UNIVERSITÀ DEGLI STUDI DI MILANO
FACOLTÀ DI SCIENZE E TECNOLOGIE

CORSO DI LAUREA TRIENNALE IN INFORMATICA MUSICALE

INTERAZIONE TRA MAX E PYTHON NELL'AMBITO DELLA PERFORMANCE MUSICALE

Relatore: Prof. Luca Andrea Ludovico
Correlatore: Prof. Giorgio Presti

Tesi di Laurea di:
Finizio Domenico
Matr. Nr. 834069

ANNO ACCADEMICO 2018-2019

Indice

Capitolo 1

Introduzione

1.1 Abstract

Il presente elaborato documenta il lavoro di ricerca svolto presso il Laboratorio di Informatica Musicale nell'anno accademico 2017/2018, al fine di realizzare un'implementazione della Sintesi per Modello Sinusoidale mediante gli ambienti software Max/MSP e Python.

La Sintesi per Modello Sinusoidale è uno strumento di manipolazione del suono sviluppato da Xavier Serra, appartenente all'insieme di tecniche di sintesi dei modelli spettrali. Lo schema di funzionamento della Sintesi per Modello Sinusoidale prevede che un utente inserisca dei parametri che regolano l'analisi delle componenti spettrali di un suono in ingresso. Viene così creata una rappresentazione del suono di partenza, modellato in una parte periodica ed in una parte stocastica. Successivamente, le informazioni estratte vengono utilizzate per guidare la sintesi. L'assunto di partenza della Sintesi per Modello Sinusoidale è che la maggior parte dei suoni di interesse musicale possono essere descritti da una componente periodica, le parziali, e da una componente residuale, costituita da segnale stocastico.

La Sintesi per Modello Sinusoidale è il risultato di studi sperimentali svolti nel contesto di un dottorato di ricerca presso l'Università di Stanford. Dal 1989, anno di pubblicazione della tesi di dottorato di Serra, questa tecnica di sintesi è stata oggetto di studi e di implementazioni che l'hanno resa fruibile in diversi ambienti software. Il presente elaborato testimonia l'intenzione di realizzare la Sintesi per Modello Sinusoidale all'interno di un ambiente di programmazione in cui non è nota un' implementazione, Max/MSP.

Al fine di realizzare tale ricerca l'attenzione è stata rivolta verso alcuni argomenti teorici, come ad esempio i modelli spettrali per la generazione di segnali sonori. Questo insieme di tecniche è uno tra i modelli di sintesi audio che nel corso degli anni si sono sviluppati in ambito di Informatica Musicale. La Short Time Fourier Trasform (STFT), che, basandosi sulla trasformata di Fourier, costituisce il sistema di analisi più diffuso per ottenere una descrizione del contenuto in frequenza di un campione audio in ingresso. Il modello sinusoidale, particolare modello di analisi/sintesi che si basa su analisi STFT per ricostruire il suono di partenza come somma di sinusoidi. Il modello sinusoidale più un residuo, tecnica che prevede la rappresentazione del segnale sonoro scomposto in una parte periodica ed una parte residuale.
I principali argomenti che costituiscono la base teorica della Sintesi per Modello Sinusoidale verranno esposti nel presente elaborato.

Lo studio dei principi teorici che regolano gli algoritmi del modello si è svolto congiuntamente alla ricerca delle sue realizzazioni software. Tra gli esempi di codice noti in letteratura che implementano in maniera esaustiva la Sintesi per Modello Sinusoidale, l'interesse si è focalizzato su un insieme di file sorgenti scritti in linguaggio di programmazione Python da Xavier Serra, che rendono eseguibile la Sintesi per Modello Sinusoidale tramite un'interfaccia grafica. La maggior parte del codice di partenza Python è stato modificato per rendere eseguibile il modello all'interno delle patch Max/MSP. A fronte di varie problematicità emerse durante i tentativi di cooperazione il risultato è una pluralità di soluzioni. Ad accomunare le diverse soluzioni sviluppate ci sono i parametri che un utente inserisce in ingresso al sistema. Un altro fattore costante alle diverse strade intraprese è la decisione di svolgere in Python la parte del modello che si occupa dell'analisi. Per implementare la sintesi, invece, tre percorsi sono stati intrapresi:

- Realizzazione della sintesi all'esterno di una patch Max/MSP, tramite codice Python. Dalla patch Max/MSP viene lanciato un'interfaccia grafica che permette all'utente di eseguire il modello e compiere analisi e sintesi in un unico passo di computazione.
- Realizzazione della sintesi all'interno di una patch Max/MSP. La patch Max/MSP utilizza dei file di analisi per effettuare la sintesi.
- Realizzazione della sintesi all'interno di una patch Max/MSP. Similmente al caso precedente la sintesi ha origine da dei file di analisi. Le differenze rispetto al caso precedente consistono nella tipologia di file di analisi generato e sostanziali differenze negli oggetti Max/MSP adottati per implementare la sintesi nella patch.

Nel percorso di implementazione si sono dunque evidenziate le differenze tra questi due ambienti software: Python è propriamente un linguaggio di programmazione. Max/MSP, diversamente, è definibile come un ambiente di sviluppo grafico, progettato e pensato per la performance musicale. In fase sperimentale è stato necessario rafforzare e consolidare la conoscenza degli ambienti, approfondendone gli aspetti pratici di programmazione al fine di indagare i punti di flessibilità tra Max/MSP e Python. Questi sforzi sono stati necessari per rendere possibile la comunicazione tra i due linguaggi.
Lo scambio di informazioni tra i due ambienti diversi ha posto questioni importanti legate alla consistenza dei dati ed alla formattazione. Particolare attenzione è stata dedicata a questa problematica, e le strategie adottate saranno esposte.

L'ambiente software, risultato degli sforzi al fine di implementare la Sintesi per Modello Sinusoidale all'interno di Max/MSP, partendo dal codice sorgente Python, è un ambiente ibrido, dove il nucleo operativo è costituito dall'interazione tra i due linguaggi di programmazione. L'interazione tra questi due ambienti di programmazione, con le problematicità emerse, e le possibili soluzioni adottate o che possono essere sviluppate in futuro è oggetto di studio del presente elaborato.

Diverse motivazioni hanno convogliato l'interesse verso il presente progetto di ricerca. La curiosità verso le tecniche di sintesi nasce durante le lezioni di Programmazione Timbrica tenute dal professor Luca Andrea Ludovico presso il dipartimento di Informatica Musicale dell'Università degli Studi di Milano. L'interesse per Max/MSP e le possibili interazioni con altri ambienti di sviluppo ha avuto origine durante alcune lezioni tenute da Andrea Agostini e Daniele Ghisi presso la Civica Scuola di Musica Claudio Abbado. Alla luce di alcune fruttuose conversazioni svolte presso il Conservatorio Giuseppe Verdi di Milano insieme alla docente di Composizione Musicale Elettroacustica del Conservatorio Giuseppe Verdi di Milano Sylviane Sapir e il docente presso il dipartimento di Informatica Musicale dell'Università degli Studi di Milano Federico Avanzini è nata l'idea di porre la Sintesi per Modello Sinusoidale al centro di una ricerca trasversale che coinvolgesse Max/MSP. Xavier Serra, docente presso l'Università Pampa Fabra di Barcellona e autore della Sintesi per Modello Sinusoidale, ha suggerito di utilizzare il codice Python da lui sviluppato come punto di partenza per una eventuale implementazione in Max/MSP.

1.2 Stato dell'arte della Sintesi per Modello Sinusoidale

Dalla sua formulazione ad oggi la Sintesi per Modello Sinusoidale ha conosciuto una larga diffusione in ambito accademico.

- Un capitolo dedicato alla trattazione della Sintesi per Modello Sinusoidale è presente nel libro "Musical Signal Processing" di G. De Poli, A. Picialli, S. T. Pope, and C. Roads.

- All'interno del libro "Digital Audio Effects" pubblicato nel 2002 di U. Zolzer, un capitolo è dedicato alla trattazione della Sintesi per Modello Sinusoidale. Nel capitolo in questione sono presenti esempi di codice MATLAB che implementano tutte le funzionalità della tecnica di sintesi, scritto oltre che da Serra da altri collaboratori (J. Bonada, X. Amatriain, A. Loscos).

- Nel 2000, l'articolo scientifico "Using the Sound Description Interchange Format within the SMS Applications" scritto da Maarten de Boer, Jordi Bonada e Xavier Serra, documenta il progetto di utilizzare lo standard SDIF come linguaggio per facilitare lo scambio di dati all'interno della Sintesi per Modello Sinusoidale, ad esempio tra analisi e sintesi, ma anche all'esterno, verso altri programmi.

- Nel 2004 è stato pubblicato un progetto di Joan Pampin del Center for Digital Arts and Experimental Media, appartenente alla University of Washington dal titolo "Analysis/Trasformation/Synthesis". ATS è una libreria di funzioni scritta in LISP che permette di effettuare analisi, trasformazioni e sintesi spettrale, sulla base del modello sinusoidale più una componente di rumore a bande critiche. Rispetto alla Sintesi per Modello Sinsuoidale, da cui il progetto ha origine, si differenzia per l'introduzione di un modello psicoacustico in fase di analisi spettrale. Il modello psicoacustico aggiunge un fattore di elaborazione sulla base del fenomeno del mascheramento acustico per fornire in fase di analisi dati che presentano maggiore verosimiglianza con l'esperienza di ascolto di un essere umano.

- Nel 2009 presso la dodicesima edizione di "International Conference on Digital Audio Effects" viene presentato un metodo per controllare la parte di sintesi della Sintesi per Modello Sinusoidale in real-time all'interno di PureData. Partendo dal codice originale che implementa la Sintesi per Modello Sinusoidale,

vengono sviluppati degli externals real-time per PureData che permetto maggiore flessibilità nell'utilizzo degli strumenti di sintesi spettrale.

- In anni più recenti è stato realizzato un corso sulla Sintesi per Modello Sinusoidale tenuto dal professor Xavier Serra e dal professor Julius Smith III tramite tramite il circuito di educazione online Coursera. Il sito del corso si trova alla pagina https://www.coursera.org/learn/audio-signal-processing. Nel corso viene utilizzato codice Python disponibile al sito https://github.com/MTG/sms-tools.

1.3 Stato dell'arte dell'interazione tra Max/MSP e Python

Il numero di progetti noti in letteratura che forniscono strumenti per l'interazione tra Python e Max/MSP è esiguo. Il paradigma che regola l'interazione tra i due ambienti di sviluppo adottato nel progetto di ricerca oggetto del presente elaborato, sulla base della stabilità mostrata e della reperibilità, prevede l'utilizzo di Max/MSP come ambiente primario all'interno di cui eseguire degli script Python.

- Thomas Grill ha sviluppato un oggetto external per Max/MSP che integra il linguaggio Python all'interno di Max/MSP. La versione di Python supportata è la 2.1, l'oggetto è in fase di testing per le versioni successive di Python. Il codice sorgente si può reperire alla pagina web https://https://github.com/grrrr/py.
- Jython è un implementazione del linguaggio Python all'interno della piattaforma Java. Codice Jython può essere eseguito in Max/MSP tramite l'oggetto mxj. Mxj consente infatti di eseguire codice Java in Max/MSP. Il codice sorgente relativo all'oggetto mxj si trova alla pagina web https://github.com/Cycling74/max-mxj.
- Jeremy Bernstein e Bill Orcutt hanno sviluppato un external per Max/MSP che permette di eseguire comandi di terminale. L'oggetto *shell* offre diverse possibilità di utilizzo all'interno di Max/MSP. Nel presente progetto di ricerca, è stato utilizzato per eseguire codice Python. Il codice sorgente dell'oggetto *shell* si può trovare alla pagina web https://github.com/jeremybernstein/shell.

1.4 Struttura dell'elaborato

Il primo capitolo introduce il caso di studio, gli argomenti e le strategie adottate in fase di elaborazione, fornendo le motivazioni che hanno portato al presente progetto

di ricerca. Viene esposto lo stato dell'arte degli studi che riguardano la Sintesi per Modello Sinusoidale e delle applicazioni che prevedono un'interazione tra Python e Max/MSP.
Il secondo capitolo fornisce una panoramica della Sintesi per Modello Sinusoidale. Sono presenti riferimenti alle esperienze di ricerca più significative che hanno portato alla formulazione della Sintesi per Modello Sinusoidale. Vengono illustrati gli argomenti che ne costituiscono la base, con riferimento allo schema analisi/sintesi per la generazione di segnali sonori, alla STFT, al modello sinusoidale e al modello sinusoidale più un residuo.
Il terzo capitolo illustra la tecnologia utilizzata per svolgere il progetto di implementazione della Sintesi per Modello Sinusoidale. Viene presentato l'ambiente di sviluppo grafico Max/MSP, e gli elementi che lo caratterizzano. Tra questi l'external di Max/MSP *shell* utilizzato per eseguire script di Python all'interno di Max/MSP. Viene data una descrizione del linguaggio di programmazione Python e dei principali packages che sono stati utilizzati nel codice. Il capitolo termina fornendo un esempio di funzionamento dell'ambiente ibrido costituito dall'interazione tra Max/MSP e Python, l'ambiente in cui si è svolta l'implementazione della Sintesi per Modello Sinusoidale.
Il quarto capitolo approfondisce i risultati della ricerca svolta. Vengono esposti tre schemi di funzionamento che prevedono l'interazione tra Python e Max/MSP al fine di realizzare la Sintesi per Modello Sinusoidale, entrando nel dettaglio del codice e delle patch Max/MSP che lo implementano. Vengono esposte le strategie adottate per favorire la comunicazione tra i due ambienti, le problematiche e le eventuali soluzioni intraprese.
Il quinto capitolo espone gli sviluppi futuri del presente progetto, i metodi di risoluzione di eventuali anomalie di funzionamento riscontrate e le idee che compongono il proseguimento del presente percorso di ricerca.

