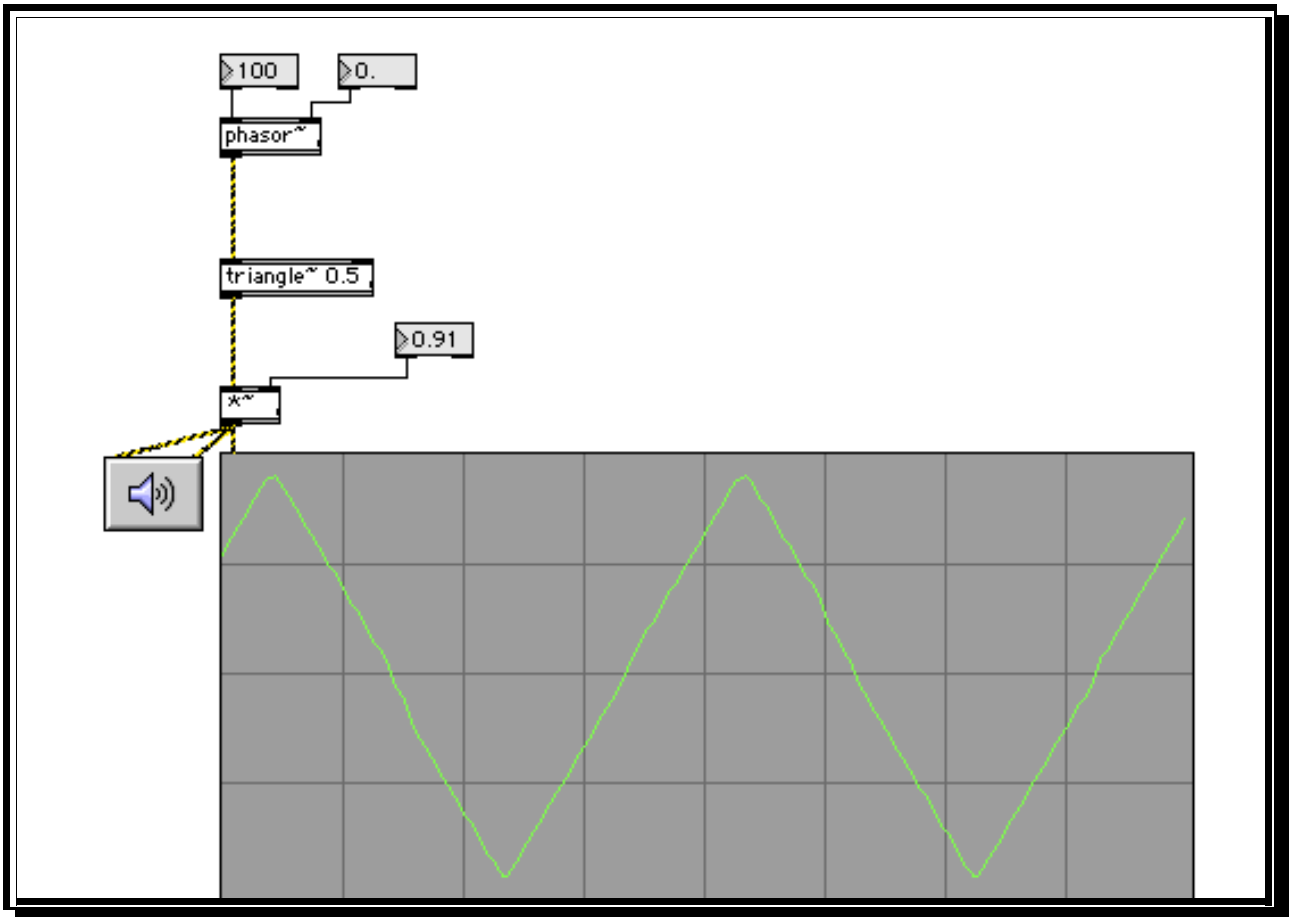


Figura 15: Onda Triangolare.

**Onda Quadra e Rumore Bianco.**

In fig.16 è indicato uno dei modi in cui è possibile ottenere un'onda quadra: amplificando molto `cycle~` e clippandolo. Un altro modo è quello di usare l'oggetto `train~`, che genera un treno d'impulsi unipolare, rendendolo bipolare, come per il dente di sega. Il rumore bianco ha un oggetto dedicato: `noise~`.