Capitolo 2

Sintesi per Modello Sinusoidale

Il presente capitolo è dedicato alla Sintesi per Modello Sinusoidale, l'argomento di ricerca intorno al quale si è sviluppato il progetto di implementazione. Con l'obiettivo di fornire una contestualizzazione di questa tecnica di sintesi, vengono definiti i confini storici all'interno del quale è la Sintesi per Modello Sinusoidale è stata sviluppata, viene approfondita la teoria retrostante al modello, si fornisce una panoramica delle principali implementazioni note in letteratura.

Come ben illustrato in [8], i modelli spettrali sono un insieme di tecniche per la generazione del suono. Si differenziano da altre tecniche di generazione del suono che si sono sviluppate dagli albori dell'Informatica Musicale ad oggi, come, ad esempio, i modelli astratti o i modelli strumentali. I modelli astratti permetto di governare la generazione del suono tramite i parametri rilevanti che controllano formule matematiche. I modelli strumentali determinano le caratteristiche fisiche delle sorgenti che generano un suono.

L'insieme di tecniche che compongono i modelli spettrali prevedono un analisi su un suono in ingresso al fine di ricavare i parametri che servono per effettuare la sintesi. Come evidenziato da [7], nell'ambito della presente trattazione, i suoni vengono modellati come sinusoidi stabili più rumore. Le sinusoidi rappresentano le parziali e il rumore rappresenta la parte residuale. La procedura di analisi rileva le parziali studiando le caratteristiche spettrali tempo-varianti di un suono. Nella sintesi le parziali sono rappresentate da sinusoidi tempo-varianti. La parte periodica viene poi sottratta dal suono originale e la parte residuale è rappresentata da un rumore filtrato da dei coefficienti tempo-varianti.

Durante l'analisi si estraggono dunque le informazioni sulle caratteristiche del segnale che ne permettono una rappresentazione fedele. Queste informazioni possono

essere modificate per produrre una rappresentazione di un segnale diverso dal segnale di partenza. Ne consegue che, nel caso in cui le informazioni vengono modificate, il segnale sonoro in uscita dalla sintesi risulta modificato, altrimenti il segnale in uscita risulta, da un punto di vista percettivo, identico al suono di ingresso.
La figura 1 illustra questo schema di lavoro.

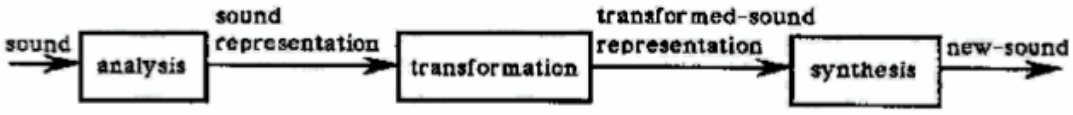

Figura 1: Schema di funzionamento dei modelli spettrali.

Il comportamento di un modello spettrale è fortemente influenzato dai modelli adottati in fase di analisi ed in fase di sintesi. La Sintesi per Modello Sinusoidale, impiega la tecnica di STFT in fase di analisi e utilizza la sintesi additiva per realizzare la sintesi.

2.1 Storia della Sintesi per Modello Sinusoidale

Come si evince da [9], la sintesi additiva è stata la prima tecnica di modellamento dello spettro. E' radicata nel teorema di Fourier, che sostiene che ogni forma d'onda periodica può essere modellata come somma di sinusoidi con ampiezze e frequenze tempo-varianti. E' stata una delle prime tecniche di sintesi della Computer Music, descritta in maniera estensiva nel primo articolo della prima edizione del Computer Music Journal (Mooer 1977).

Nella tesi di Xavier Serra [7] si afferma che uno dei primi sistemi di analisi/sintesi per elaboratore, pensato per applicazioni musicali, fu realizzato da David Luce negli anni '60. Il metodo ideato da Luce era quello di determinare le ampiezze e le frequenze di ognuna delle parziali di un suono come funzioni del tempo. La sintesi aveva come unico scopo di verificare la correttezza dell'analisi svolta. Questo modello fu negli anni successivi migliorato ed ampliato.

Nel 1987, presso il Center for Computer Research in Music and Acoustics (CCRMA) del Dipartimento di Musica dell'Università di Stanford, viene pubblicato un articolo scientifico a cura di Julius Smith e di Xavier Serra dal titolo "PARSHL: An

Analysis/Synthesis Program for Non-Harmonic Sounds Based on a Sinusoidal Representation". Come espresso in [10], PARSHL è un sistema di analisi/sintesi del suono basato sulla STFT. Dato lo spettro di un suono in ingresso, la STFT permette di acquisire informazioni di frequenza, ampiezza e fase delle sinusoidi. Svolta l'analisi delle componenti sinusoidali del suono dato in ingresso, le informazioni estratte vengono parametrizzate. Le funzioni di frequenza, ampiezza e fase delle sinusoidi diventano i parametri che guidano un banco di oscillatori. Tramite la sintesi additiva è quindi possibile sintetizzare il suono di partenza, ottenendo una riproduzione del suono originale, con un collegamento diretto tra la scelta dei parametri che guidano la SFTF e la fedeltà del suono sintetizzato rispetto al suono di partenza.
PARSHL è stato progettato per analizzare registrazioni di pianoforte, ma ha dimostrato di essere un ottimo strumento per estrarre i parametri per la sintesi additiva anche per suoni radicalmente inarmonici, costituendo di fatto un antecedente storico fondamentale nello sviluppo della Sintesi per Modello Sinusoidale.

Gli studi e gli sviluppi successivi di PARSHL hanno portato alla formulazione nel 1989 della Sintesi per Modello Sinusoidale. La Sintesi per Modello Sinusoidale è un modello spettrale che si basa su un sistema di analisi/sintesi per la manipolazione e generazione del suono. Il modello di analisi è basato sull'elaborazione spettrale, il modello di sintesi è basato sulla sintesi additiva.

Gli assunti alla base della ricerca che ha condotto alla formulazione della Sintesi per Modello Sinusoidale sono due:

- Ogni suono può essere rappresentato in maniera intercambiabile nel dominio del tempo da una forma d'onda oppure nel dominio delle frequenze da un insieme di spettri.
- Una buona parte dei suoni di interesse musicale può essere scomposta in una componente deterministica, approssimata da una somma di sinusoidi tempo-varianti e in una componente stocastica ossia non pre-determinabile.

La valutazione qualitativa del sistema deve tenere in considerazione diverse caratteristiche:

- Il sistema deve prevedere una rappresentazione flessibile, ossia facilmente trasformabile.
- Il sistema deve essere efficiente da un punto di vista computazionale.
- La rappresentazione deve essere giudicata solo dalla qualità del suono che produce, dal momento che l'interesse è strettamente musicale. Assumendo questo punto di vista molti aspetti delle caratterische spettrali del suono originale diventano poco rilevanti.

La figura 2 illustra lo schema di funzionamento della Sintesi per Modello Sinusoidale. Nella figura si evidenzia come al centro dell'analisi vi sia la rappresentazione nel dominio delle frequenze fornita dall'analisi di Fourier.

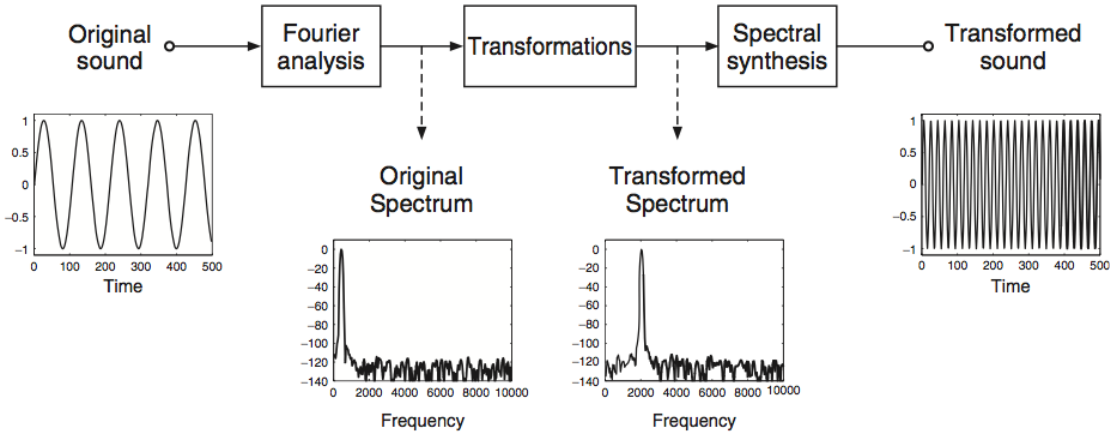

Figura 2: Schema base di funzionamento della Sintesi per Modello Sinusoidale.

Come ben illustrato in [12], trasformare un segnale sonoro dal dominio del tempo al dominio delle frequenze è un processo delicato. In questo procedimento la scelta di alcuni parametri diventa fondamentale, e talvolta, forza ad effettuare dei compromessi. In un primo momento si può ritenere che alcuni parametri abbiano poco a che fare con l'approccio ad alto livello che si intende valutare alla fine, come ad esempio la dimensione della finestra di analisi. In realtà in questi passi intermedi di analisi si ricavano le caratteristiche rilevanti dello spettro. Per tale ragione, grazie ad una flessibile rappresentazione, la modifica delle caratteristiche del suono ad un livello basso di dettaglio comporta una modifica ad alto livello, ossia quello percettivo.

Un' esemplificazione delle possibilità di questo approccio potrebbe essere quella di analizzare un suono parzialmente armonico, ricavare la frequenza fondamentale e la forma spettrale, e, successivamente, agire sulla frequenza fondamentale, modificandola, ma lasciando inalterato lo spettro.
La figura 3 mostra con maggiore dettaglio lo schema.

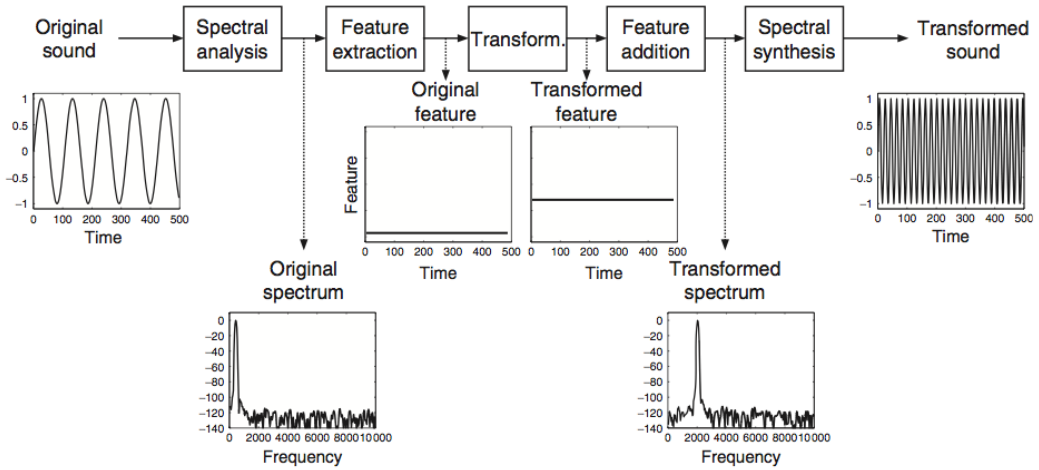

Figura 3: Schema più dettagliato di funzionamento della Sintesi per Modello Sinusoidale.

2.2 Background tecnico

2.2.1 Modelli Spettrali

In [7] viene evidenziato come il sistema uditivo rilevi in ogni momento, senza interruzione, le frequenze presenti nei suoni in ingresso. La coclea, delicato organo dell'orecchio interno, può essere schematizzato come un insieme di filtri passa banda, ognuno incentrato su una determinata frequenza e con una banda passante molto stretta.
Il funzionamento di un analizzatore di spettro mostra delle analogie con il sistema uditivo degli esseri umani. Le rappresentazioni spettrali sono state oggetto di numerosi progetti di ricerca. Gli studi esposti in [3], [1], [2] sono alcuni importanti studi precedenti allo sviluppo della Sintesi per Modello Sinusoidale, che riguardano le rappresentazioni spettrali. Queste rappresentazioni, note anche come rappresentazioni di Fourier, sono in uso in varie applicazioni tecniche e scientifiche. Nell'ambito della computer music, in particolare, sono state largamente utilizzate per l'analisi e la sintesi dei suoni.

In [7] viene specificato che l'analisi spettrale svolta dal sistema uditivo differisce in alcuni aspetti sia dal metodo di analisi di Fourier divenuto standard, sia dalla STFT. La differenza maggiore sta nel fatto che il sistema uditivo effettua una rappresentazione in scala logaritmica dello spettro, mentre il metodo tradizionale di analisi di Fourier calcola lo spettro in scala lineare. Inoltre determinate caratteristiche del

sistema uditivo non vengono di solito considerate nelle implementazioni standard dell'analisi spettrale. Queste caratteristiche sono, ad esempio, il mascheramento sia in termini temporali che in termini frequenziali e la percezione dell'ampiezza in relazione alla frequenza, fenomeni oggetto di indagine ed esposti in [4], [11] e [6].

2.2.2 Short Time Fourier Trasform

Come espresso in [12], la STFT è la tecnica maggiormente utilizzata nel convertire un segnale nel dominio del tempo ad una rappresentazione del segnale nel dominio delle frequenze. La STFT è espressa dalla seguente equazione:

$$K_l(k) = \sum_{n=0}^{N-1} w(n)x(n + lH)e^{-j\omega_k n},$$

dove $K_l(k)$ è lo spettro complesso di un determinato frame, $x(n)$ è il segnale in ingresso, $e^{-j\omega_k n}$ è una sinusoide complessa avente come frequenza ω_k espressa in radianti, $w(n)$ è la finestra di analisi, l è il numero di frame, k è l'indice della frequenza, H è l'hop size e N è la dimensione della FFT.

La figura 4 mostra lo schema di funzionamento di un sistema di analisi/sintesi basato sulla STFT

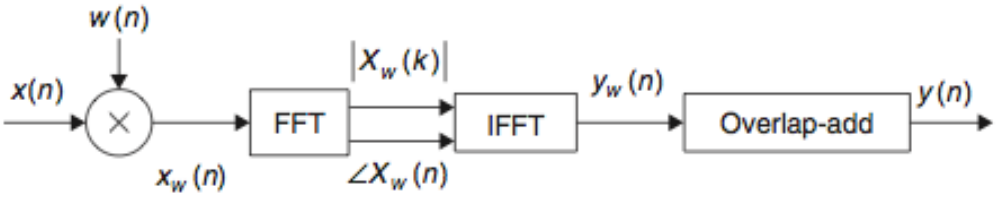

Figura 4: Sistema di analisi/sintesi basato sulla STFT.

Nell'utilizzo della STFT bisogna prestare una particolare attenzione ad alcuni fattori determinanti. La dimensione della finestra di campionamento, ad esempio, indica il numero di campioni che vengono presi per ogni passo di campionamento per essere moltiplicati dal tipo di finestra scelto. Più lunga è la finestra di campionamento, maggiore definizione frequenziale si otterrà. Al contempo, la risoluzione temporale sarà meno raffinata. Un esempio di questo fenomeno può essere osservato nella seguenti figure:

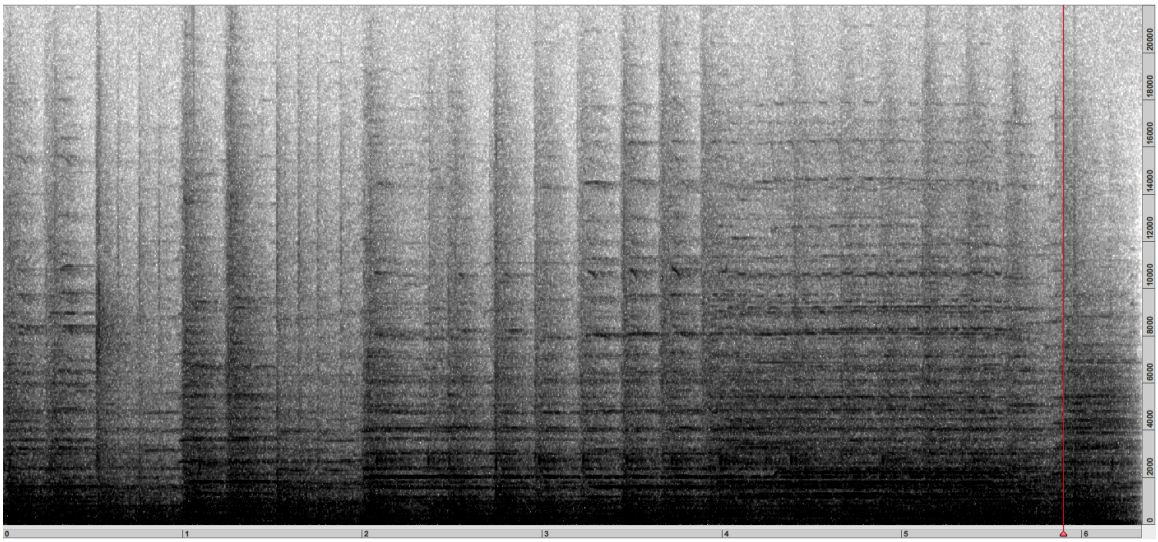

Figura 5: FFT di un segnale audio con finestra di campionamento grande 1024 samples.

Nella figura 5 un campione di orchestra viene analizzato con una finestra di campionamento grande 1024 samples. Nella figura 6 lo stesso campione viene analizzato con una finestra di campionamento grande 8192 samples. Si può notare come l'algoritmo a 1024 samples riesca a conservare meglio la descrizione dei transienti di attacco sul campione audio, dove invece lo stesso algoritmo a 8192 samples abbia una risposta temporale meno raffinata. Al contrario invece questo ultimo esempio mostra una maggiore definizione frequenziale, dello stesso campione a 1024 samples.

Il tipo di finestra utilizzato è un altro fattore da tenere in considerazione, per l'effetto significativo che esercita sulla qualità della rappresentazione spettrale. Una moltiplicazione nel dominio del tempo come quella che avviene nel caso della moltiplicazione di un numero di campione per una finestra di campionamento, diventa nel dominio delle frequenze una convoluzione tra le trasformate di entrambi i segnali. Le finestre tipiche che vengono utilizzate nella fase di analisi sono rettangolare, triangolare, Kaiser-Bessel, Hamming, Hanning e Blackman-Harris.

Le caratteristiche importanti di una finestra di campionamento sono l'ampiezza del lobo principale e la relazione tra il lobo principale e la dimensione del lobo più alto. L'ampiezza del lobo principale viene espressa in campioni spettrali, ossia bins, e insieme, alla dimensione della finestra, definisce la capacità di distinguere due picchi sinusoidali.
Questa capacità è espressa dalla seguente formula:

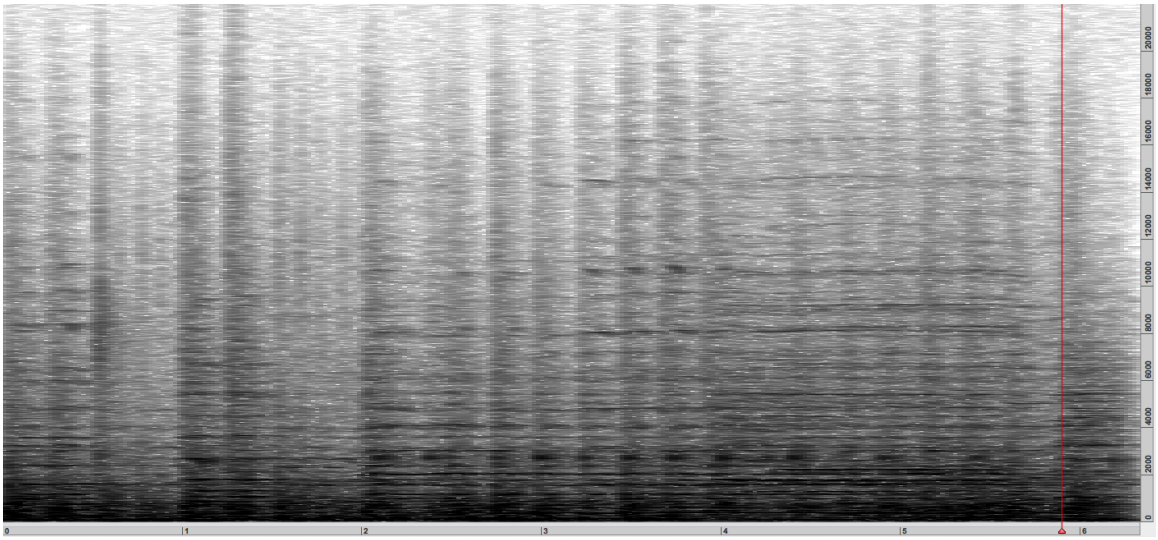

Figura 6: FFT di un segnale audio con finestra di campionamento grande 8192 samples.

$$M >= B_s \frac{f_s}{|f_{k+1} - f_k|}$$

Dove M è la dimensione della finestra, $B_(s)$ è l'ampiezza del lobo principale, $f_(k)$ e $f_(k+1)$ sono le frequenze di due sinusoidi e fs è la frequenza di campionamento.

La relazione tra l'ampiezza del lobo principale e quella del lobo più alto, invece, influenza il quantitativo di distorsione che un determinato picco riceverà dalle parziali circostanti.

Al fine di aumentare la risoluzione frequenziale si utilizza la tecnica di zero padding. Il procedimento di zero padding consiste nell'aggiungere degli zero agli estremi della rappresentazione del segnale che viene moltiplicato con la finestra, per avere la dimensione della FFT maggiore della dimensione della finestra. In questo modo si ottiene un'interpolazione del segnale, che ha appunto l'effetto di incrementare la risoluzione frequenziale.

Una volta calcolato lo spettro di un determinato frame, la finestra si sposta nella posizione successiva del segnale in ingresso. La distanza che intercorre tra i centri di due finestre consecutive è definito hop size. Se la dimensione di hop size è minore della dimensione della finestra si ottiene una sovrapposizione, o overlap. Quando si determina un overlap alcuni campioni sono utilizzati più di una volta nel processo di

analisi. Questo procedimento ha l'effetto di determinare delle transizioni di spettro più morbide nel tempo.

2.2.3 Modello sinusoidale

Il modello sinusoidale utilizza la rappresentazione fornita dalla SFTF per modellare le caratteristiche spettrali tempo-varianti di un suono come somma di di sinusoidi tempo-varianti. Il suono $s(t)$ è modellato da:

$$s(t) = \sum_{r=1}^{R} A_r(t) \cos[\theta_r(t)],$$

dove $A_r(t)$ e $\theta_r(t)$ sono ampiezza e fase istantanea della sinusoide r, e R è il numero totale di sinusoidi.

Come espresso da [12], per ottenere una rappresentazione sinusoidale partendo da un suono, viene eseguita un'analisi per stimare le fasi e le ampiezze istantanee delle sinusoidi. Questa stima viene effettuata in più passi: si calcola la SFTF del suono, poi si rilevano i picchi spettrali, misurando magnitudine, frequenza e fase di ciascun picco e in fine si organizzano i picchi in tracce sinusoidali tempo-varianti. Dopo aver stimato le fasi e le ampiezze istantanee delle sinusoidi si può ricostruire il suono iniziale tramite sintesi additiva. Il modello sinusoidale è una tecnica di analisi/sintesi che offre un guadagno in termini di flessibilità, se comparata ad un'implementazione della sola STFT.

Come espresso in [7], uno degli assunti alla base del modello sinusoidale consiste nel fatto che ogni spettro rappresentato dalla STFT può essere spiegato come somma di un piccolo numero di sinusoidi. Teoricamente una sinusoide che è stabile sia in ampiezza sia in frequenza possiede una rappresentazione stabile di ampiezza e frequenza. Tuttavia questo succede raramente in pratica, poiché i suoni naturali non sono perfettamente periodici e ci sono interazioni tra le componenti. [12] evidenzia che, in natura, solo un esiguo numero di suoni si presentano sufficientemente periodici e senza una componente di rumore, ad esempio tra gli strumenti acustici, la parte stabile di un suono d'oboe. La soluzione adottata nel modello sinusoidale consiste nel rilevare il maggior numero di picchi e ricostruire una determinata parziale tramite un algoritmo di continuazione dei picchi spettrali.

Un picco spettrale è definito come un massimo locale nella magnitudine dello spettro. Fissato un range frequenziale e una soglia di ampiezza, per verificare la presenza

di un determinato picco spettrale basta controllare che il picco in esame ricada all'interno del range frequenziale e che superi la soglia di ampiezza.
Un campione spettrale rappresenta un intervallo di frequenze dato dal rapporto sr/N dove sr è la frequenza di campionamento e N è la dimensione della FFT. Tramite la tecnica dello zero-padding precedentemente esposta si può aumentare il numero di campioni spettrali per Hz e conseguentemente aumentare l'accuratezza della rilevamento di un picco spettrale. Fatta stima di quanto vicina è la forma di un picco da un teorico picco sinusoidale, si può determinare se un picco spettrale corrisponde ad una parziale.

Successivamente alla determinazione dei picchi spettrali di un frame, tramite un algoritmo di continuazione, i picchi vengono organizzati in traiettorie, dove ogni traiettoria modella una sinusoide tempo-variante.
Diverse strategie nel corso degli anni sono state studiate per determinare il modo migliore di collegare le componenti sinusoidali. L'algoritmo proposto da McAulay e Quatieri consiste nel cercare per ogni picco spettrale i più vicini in frequenza nel frame successivo. Serra ha aggiunto a questo algoritmo un insieme di guide frequenziali utilizzate per creare delle traiettorie. I valori delle guide sono ottenuti dai valori dei picchi e dal loro contesto, come ad esempio i picchi circostanti e la frequenza fondamentale. Nel caso di suoni armonici le guide vengono inizializzate secondo la serie degli armonici relativa ad una determinata frequenza fondamentale. Le traiettorie vengono poi calcolate assegnando ad ogni guida il picco più vicino in frequenza. Questa restrizione assume che il suono di ingresso sia monofonico e armonico.
Con le precedenti assunzioni alla base è possibile identificare la frequenza fondamentale F_0 in ogni frame. Esistono diverse strategie per determinare la frequenza fondamentale.