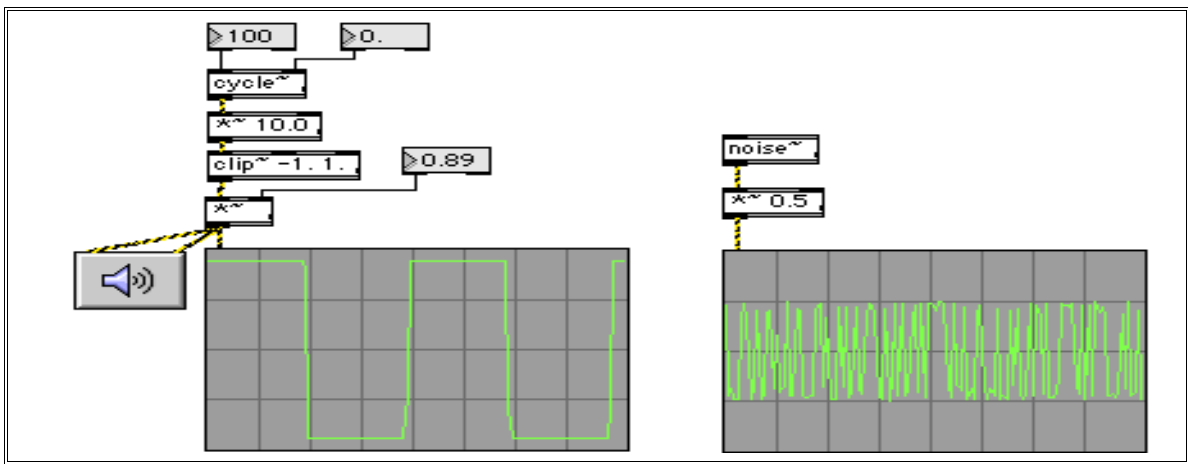


Figura 16: Onda Quadra e White Noise.

## ***L'oggetto Patcher: l'incapsulazione in Max/MSP***

Ogni linguaggio di programmazione dispone di tecniche di **incapsulazione** più o meno complesse e strutturate. Ad esempio il C/C++ si basa sulle funzioni, e nella versione a oggetti, sulle classi. Max/MSP basa il meccanismo dell'incapsulazione sull'oggetto **patcher**. In pratica è possibile racchiudere una parte della patch all'interno di una **sub-patch**, ottenendo il duplice scopo di avere patches pulite e organizzate, e di dividere funzionalmente e logicamente l'algoritmo che stiamo creando. Supponiamo di voler creare un effetto di **delay stereo**. Dobbiamo prevedere due linee di ritardo, con eventuale percorso di **feedback**. Perciò possiamo individuare subito l'unità logica del nostro algoritmo : la linea di ritardo. Una volta programmata, e racchiusa in un **patcher**, diverrà facilissimo duplicarla, ottenendo così non due ma quante linee di ritardo desideriamo!

Vediamo prima come si realizza un delay con percorso di feedback.

In Max/MSP le linee di ritardo si realizzano con gli oggetti **tapin~** e **tapout~**.

L'oggetto **tapin~** è un buffer, la cui grandezza in millisecondi è specificata dall'argomento che segue il nome, che alloca abbastanza RAM per ospitare la quantità di audio che occupa quel tot di millisecondi (la quantità di RAM necessaria dipenderà dalla fs scelta). Il buffer si aggiorna continuamente, contenendo sempre gli n msec del segnale in ingresso più recenti.

L'oggetto **tapout~**, l'unico che può essere usato con **tapin~**, è invece un puntatore che legge il buffer creato da **tapin~**, con un ritardo pari a quello specificato come argomento. Con più argomenti si creano **delay multitap**.

Per creare il feedback, ossia poter controllare il numero di ripetizioni, è necessario rimandare il segnale che esce da **tapout~** all'ingresso di **tapin~**, controllandone la quantità con un DAC.

Il tutto è riportato in fig. 17.

Si noti che abbiamo assegnato gli ingressi alla linea di ritardo attraverso l'oggetto **ezdac~**, il complementare dell'**ezdac~**. Esso riceve gli ingressi 1 e 2 della scheda, ed è dunque un convertitore A/D. Si noti anche che non abbiamo sommato il segnale wet al dry. In figura è rappresentata solamente una linea di ritardo.