Identificati i picchi ed ordinati, la parte di sintesi consiste nel generare le sinusoidi che corrispondono ai picchi connessi. Le sinusoidi vengono generate nel dominio del tempo tramite sintesi additiva. Come mostra la figura 7, le sinusoidi vengono generate con un banco di oscillatori che controllano la frequenza e l'ampiezza istantanea.

2.2.4 Modello sinusoidale più un residuo

Da [12], si evince che il modello sinusoidale più un residuo modella una parte dell'informazione spettrale come sinusoidi e il restante come STFT. In questo approccio la rappresentazione sinusoidale viene utilizzata per modellare la parte periodica di un suono. La parte residuale, o la sua approssimazione, modella il rimanente, che idealmente dovrebbe essere la componente stocastica. Questa tecnica rappresenta un ulteriore passo avanti in termini di flessibilità rispetto al modello sinusoidale. Il suono

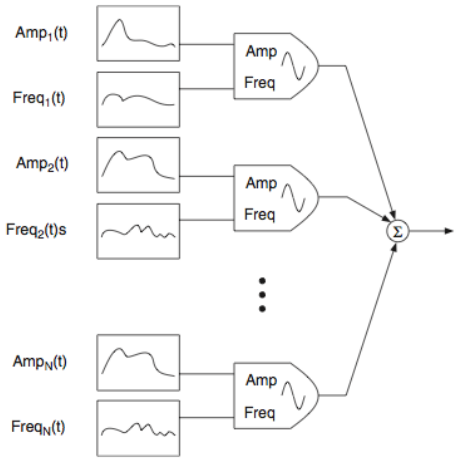

Figura 7: Banco di oscillatori che implementano sintesi additiva.

$s(t)$ è dunque modellato da:

$$s(t) = \sum_{r=1}^{R} A_r(t) \cos[\theta_[r] + (t)] + e(t),$$

dove $A_r(t)$ e $\theta_r(t)$ sono ampiezza e fase istantanea della sinusoide r, e R è il numero totale di sinusoidi e $e(t)$ è la componente residuale. Il modello sinusoidale più un residuo assume che le sinusoidi costituiscano la parte periodica del suono, con lente e non significative variazioni di ampiezza e frequenza.

La parte residuale viene ottenuta sottraendo le sinusoidi dal suono originale. Una volta che la parte periodica del suono in ingresso è stata identifica e sottratta, la parte residuale può è essere considerata come un segnale stocastico, e analizzata a sua volta. E' possibile rappresentare il residuo tramite un modello di filtraggio del rumore. Per essere sintetizzato viene generato uno spettro in ogni frame partendo dall'inviluppo di magnitudine spettrale che nella parte di analisi ha approssimato la componente residuale. Un rumore bianco viene filtrato da inviluppi di frequenza che variano nel tempo. Il suono originale viene quindi ricostruito tramite sintesi additiva per la componente sinusoidale e tramite sintesi sottrattiva per la componente residuale.

Capitolo 3

Tecnologie utilizzate

Nel capitolo 2 la trattazione si è concentrata sulle basi storiche e teoriche della Sintesi per Modello Sinusoidale e nel 1 sono state illustrate le principali implementazioni note in letteratura. Ogni implementazione prescinde dall'utilizzo di più mezzi tecnologici, siano questi di tipo hardware o software.

Come espresso in [7], la Sintesi per Modello Sinusoidale è stata realizzata utilizzando una Lisp Machine workstation, su cui veniva eseguito del codice scritto in un dialetto del Lisp. Le conversioni A/D e D/A invece venivano effettuate tramite un Digital Signal Controller DSC-200.

La trattazione del presente capitolo è incentrata sulle tecnologie principali utilizzate per implementare la Sintesi per Modello Sinusoidale. Viene riservata una particolare attenzione agli strumenti software utilizzati, Max/MSP e Python, e viene brevemente illustrato l'ambiente hardware utilizzato.

3.1 Max/MSP

Come esposto in [5], Max/MSP è un ambiente di sviluppo grafico ideato da Miller Puckette alla fine degli anni '80. E' stato progettato per lo sviluppo di applicazioni musicali in real-time. Originariamente scritto per computer Apple, successivamente è stato implementato anche negli elaboratori NeXT, come parte dell'IRCAM Music Workstation. In origine era stato progettato per il MIDI, ma dopo successivi sviluppi, oggi permette di descrivere flussi sia di controllo che di segnale in un ambiente unificato.

Il concetto fondamentale che sta alla base del funzionamento di Max/MSP è la patch. Una patch è una collezione di scatole connesse da linee. Le scatole rappresentano degli oggetti, i quali possono ricevere dei messaggi in ingresso e mandare messaggi in uscita. Gli ingressi e le uscite delle scatole si chiamano "inlets" e "outlets".
I messaggi passano attraverso le linee o "patch cables", segmenti che connettono l'outlet di una scatola all'inlet di un'altra scatola. Ogni inlet e outlet può avere più patch cables ad esso collegati. Se coesistono più linee collegate ad una stessa uscita, il messaggio può raggiungere gli inlet di più di una scatola. Se invece più linee provenienti da outlets di scatole diverse si collegano all'inlet di un'unica scatola diversi messaggi possono entrare allo stesso ingresso.
I messaggi che vengono passati sono frequentemente numeri e "bangs", questi ultimi denotano convenzionalmente un evento che non ha parametri. Ma esistono anche altri messaggi che possono essere passati, ad esempio i caratteri e le liste.
Oltre allo scambio di messaggi tramite il meccanismo degli inlet/outlet, gli oggetti possono accedere al clock o al MIDI I/O.
Per modificare una patch bisogna assicurarsi che essa sia in modalità "edit", viceversa per osservarne l'operatività bisogna mettere la patch in modalità "run".
La figura 8 illustra le tipologie di boxes che si possono utilizzare in una patch.

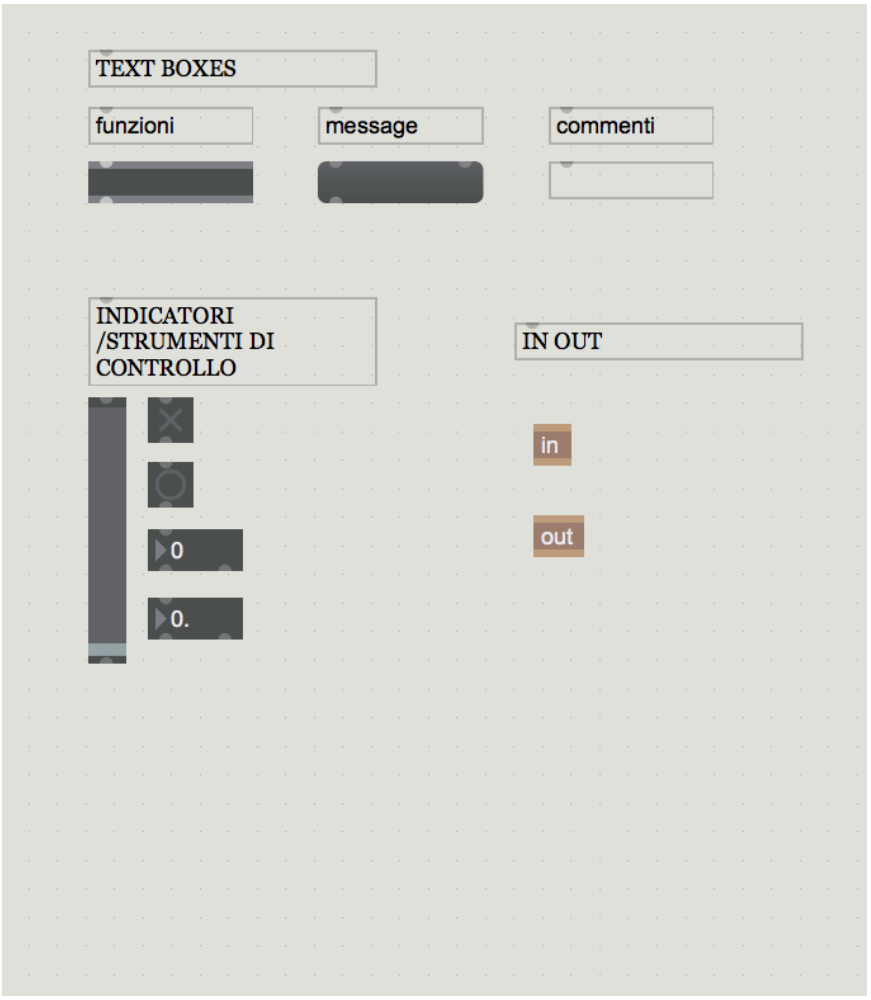

Figura 8: Tipologie di boxes che si usano nelle patch. Notare che in e out sono in rosso perché funzionano solamente all'interno di patch annidate.

Le scatole o boxes che si possono utilizzare in Max/MSP sono raggruppate, per funzionalità, in tre famiglie:

- I text boxes sono le scatole che contengono le classi di funzioni, i messaggi e i commenti
- Gli indicatori o strumenti di controllo sono i boxes che permettono il controllo di parametri. Fanno parte di quest gruppo i buttons, i toggles, gli sliders, le number box che possono contenere interi o numeri flottanti.
- Le scatole di comunicazione Input Output, che servono per mettere in comunicazioni più patch, quando queste sono annidate.

Il flusso di esecuzione delle operazioni all'interno di Max/MSP segue una convenzione: se una scatola, o oggetto ha più di un inlet, l'inlet più a sinistra è definito "attivo" o "caldo". Passare un messaggio a questo inlet causa qualche comportamento da parte dell'oggetto, mentre passare messaggi agli altri inlet sull'oggetto ne determinano un cambiamento di stato.
La figura 9 mostra due casi di funzionamento in cui un box corrispondente all'oggetto somma riceve nei suoi inlet due valori interi inseriti in due number box. Nel caso a) il risultato mostrato nella number box che riceve il valore del risultato dall'outlet della somma non è aggiornato perché l'ultimo valore che l'oggetto somma ha ricevuto è arrivato dall'inlet di destra, non causando nessuna operazione nell'oggetto somma. Nel caso b) il valore mostrato nella number box del risultato è aggiornato poiché il valore che arriva nell'inlet sinistro é l'ultimo in ordine temporale ad essere arrivato, ed ha causato l'oggetto somma a svolgere l'operazione.

Oggetti più complessi possono prendere in ingresso un'ampia varietà di messaggi diversi. Ad esempio un *sequencer* potrebbe prendere in ingresso messaggi di "start", "stop", "pause".
E' fondamentale che i messaggi arrivino a destinazione in un particolare ordine, al fine di un corretto funzionamento della patch. Da un'analisi della disposizione grafica degli oggetti in una patch si può ricavare l'ordine in cui gli eventi si svolgeranno. Se un oggetto manda un messaggio tramite più di un outlet per un singolo evento, i messaggi escono dall'outlet in ordine da destra a sinistra. Questo perché come si è visto, di solito la posizione più a sinistra in ingresso di un oggetto ne causa una determinata azione.

In Max/MSP l'elaborazione dei segnali viene effettuata attraverso il modulo MSP. MSP (Max Signal Processing) corrisponde ad una collezione di particolari oggetti, anche detti appartenenti alla classe "tilde" o classe "$\sim$". Gli oggetti appartenenti a questa classe eseguono operazioni di elaborazione dei segnali e comunicano tramite i

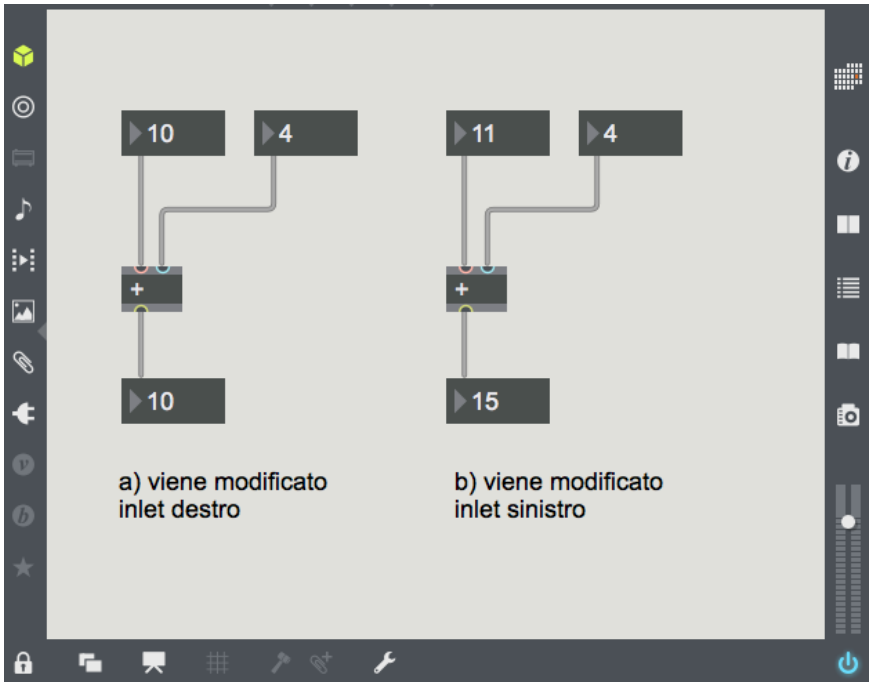

Figura 9: Esempio di funzionamento dell'ordine delle operazioni in Max/MSP.

messaggi "segnale" che passano continuativamente ad audio rate tra gli outlets e gli inlets.
Nella figura 10 viene mostrato un esempio in cui vengono utilizzati oggetti appartenenti alla classe tilde. Nella figura gli oggetti presenti sono *cycle*~, un semplice oscillatore che prende come argomento la frequenza, un *live.gain*~ che gestisce il volume di uscita del segnale e un *ezdac*~, un convertitore digitale-analogico che permette di far uscire il suono. Il display grafico dei cavi che collegano gli oggetti della classe ~ è diverso dal display grafico dei cavi che collegano gli oggetti comuni di Max/MSP visti in precedenza. Questo perché i segnali che passano all'interno dei cavi che collegano gli oggetti della classe tilde, a differenza degli altri, si muovono ad audio rate.

Oltre agli oggetti che vengono forniti nativamente da Max/MSP, è prevista la possibilità di programmare degli oggetti esteramente ed utilizzarli poi all'interno delle patch. Questi oggetti, denominati externals, sono solitamente scritti in linguaggio di programmazione C o C++. Una volta compilati, gli externals vengono inseriti in una cartella per poter essere utilizzati all'interno delle patch. La possibilità di avere funzionalità scritte dagli utenti permette di estendere ulteriormente le funzionalità di Max/MSP.

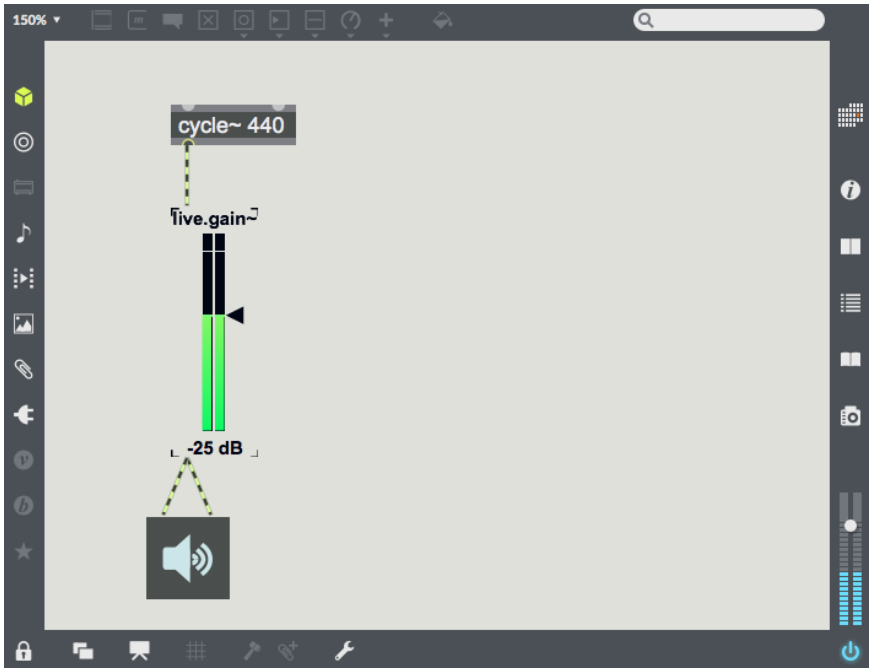

Figura 10: Esempio di oggetti del modulo MSP.

Durante lo svolgimento del progetto di ricerca è stato utilizzato l'oggetto *shell*. L'oggetto *shell* è un external scritto da Jeremy Bernstein e Bill Orcutt, scaricabile alla pagina https://github.com/jeremybernstein/shell e, compilato il codice sorgente, si ottengono due file, *shell.mxo* e *shell.maxhelp*.
L'oggetto *shell* implementa le funzionalità di un terminale. Nel contesto del presente progetto è stato utilizzato per eseguire all'interno delle patch Max/MSP alcuni file scritti in Python.
Nell'unico inlet dell'oggetto *shell* vengono passati dei messaggi contenenti i comandi che normalmente si utilizzano all'interno di un terminale. L'outlet sinistro permette di visualizzare i canali *standard error* e di*standard out*. L'outlet destro invece manda in uscita un messaggio di "bang" quando l'oggetto *shell* ha finito di elaborare le operazioni che gli sono state mandate in input.

Nell'immagine 11 è possibile osservare il funzionamento dell'oggetto *shell*. All'inlet viene passato un messaggio contenente un comando per l'interprete, in questo caso il comando "date". L'oggetto *shell* restituisce il risultato del comando tramite l'outlet sinistro. All'outlet sinistro è collegato una message box per la visualizzazione del contenuto output all'interno della patch e un oggetto *print* che permette di stampare il risultato sulla consolle di Max/MSP. L'oggetto *print* prevede la possibilità di ricevere come argomento un alias, che viene utilizzato nella consolle di Max/MSP per

precedere il messaggio. In questo caso si è scelto l'alias "stdout" A fine esecuzione, *shell* lancia un messaggio di "bang" tramite l'outlet destro. Questo comportamento si può osservare tramite un altro oggetto *print*, avente come alias "bang" che permette di stampare nella console di Max/MSP il messaggio bang ricevuto da *shell* a fine esecuzione.

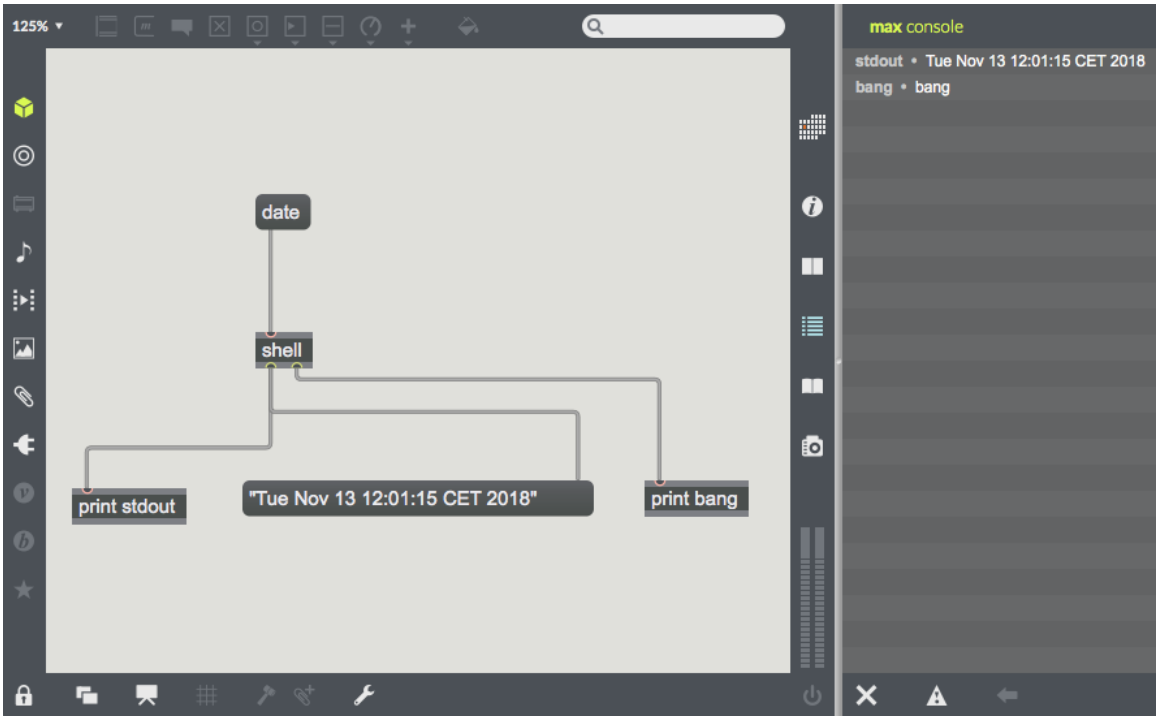

Figura 11: Funzionamento dell'oggetto external *shell*.

3.2 Python

Python è un linguaggio di programmazione concepito alla fine degli anni ottanta. L'implementazione è iniziata nel 1989 ad opera di Guido van Rossum, e la prima versione è stata rilasciata nel 1991. Successivamente, nel 2000 è stata realizzata la seconda versione. Nell'ambito del presente progetto è stata utilizzata la versione di Python 3.0, rilasciata nel 2008.
Nel sito http://www.python.it/about/ si legge che Python supporta diversi paradigmi di programmazione, come quello object-oriented, quello imperativo e quello funzionale, ed offre una tipizzazione dinamica forte.
Python è un linguaggio pseudocompilato: un interprete si occupa di analizzare il codice sorgente (semplici file testuali con estensione *.py*) e, se sintatticamente corretto,

di eseguirlo. In Python, non esiste una fase di compilazione separata (come avviene in C, per esempio) che generi un file eseguibile partendo dal sorgente. Python è inoltre un free software: non solo il download dell'interprete per la propria piattaforma, così come l'uso di Python nelle proprie applicazioni, è completamente gratuito; ma oltre a questo Python può essere liberamente modificato e così ridistribuito, secondo le regole di una licenza pienamente open-source.

Un aspetto che è risultato di grande importanza nell'utilizzo di Python è il concetto di modulo. Alla pagina https://docs.python.it/html/tut/node8.html viene data una descrizione di questa caratteristica. Python prevede la possibilità di scrivere in un file la definizione di uno script. Una volta salvato in estensione *.py* questo file, o modulo, può essere richiamato in altri file per far riferimento agli script in esso contenuti.
Un modulo è quindi un file che contiene definizioni e istruzioni Python. Il nome del file è il nome del modulo con il suffisso *.py* aggiunto. All'interno di un modulo, il nome del modulo è disponibile (sotto forma di stringa) come valore della variabile globale *_name_*.
Python viene fornito con una libreria di moduli standard, descritta in un documento separato. Alcuni moduli sono interni all'interprete ("built-in"). Forniscono supporto a operazioni che non fanno parte del nucleo del linguaggio ma sono comunque interne, per garantire efficienza o per fornire accesso alle primitive del sistema operativo, come chiamate di sistema. L'insieme di tali moduli è un'opzione di configurazione che dipende dalla piattaforma sottostante.
Un modulo particolare merita un po' di attenzione: *sys*, che è presente come modulo built-in in ogni interprete Python. Le variabili *sys.ps1* e *sys.ps2* definiscono le stringhe usate rispettivamente come prompt primario e secondario. La variabile *sys.path* è una lista di stringhe che determinano il percorso di ricerca dei moduli dell'interprete. Viene inizializzata con un percorso predefinito ottenuto dalla variabile di ambiente *PYTHONPATH*, o da un valore predefinito built-in se *PYTHONPATH* non è configurata.

I "package" sono un modo di strutturare lo spazio dei nomi dei moduli di Python usando nomi di modulo separati da punti. Per esempio, il nome del modulo A.B designa un sottomodulo chiamato "B" in un package chiamato "A". Proprio come l'uso dei moduli permette ad autori di moduli differenti di non doversi preoccupare dei nomi delle rispettive variabili globali, usare nomi di moduli separati da punti risparmia agli autori di package multi-modulari come NumPy o PIL (Python Imaging Library) ogni preoccupazione circa i nomi dei moduli.

Nello svolgimento della presente ricerca è stata utilizzata un'implementazione della Sintesi per Modello Sinusoidale realizzata in Python da Xavier Serra.
Alla pagina https://github.com/MTG/sms-tools Xavier Serra ha messo a disposizione gli "sms tools" un insieme di file, che comprendono per la maggior parte codice sorgente Python e, in minor parte, del codice scritto in C. Questo materiale ha costituito il punto di partenza e il nucleo principale da cui la ricerca ha avuto inizio.
Nel codice sorgente di Serra vengono utilizzati diversi moduli. E' risultato necessario scaricare ed installare ipython, numpy, matplotlib, scipy, and cython. Ipython è un'interprete di comandi interattivo, che permette all'utente di lanciare comandi Python. Numpy è un'estensione di Python che consente di operare con funzioni matematiche di alto livello, è un package fondamentale per calcoli matematici, che consente di utilizzare array n-dimensionali e, tra le altre funzioni, implementa la trasformata di Fourier. Matplotlib è una libreria che permette la gestione e la visualizzazione di grafici 2D. SciPy, similmente a Numpy consiste in un insieme di strumenti scientifici e matematici open source. Cython è un linguaggio di programmazione che permette facilmente di scrivere extensions in linguaggio C da utilizzare facilmente in Python. Permette una programmazione ad alto livello, orientata agli oggetti, funzionale e dinamica.
All'interno degli sms tools ci sono dei file audio di estensione wav che sono stati utilizzati per testare le funzionalità dell'applicativo sviluppato. Questi file audio provengono da http://www.freesound.org, un sito che ospita suoni con licenza Creative Commons Attribution 4.0. Con tale licenza, purché vengano dati i crediti al sito che li ospita, i suoni possono essere copiati, trasformati, e distribuiti tramite qualsiasi mezzo.

3.3 Interazione tra Max/MSP e Python

L'implementazione della Sintesi per Modello Sinusoidale, come esposto precedentemente, è stata effettuata tramite l'utilizzo congiunto dell'ambiente di sviluppo grafico Max/MSP e degli script Python.
Per far interagire questi due ambienti, l'external *shell* è risultato di fondamentale importanza.
Di seguito viene fornita un'esemplificazione dell'interazione dei due ambienti. Per testare l'effettiva operatività del meccanismo è stata realizzata una patch che implementa un semplice script Python. Lo script Python consta di una sola riga che effettua il comando:

```
1 print("hello␣world")
```

Questa riga è stata scritta in un editor di testo e salvata in estensione *.py*. All'interno di Max/MSP viene passato all'oggetto *shell* un messaggio contenente il comando di esecuzione dello script Python. Questo messaggio è composto dal path assoluto dell'applicazione Python 3.0, in questo caso `usr/local/bin/python3` e il path intero del file di script `/Users/brostiamo/Desktop/TESI3/helloworld.py` Una volta che il messaggio viene passato all'oggetto *shell*, il contenuto del file viene elaborato dall'interprete Python 3.0 e viene stampato su canale *standard output* il contenuto dell'argomento di *print*. Il contenuto esce quindi dall'outlet sinistro di *shell* e viene visualizzato all'interno di Max/MSP, sia nella consolle di Max/MSP, sia in una message box all'interno della patch. Alla fine dell'esecuzione dall'outlet destro di *shell* esce un "bang". La figura 12 mostra la patch Max/MSP.