Procediamo adesso all'incapsulazione. Aprendo un oggetto box e scrivendo **patcher** o più brevemente **p**, verrà aperta una sub-patch, in un'alta finestra, dove potremo copiare la linea di ritardo già programmata. Tale sub-patch deve essere nominata. Chiudendo tale sub-patch, avremo un'oggetto **patcher**, con il nome da noi assegnato, che svolgerà le funzioni della linea di ritardo e potrà essere replicato quante volte vorremo. Le comunicazioni con l'esterno sono garantite dagli oggetti **inlet** e **outlet** (si noti che tali oggetti sono identici sia per i controlli che per i segnali audio, costituendo solo porte verso l'esterno del **patcher**). Il risultato è riportato in fig. 18, mentre in figura 19 è riportata la patch completa, con due linee di ritardo e il controllo **dry/wet**. Si noti che selezionando un oggetto **inlet** o **outlet**, e premendo **Mela-i**, apparirà una finestra di dialogo nella quale è possibile scrivere una descrizione dell'**inlet**, che apparirà nella assistance area della finestra principale, quando il mouse punterà l'**inlet** relativa.

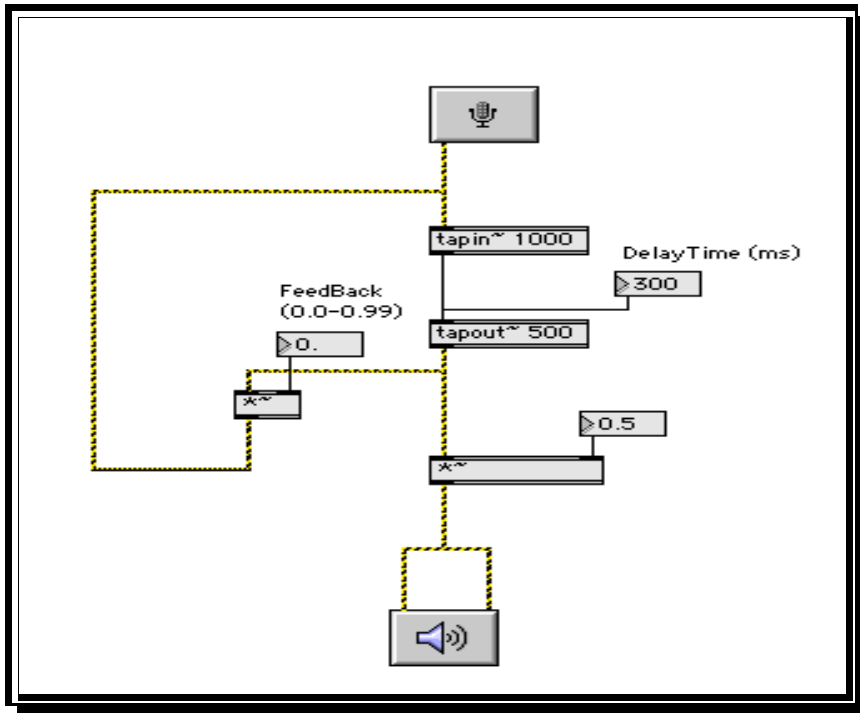


Figura 17: Linea di Ritardo con Feedback.