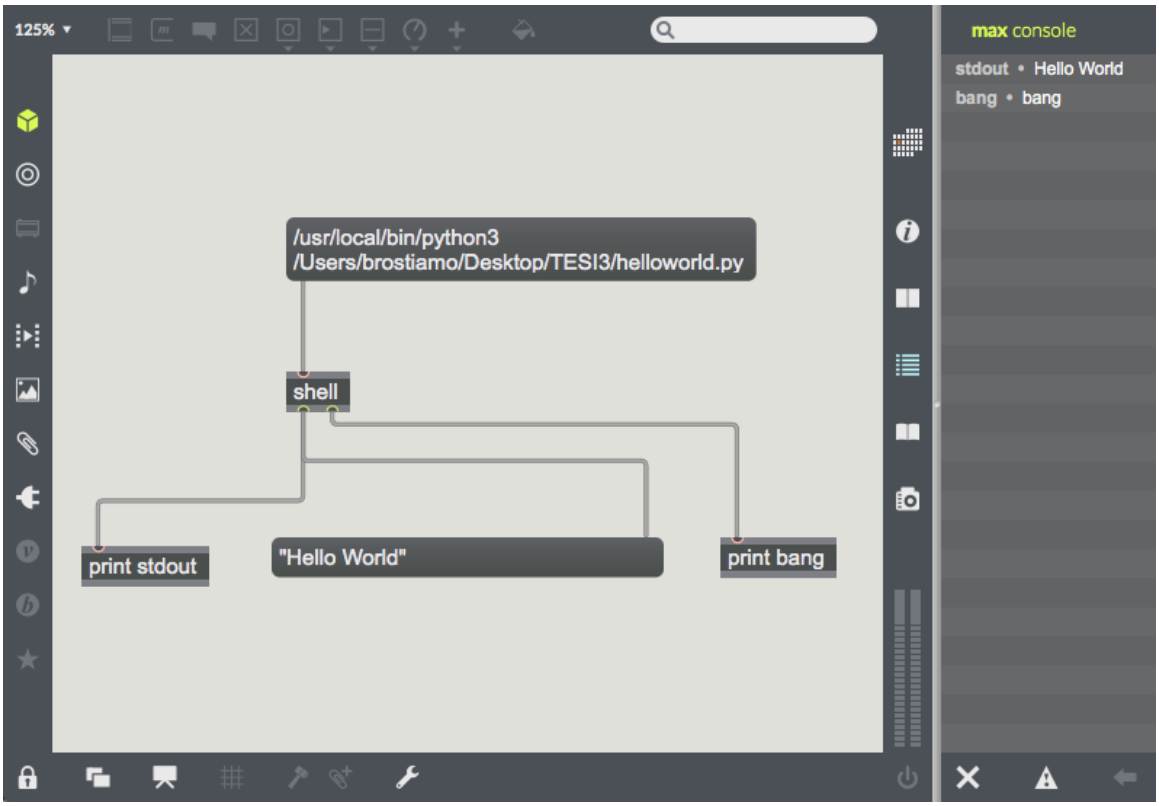

Figura 12: In figura viene illustrato il contenuto della patch creata per testare l'interazione tra Max/MSP e Python.

Capitolo 4

Caso di studio

Il presente capitolo ha lo scopo di illustrare le varie fasi che hanno composto la realizzazione del progetto di ricerca. Come precedentemente esposto, il progetto ha coinvolto gli ambienti di sviluppo Max/MSP e Python al fine di implementare la Sintesi per Modello Sinusoidale. Il capitolo incomincia quindi con una panoramica dei metodi impiegati, esponendo le problematiche incontrate nello sviluppo e le soluzioni che sono state adottate, procedendo successivamente con un'illustrazione più dettagliata del lavoro svolto. Dove necessario viene inserito il codice Python e parte delle patch Max/MSP che sono state realizzate.

4.1 Panoramica

Il risultato del percorso di ricerca intrapreso al fine di implementare la Sintesi per Modello Sinusoidale consiste in tre schemi di lavoro alternativi e diversificati tra loro. Ognuno di questi schemi implementa la Sintesi per Modello Sinusoidale e coinvolge gli ambienti Max/MSP e Python. In questa sezione viene fornita una panoramica del lavoro svolto. Nel seguito del capitolo i tre schemi vengono presi singolarmente in esame, illustrandone le eventuali problematiche, i punti di forza e le debolezze.

Ciascuna delle tre istanze di cui si compone il risultato comprende una o più patch Max/MSP associate e una parte di codice Python diviso in più file. I file contenente codice Python si sono diversificati a partire dal codice originariamente scritto da Xavier Serra, sviluppandosi in relazione all'idea di lavoro retrostante a ciascuna delle tre istanze.

- Il primo metodo sviluppato nel lavoro di ricerca prevede l'utilizzo di una patch Max/MSP per eseguire un applicativo Python completo di interfaccia grafica, che sviluppa l'analisi e la sintesi. Tramite l'interfaccia grafica vengono inseriti i

parametri di input necessari e l'applicativo compie in un unico passo sia l'analisi che la sintesi, utilizzando il modello sinusoidale più una parte stocastica. Questo schema implementativo permette di utilizzare l'applicativo sviluppato da Xavier Serra all'interno Max/MSP. Questa istanza rappresenta perciò l'implementazione più fedele al lavoro originale di Xavier Serra, rendendo utilizzabile la Sintesi per Modello Sinusoidale all'interno di Max/MSP.

- Il secondo schema utilizza una patch Max/MSP per eseguire un applicativo Python che effettua l'analisi usando il modello sinusoidale più una parte stocastica. Il risultato ottenuto dall'analisi viene utilizzato in una seconda patch Max/MSP per essere sintetizzato. Quest'implementazione si differenzia dalla precedente nel fatto che la sintesi è stata realizzata completamente in ambiente Max/MSP. Pur cercando di rimanere fedele all'algoritmo di sintesi di Serra, questa implementazione testimonia una deviazione dal lavoro di Serra, verso un percorso inedito di sperimentazione. La motivazione è l'opportunità di creare un caso di studio, che lascia spazio ad ampi margini di miglioramento.

- Il terzo schema di lavoro, in modo simile al precedente, da cui ha avuto inizialmente origine, utilizza Python per eseguire l'analisi ed effettua la sintesi in Max/MSP. Si differenzia dal secondo schema per la tipologia di file di interscambio generato per passare i risultati dell'analisi in Python, e per il percorso di sperimentazione intrapreso nell'implementare la sintesi in Max/MSP. Studiare una strada alternativa dallo schema precedente al fine di implementare la sintesi in Max/MSP ha portato ad un caso di studio che, pur conservando ampi margini di miglioramento, si è dimostrato più performante del precedente.

Diversi elementi accomunano i tre schemi di lavoro. Uno di questi elementi è costituito dalla tipologia parametri che l'utente inserisce per utilizzare gli applicativi. Come esposto precedentemente, la Sintesi per Modello Sinusoidale lavora su un file in ingresso il quale viene in una prima fase analizzato e successivamente sintetizzato. Oltre al file in ingresso, l'utente deve specificare un insieme di parametri che servono all'algoritmo per analizzare il segnale sonoro. Questi parametri rivestono un ruolo fondamentale nella qualità di analisi che viene prodotta. Nei tre schemi di lavoro presentati è previsto che l'utente inserisca questi parametri necessari. Tali parametri sono:

- Il nome file che deve essere analizzato con il percorso nel file system dove risiede il file. Il file deve essere rigorosamente monofonico e a 44100 di frequenza di campionamento.

- La tipologia di finestra di analisi. L'utente può scegliere tra rettangolare, Hanning, Hamming, Blackman e Blackman-Harris.

- La dimensione della finestra di analisi, solitamente è 2001.
- La dimensione della FFT di analisi. Il valore della dimensione della FFT deve essere una potenza di 2 e maggiore della dimensione della finestra di analisi.
- La soglia minima di magnitudine che una stimato picco deve superare per poter essere considerato come possibile picco appartenente ad una traccia sinusoidale.
- La durata minima di una traccia sinusoidale. Sotto questa soglia temporale eventuali picchi non saranno ricondotti ad una traccia sinusoidale.
- Il numero massimo di sinusoidi parallele che possono essere trovate.
- La deviazione massima in frequenza che un picco può avere per essere considerato appartenente alla stessa traccia sinusoidale. Se viene superata questa soglia di deviazione, calcolata in base alla frequenza 0 e poi trasportata in proporzione per tutto lo spettro, il picco non viene ricondotto a quella traccia sinusoidale.
- La pendenza della deviazione dalla frequenza, calcolata come funzione della frequenza.
- Il fattore di approssimazione stocastico.

Impostati tali parametri si può procedere con l'algoritmo di analisi. Questo algoritmo, da un punto di vista di implementazione, risulta essere lo stesso per ognuno dei tre schemi di lavoro presentati. Riguardo all'insieme di operazioni che coinvolgono l'algoritmo di analisi, la principale differenza tra i tre schemi di lavoro consiste nel fatto che il primo schema di lavoro, una volta computata l'analisi, esegue automaticamente l'algoritmo di sintesi. Per l'utente l'operazione di analisi/sintesi risulta essere atomica, ossia viene eseguito con un unico comando sia l'analisi, che la sintesi. Per il secondo e per il terzo modello di lavoro invece l'analisi produce dei file output che vengono utilizzati per effettuare la sintesi in maniera separata e sotto esplicito comando dell'utente.

Un terzo elemento da tenere in considerazione tra le similarità rispetto alle tre implementazioni, è l'utilizzo in Max/MSP dell'external *shell*. Nel capitolo 3 è stato discusso l'utilizzo di questo oggetto, sviluppato per implementare le funzionalità di un terminale in Max/MSP. Nell'ambito del presente progetto l'oggetto *shell* è stato utilizzato per eseguire gli script Python e costituisce di fatto, il punto di congiunzione dell'ambiente Python dentro Max/MSP. Come precedentemente esposto, l'oggetto *shell* utilizza l'outlet sinistro come canale standard output. Dalle impostazioni dell'oggetto è possibile unire al canale standard output anche il canale *standard error*. Gli errori nell'applicativo Python, vengono così stampati all'interno dell'ambiente

Max/MSP, rendendo più agile il debugging.

4.2 Primo caso di studio

Il primo caso di studio sviluppato permette di utilizzare in ambiente Max/MSP l'applicativo Python che implementa la Sintesi per Modello Sinusoidale.

Il risultato proposto in questa sezione si compone di:

- Una patch Max/MSP che lancia l'interfaccia grafica Python per inserire i parametri di analisi.
- Una collezione di file Python che implementano l'interfaccia grafica, l'analisi e la sintesi.

4.2.1 Max/MSP: patch di avvio

L'esecuzione degli script Python sono gestiti da una patch Max/MSP. Nella figura 13, che mostra il contenuto della patch, ci sono due message box collegate all'oggetto external *shell*. Il messaggio a sinistra è il messaggio che permette di eseguire gli script Python. Il comando per terminale che lancia un applicativo Python è infatti composto da due parti: la versione installata di Python, in questo caso la 3.0, e il nome del file che contiene l'applicativo Python che deve essere eseguito. Il messaggio di sinistra contenuto in una message box è quindi composto in due parti, la prima parte è il path assoluto di Python 3.0 la seconda parte invece indica il path assoluto dell'applicativo Python. Con un click di mouse sulla message box il contenuto del messaggio viene passato all'oggetto *shell* che esegue l'applicativo Python.
La seconda message box collegata all'oggetto *shell* contiene il messaggio "pkill". Passare questo messaggio a *shell*, esattamente come ad un terminale, forza la chiusura di qualsiasi processo sia in quel momento in esecuzione. Questo comando è stato inserito nel caso in cui si verificasse la necessità di forzare la chiusura dell'applicativo Python. L'outlet di sinistra è collegato ad un oggetto *print* che stampa nella console di Max/MSP sia i risultati che escono nel canale *standard output*, sia i risultati che escono nel canale *standard error*. Tali risultati e le eventuali anomalie di comportamento vengono perciò monitorate all'interno della patch Max/MSP. L'outlet di destra invece è collegato ad un oggetto *print* che stampa il messaggio "bang" una volta che l'oggetto *shell* ha terminato di svolgere tutte le operazioni.

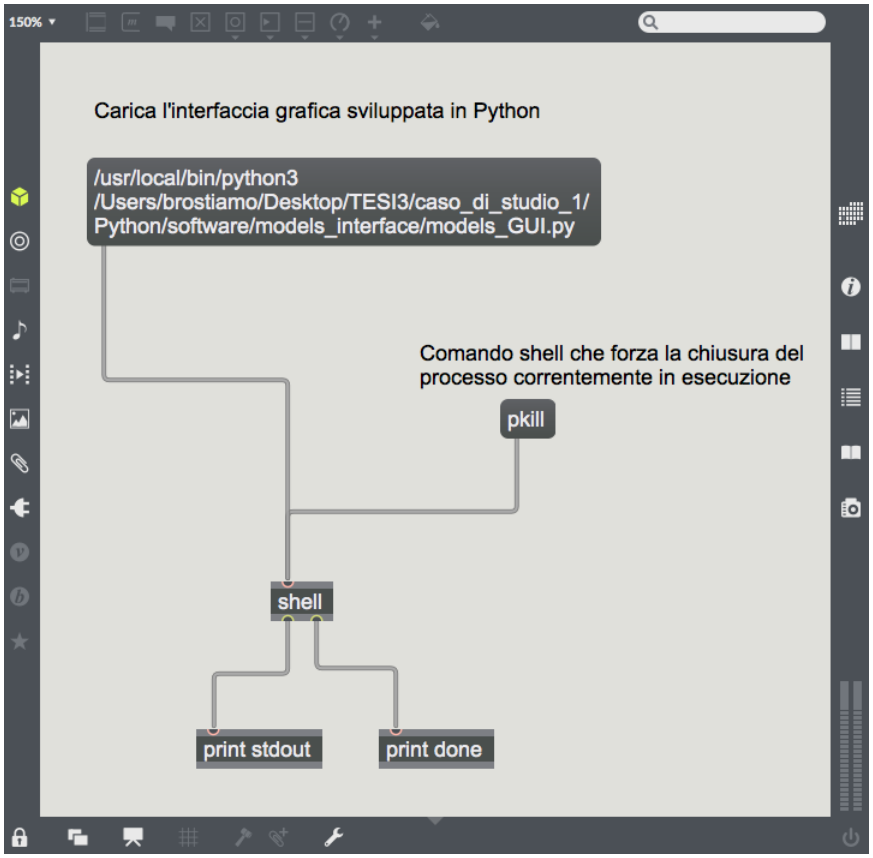

Figura 13: La figura mostra la patch Max/MSP principale.

4.2.2 Python: interfaccia grafica

Come precedentemente esposto, per utilizzare l'applicativo è necessario passare il contenuto della message box di sinistra all'oggetto *shell*. Per compiere tale operazione occorre fare un click sulla message box. Una volta che il contenuto del messaggio viene passato, l'oggetto *shell* lancia Python versione 3.0 per eseguire l'applicativo. L'ambiente di esecuzione si sposta quindi su Python, e su un interfaccia grafica sviluppata originariamente da Xavier Serra. La figura 14 illustra l'interfaccia grafica. Tramite questa interfaccia grafica si inseriscono i parametri di ingresso che occorrono per svolgere l'analisi. Una volta inseriti i parametri si può far eseguire l'algoritmo di analisi e sintesi tramite il pulsante *Compute*.

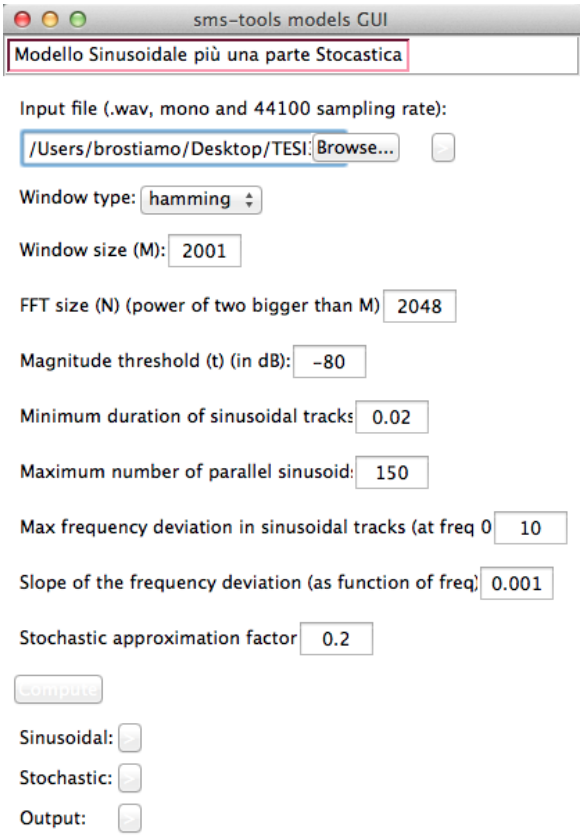

Figura 14: Interfaccia grafica sviluppata in Python.

Il pulsante *Compute* una volta che viene selezionato fa eseguire la funzione *compute_model*. Questa parte di codice Python si occupa di salvare i parametri indicati in input dall'utente all'interno di alcune variabili. Le variabili vengono successivamente passate alla funzione *spsModel_function.main*. Viene qui riportato il codice di riferimento per questa parte:

```
1 def compute_model(self):
2
3         try:
4                 inputFile = self.filelocation.get()
5                 window = self.w_type.get()
6                 M = int(self.M.get())
7                 N = int(self.N.get())
8                 t = int(self.t.get())
```

```
9                   minSineDur = float(self.minSineDur.get())
10                  maxnSines = int(self.maxnSines.get())
11                  freqDevOffset = int(self.freqDevOffset.get())
12                  freqDevSlope = float(self.freqDevSlope.get())
13                  stocf = float(self.stocf.get())
14
15                  spsModel_function.main(inputFile, window, M, N
                        , t, minSineDur, maxnSines, freqDevOffset,
                        freqDevSlope, stocf)
16
17          except ValueError as errorMessage:
18                  tkMessageBox.showerror("Input␣values␣error",
                        errorMessage)
```

4.2.3 Python: analisi

La funzione spsModel_function.main, che viene eseguita nella parte finale del codice, prende come argomento i parametri che sono stati precedentemente impostati dall'utente, e svolge l'analisi e la sintesi. Il codice che si riferisce alla funzione spsModel_function.main funzione risiede in un altro file Python, il quale viene chiamato ad esecuzione.

Entrando nel dettaglio implementativo, la funzione *spsModel_function.main* dichiara la dimensione della FFT utilizzata per fare la sintesi, con un valore di 512 samples. Successivamente viene stabilita la dimensione del salto di hop che, come è stato osservato nel capitolo 2, determina il fattore di overlap. Tale valore, come la dimensione della FFT utilizzata per la sintesi, è fissato in partenza. Il valore fissato è un quarto del valore di FFT per la sintesi, ossia 128. Tramite una funzione importata da un ulteriore package esterno chiamato *Util Function*, viene letto il file *.wav*, passato nell'argomento *inputFile*. La funzione *wavread* restituisce la frequenza di campionamento in una variabile chiamata *fs* e il file audio scalato in valori tra -1 e 1 e convertito in valori di ampiezza floating point, in un array chiamato x. Il codice procede calcolando la finestra di analisi partendo dal valore M che indica la dimensione e dal tipo di finestra, entrambi argomento della funzione *main*. Successivamente viene eseguita la porzione di codice che implementa l'analisi secondo il modello sinusoidale più una parte stocastica.

```
1 def main(inputFile='/Users/SMSuser/sounds/bendir.wav', window=
      'hamming', M=2001, N=2048, t=-80, minSineDur=0.02,
```

```
2         maxnSines=150, freqDevOffset=10, freqDevSlope=0.001,
             stocf=0.2):
3
4         Ns = 512
5         H = 128
6
7         (fs, x) = UF.wavread(inputFile)
8
9         w = get_window(window, M)
10
11        tfreq, tmag, tphase, stocEnv = SPS.spsModelAnal(x, fs,
              w, N, H, t, minSineDur, maxnSines, freqDevOffset,
             freqDevSlope, stocf)
```

La funzione che implementa l'analisi, chiamata in un ulteriore file Python esterno, prende come argomenti l'array x, la frequenza di campionamento, la finestra di analisi, la dimensione della FFT, il fattore di overlap, il valore di soglia espresso in decibel negativi, il minimo di durata di una sinusoide, il massimo di sinusoidi parallele, la deviazione massima in frequenza che un picco può avere per essere considerato appartenente alla stessa traccia sinusoidale, la pendenza della deviazione dalla frequenza, il fattore di approssimazione stocastico. Con questi valori in ingresso la funzione di analisi calcola per prima cosa l'analisi sinusoidale, ottenendo tre array che indicano la frequenza, la magnitudine e la fase di ciascuna traccia sinusoidale. Le tracce sinusoidali vengono quindi sottratte al suono di partenza e sul file risultato viene effettuata un'analisi per determinare il modello stocastico della parte residuale.

4.2.4 Python: sintesi

```
1         y, ys, yst = SPS.spsModelSynth(tfreq, tmag, tphase,
             stocEnv, Ns, H, fs)
2
3         outputFileSines = '/Users/SMSuser/output_sounds/' + os
             .path.basename(inputFile)[:-4] + '_spsModel_sines.
             wav'
4         outputFileStochastic = '/Users/SMSuser/output_sounds/'
              + os.path.basename(inputFile)[:-4] + '
             _spsModel_stochastic.wav'
5         outputFile = '/Users/SMSuser/output_sounds/' + os.path
             .basename(inputFile)[:-4] + '_spsModel.wav'
```

```
6
7           UF.wavwrite(ys, fs, outputFileSines)
8           UF.wavwrite(yst, fs, outputFileStochastic)
9           UF.wavwrite(y, fs, outputFile)
```

La funzione che implementa l'analisi restituisce quindi quattro array relativi a frequenza, magnitudine e fase delle tracce sinusoidale e al modello stocastico della parte residuale. Questi quattro array, insieme alla dimensione della finestra FFT di sintesi, al valore di overlap e alla frequenza di campionamento del file iniziale, vengono passati alla funzione che implementa la sintesi come argomenti. Tale funzione sintetizza prima la parte periodica, ossia le sinusoidi, successivamente calcola il contenuto frequenziale della parte stocastica e somma il risultato. La funzione di sintesi restituisce la somma generale in un array chiamato *y*, la parte sinusoidale in un array chiamato *ys* e relativa al modello stocastico in un terzo array chiamato *yst*. Questi array vengono scritti su tre file *.wav* e tramite l'interfaccia grafica, si possono ascoltare separatamente. Nel seguito viene mostrato il codice che implementa quanto appena esposto.

4.3 Secondo caso di studio

Il precedente caso di studio risulta essere fedele all'implementazione originale di Xavier Serra, ma, nella realizzazione dello schema, la quasi totalità dell'algoritmo è stato realizzato in Python, lasciando poco spazio all'utilizzo di Max/MSP. La presente implementazione, e quella successivamente esposta, testimoniano gli studi e gli sforzi al fine di elaborare delle versioni alternative a quella appena presentata, che coinvolgano in maniera più strutturale l'ambiente Max/MSP.

Il risultato proposto in questa sezione si compone di:

- Una patch Max/MSP per raccogliere i parametri di analisi.
- Una collezione di file Python che esegue l'analisi e scrive il risultato su dei file di testo.
- Una patch Max/MSP che implementa la sintesi.

4.3.1 Max/MSP: patch di avvio

La prima parte dello schema si occupa di elaborare l'analisi di un file *.wav* secondo il modello sinusoidale più una componente residuale. L'utente specifica i parametri

all'interno di una patch Max/MSP, il cui funzionamento prevede che i valori vengano raccolti e passati all'external *shell*. Nella figura 15 è rappresentata la patch Max/MSP che colleziona i parametri di analisi. Per alcuni dei parametri che l'utente inserisce è previsto il suggerimento di un valore di default. La tipologia di parametri risulta essere la stessa esposta per il caso di studio precedente. Tramite un click sul bang vicino al commento "clear", vi è la possibilità di pulire i campi dai valori che sono stati inseriti e mandare un messaggio "pkill" all'oggetto *shell*, per forzare la chiusura di ogni processo in corso.

Max/MSP prevede due modalità di visualizzazione di una patch. La modalità presentata fino al presente paragrafo è definita modalità "patching". Vi è una seconda modalità che è definita modalità "presentazione". Quest'ultima, esposta in figura 15, permette di modellare la patch sotto un profilo grafico. Alcuni elementi della modalità patching vengono nascosti, come ad esempio i cavi, ed è consentito selezionare un insieme di elementi affinchè vengano visualizzati nella modalità presentazione. Tale modalità permette quindi di organizzare la patch in una maniera graficamente ordinata, al fine di predisporre l'utente alla comprensione del funzionamento e all'utilizzo, della patch.

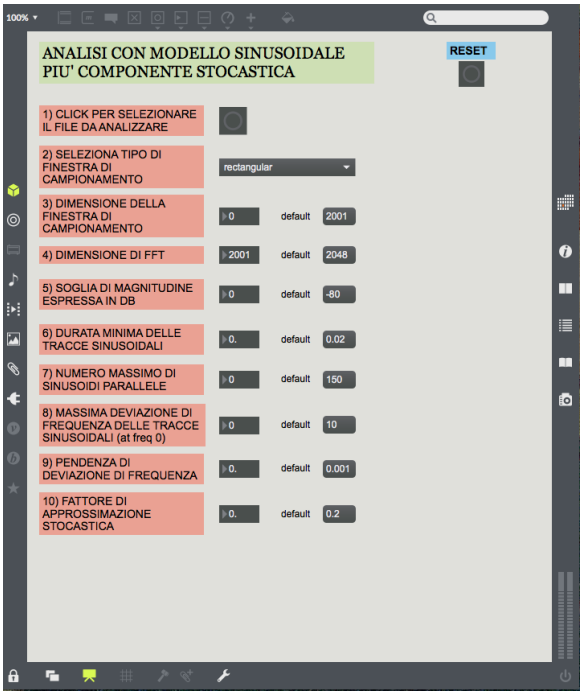

Figura 15: Patch principale di analisi in modalità "presentazione".