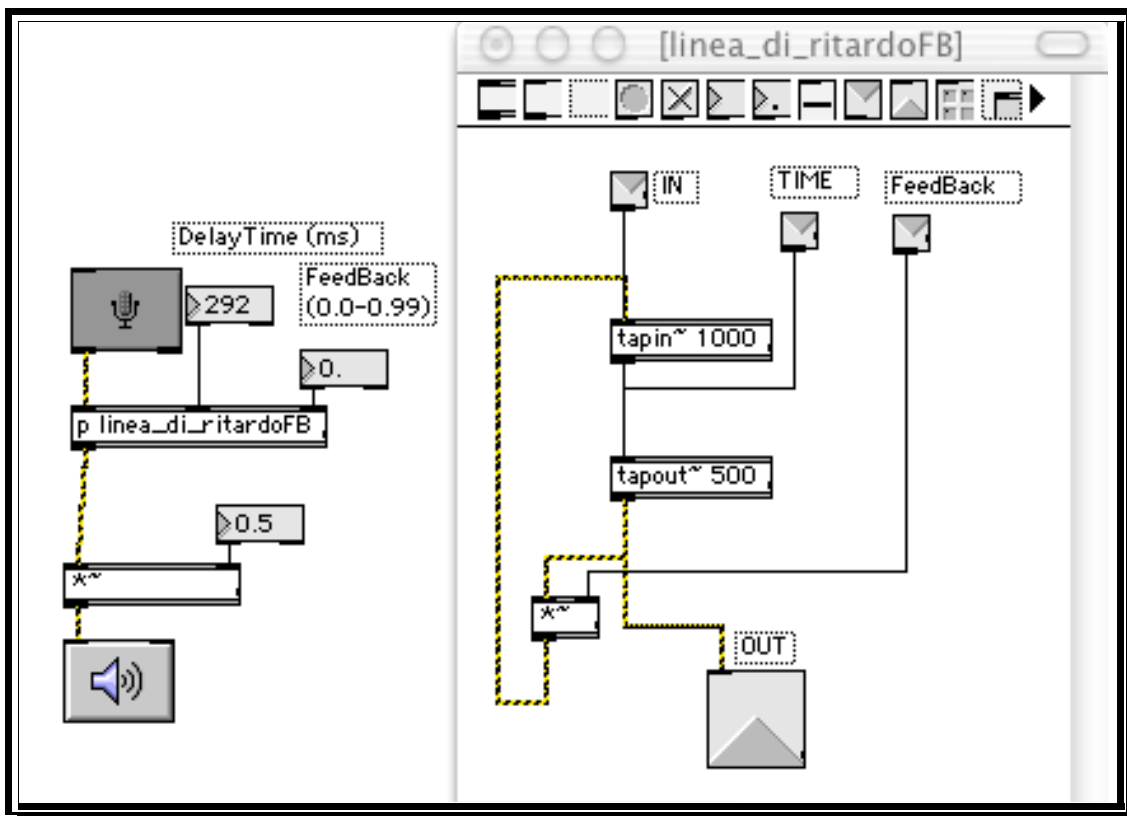


Figura 18: Patcher e sub-patch.

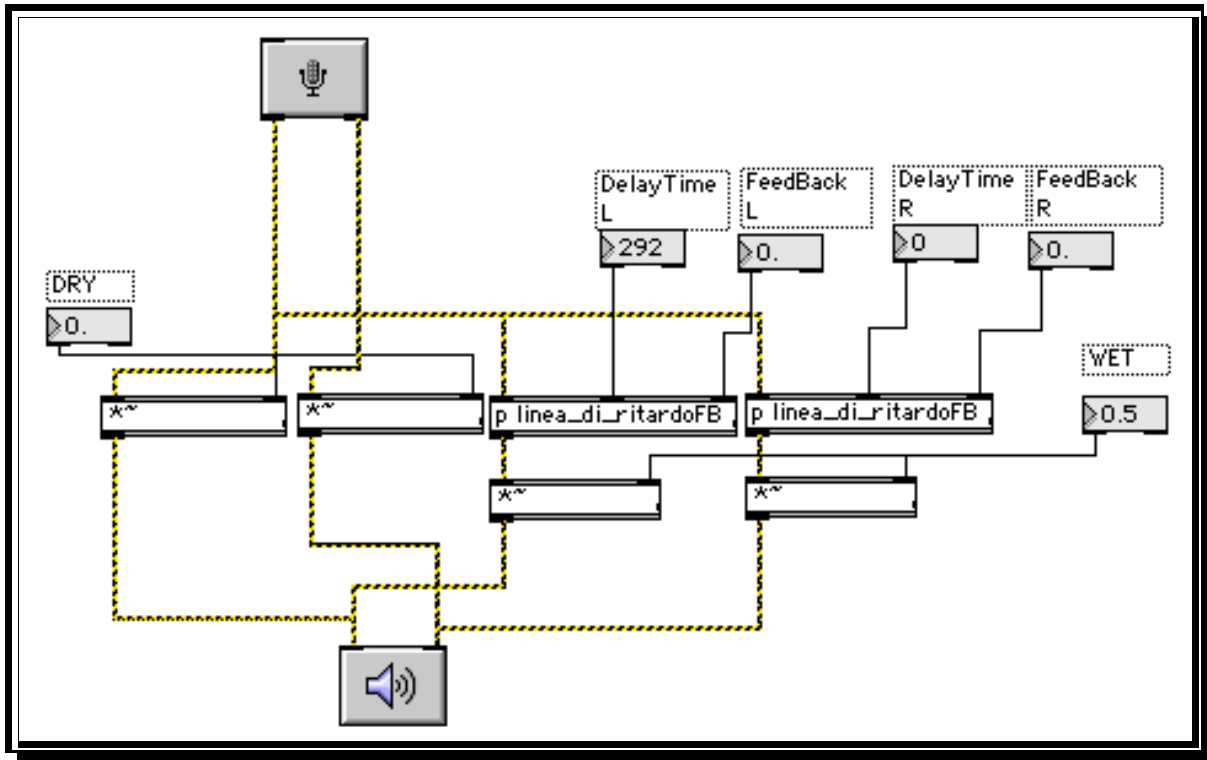


Figura 19: La Patch Delay completa. Notare le due sub-patch replicate.