Per esaminare il funzionamento della patch di avvio è necessario osservare il contenuto della patch Max/MSP in modalità patching. Nell'immagine 16 si può notare la presenza di tutti gli oggetti riscontrati nella stessa patch, in modalità presentazione, oltre ad ulteriori oggetti, nascosti nella modalità presentazione. Il funzionamento della patch, che rimane lo stesso in entrambe le modalità, prevede che ogni parametro inserito dall'utente costruisca gradualmente una stringa di valori, tramite l'utilizzo del messaggio "append", che consente di aggiungere caratteri ad un messaggio. Una volta specificati, la stringa completa di tutti i parametri viene aggiunta alla stringa contenente il nome del programma, Python versione 3.0, e il nome dell'applicativo che l'oggetto *shell* deve eseguire. L'oggetto *shell* esegue quindi tramite Python 3, il file *.py* dove è contenuto il codice che implementa l'analisi, con la stringa di valori creata sugli ingressi dell'utente, come argomenti di linea di comando.

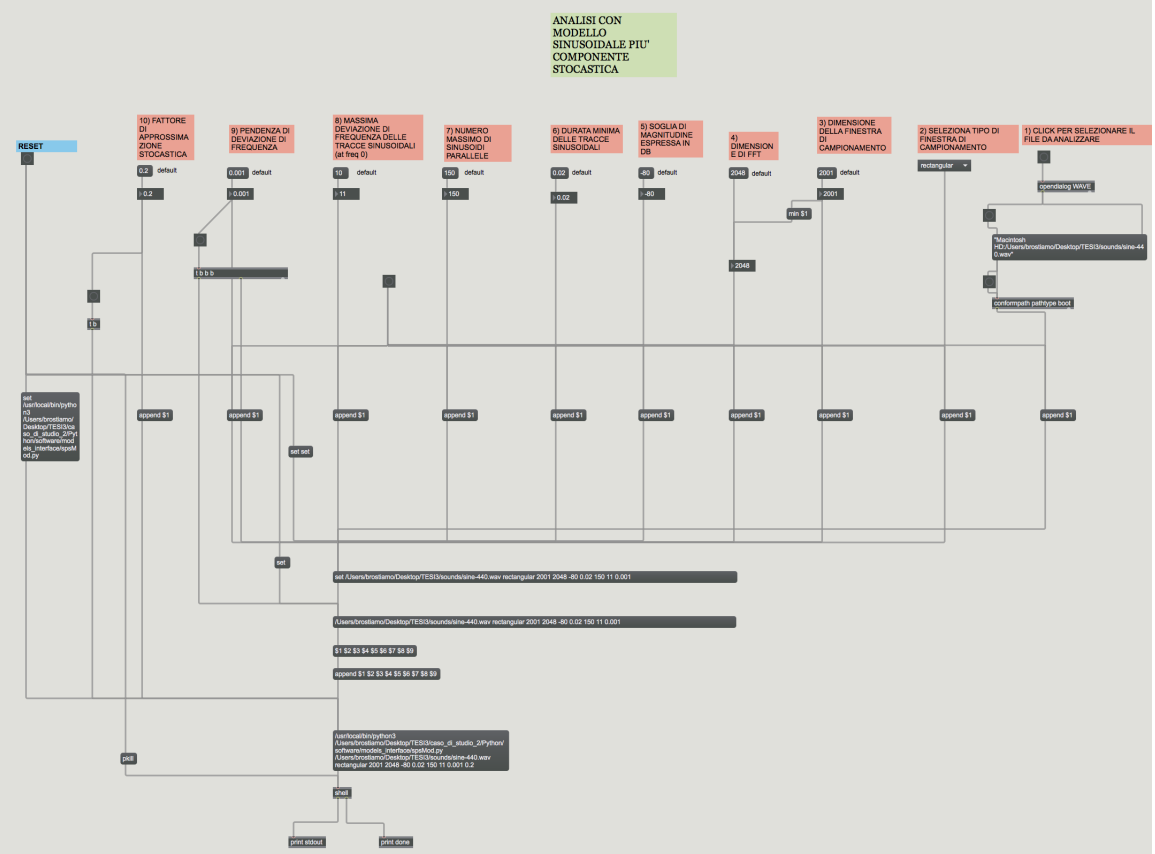

Figura 16: patch principale di analisi in modalità "patching".

4.3.2 Python: analisi e generazione del file di interscambio

Quando l'external *shell* manda in esecuzione il sorgente Python, viene eseguita la seguente porzione di codice:

```
1  nome_script, primo, secondo, terzo, quarto, quinto, sesto,
       settimo, ottavo, nono, decimo = sys.argv
2
3  inputFile = primo
4  print (inputFile)
5  window = secondo
6  print (window)
7  M = int(terzo)
8  print (M)
9  N = int(quarto)
10 print (N)
11 t = int(quinto)
12 print (t)
13 minSineDur = float(sesto)
14 print (minSineDur)
15 maxnSines = int(settimo)
16 print (maxnSines)
17 freqDevOffset = int(ottavo)
18 print (freqDevOffset)
19 freqDevSlope = float(nono)
20 print (freqDevSlope)
21 stocf = float(decimo)
22 print (stocf)
23
24 spsModel_function.main(inputFile, window, M, N, t, minSineDur,
       maxnSines, freqDevOffset, freqDevSlope, stocf)
```

La prima riga di codice recupera i valori passati da riga di comando scrivendoli in alcune variabili. Nelle righe di codice successivo i valori vengono salvati nelle variabili di analisi. Le variabili contenenti i valori dei parametri di analisi vengono quindi passati come argomenti della funzione *spsModel_function.main*. Come nel caso di studio precedente, la funzione *spsModel_function.main* risiede in un altro file *.py*, che viene chiamato ad esecuzione. Viene successivamente mostrato il codice relativo a questo schema di esecuzione.

```
Ns = 512
H = 128
(fs, x) = UF.wavread(inputFile)
w = get_window(window, M)
tfreq, tmag, tphase, stocEnv = SPS.spsModelAnal(x, fs, w, N, H
    , t, minSineDur, maxnSines, freqDevOffset, freqDevSlope,
    stocf)

numrows = tfreq.shape[0]
numcol = tfreq.shape[1]

for i in range (0, numrows):
        for j in range (0, numcol):
                if tfreq[i,j] == 0:
                        tmag[i,j] = 0
                else:
                        tmag[i,j] = 10**(tmag[i,j]/20)

saveToFile("/Users/SMSuser/caso_di_studio_2/Max/freqColl.txt",
    tfreq)
saveToFile("/Users/SMSuser/caso_di_studio_2/MAX/magColl.txt",
    tmag)
saveToFile("/Users/SMSuser/caso_di_studio_2/MAX/phasColl.txt",
    tphase)
saveToFile("/Users/SMSuser/caso_di_studio_2/MAX/stocEnv.txt",
    stocEnv)
```

Il codice riguardante la funzione che svolge l'analisi è il medesimo esposto nel caso precedente. Come output dell'analisi vengono generati quattro array, *tfreq*, *tmag*, *tpahse*, e *stocEnv*, relativi a frequenza, magnitudine, fase delle sinusoidi, e al modello stocastico. I valori presenti nell'array riguardante la magnitudine vengono convertiti in valori di ampiezze lineari. Una volta effettuata l'analisi e convertiti i valori di magnitudine in scala lineare, il codice Python, a differenza del caso di studio precedentemente illustrato, non prevede che venga effettuata la sintesi. Per passare i valori di analisi dall'ambiente Python a Max/MSP vengono utilizzati dei file di interscambio. Dal momento che l'oggetto Max/MSP che si occupa di recuperare i valori all'interno della patch lavora con i file di testo, i valori contenuti nei quattro array vengono scritti su quattro file di testo separati. Gli array di frequenza, magnitudine e fase sono caratterizzati da tante righe quanti sono i frame temporali e tante colonne quanto il valore specificato dall'utente con il parametro *maxnSines*. L'array relativo

al contenuto stocastico invece divide lo spettro in un insieme di bande ed assegna a ciascuna banda un peso. Nell'operazione di salvataggio del contenuto degli array nel file di testo, ogni riga viene numerata e i file generati vengono rinominati *freqColl.txt*, *magColl.txt* e *phasColl.txt*.

```
1, 439.7906287956646 0.0 0.0 0.0(…);
2, 440.4952139786010 0.0 0.0 0.0(…);
3, 440.1332941540446 0.0 0.0 0.0(…);
4, 440.5507703398360 0.0 0.0 0.0(…);
5, 440.6247755038769 0.0 0.0 0.0(…);
```

Figura 17: Risultato di analisi dei primi 5 frame temporali di una sinusoide con frequenza 440. All'algoritmo è stato impostato come parametro "maxnSines" il valore 150.

Una volta stampati i valori nei quattro file di testo, l'esecuzione dello script Python termina e il funzionamento dello schema prosegue sulla patch Max/MSP adibita alla sintesi.

4.3.3 Max/MSP: Patch di sintesi

La patch Max/MSP mostrata nella figura 18 si occupa di effettuare la sintesi utilizzando l'oggetto *pfft~*. *Pfft~* è uno degli oggetti che il modulo MSP offre per controllare l'elaborazione spettrale all'interno dell'ambiente Max. L'oggetto *pfft~* prende come argomento obbligatorio il nome di una patch annidata, o subpatch e, tra gli argomenti opzionali vi sono la dimensione della FFT e il fattore di sovrapposizione. Il numero di inlet ed outlet dell'oggetto può essere determinato dal numero di *fftin~* e *fftout~* presenti nella subpatch. E' infatti all'interno della subpatch che si svolge la parte di elaborazione spettrale. Mentre l'oggetto *pfft~* controlla la dimensione della finestra e l'overlap di un segnale in ingresso, *fftin~* applica la funzione finestra effettuando la FFT. L'oggetto *fftout~* realizza, al contrario, la IFFT. Una volta attivato l'audio, l'oggetto *pfft~* inizia l'attività computazionale e la subpatch performa l'algoritmo di sintesi. Il nucleo delle operazioni principali si svolge quindi all'interno della subpatch. Si può notare come l'oggetto *pfft~* abbia tre inlet, corrispondenti a tre oggetti di input all'interno della subpatch. Con un messaggio di "bang" al secondo inlet viene azzerato il contatore che si riferisce ai frame temporali, come verrà spiegato successivamente. Con un messaggio di "bang" al terzo inlet invece si forza la rilettura dei

file di interscambio. Il risultato del lavoro svolto dalla subpatch viene passato all'oggetto *live.gain∼* che gestisce il volume di uscita. L'output passa in fine attraverso un *ezadac∼*, il convertitore digitale-analogico per emettere il segnale sonoro.

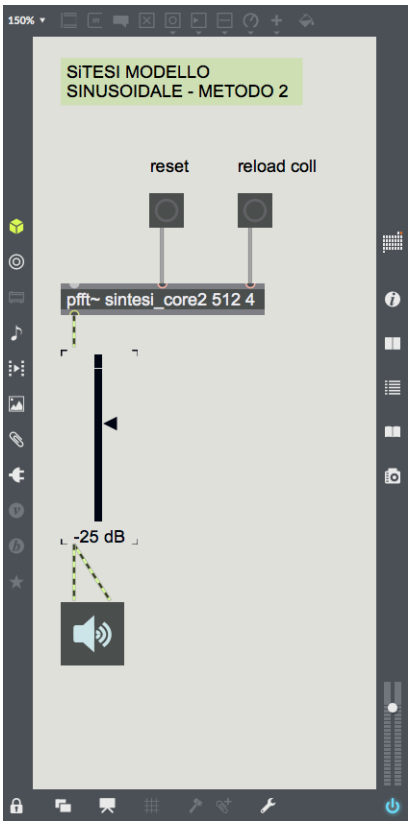

Figura 18: patch di sintesi.

In figura 19 viene mostrata la subpatch utilizzata dall'oggetto *pfft∼*, nella patch di livello superiore, per implementare la sintesi. La subpatch utilizza i valori contenuti nei file di interscambio per sintetizzare il suono di partenza, lavorando su un frame temporale alla volta. I valori riguardanti i frame temporali sono contenuti in ciascuna riga dei file "freqColl", "magColl" e "phasColl". Appena lanciata la subpatch, vengono allocati due buffer, mag e phase, di dimensione pari alla dimensione della FFT. Dal terzo outlet dell'oggetto *fftin∼* esce l'indice del bin corrente, con valori che vanno da 0 alla dimensione della FFT. Il passaggio dall'ultimo bin al bin numero 0 determina il passaggio da un frame temporale al successivo. Il passaggio da un frame temporale al successivo viene monitorato tramite l'oggetto *edge∼*. In ingresso ad edge∼ entrano

gli indici di bin e ogni volta che viene rilevato un passaggio da un numero diverso da 0 a 0, *edge~* manda fuori un "bang". Quindi all'inizio di ogni frame temporale viene lanciato un "bang" che mette in funzione un oggetto *trigger*. Questo oggetto manda in catena due messaggi di "bang". Il primo tra i due messaggi di "bang" che esce, quello di destra, pulisce il contenuto dei buffer "mag" e "phase". Il secondo "bang" in uscita dall'oggetto *trigger*, entra in un oggetto *counter*. Grazie a questo oggetto si tiene conto del numero di frame temporale corrente, che esce dall'outlet più a sinistra. Il numero di frame temporale corrente entra in un altro oggetto *trigger*, che fa uscire in successione il valore del contatore agli ingressi degli oggetti *coll*. L'oggetto *coll* permette di salvare e modificare una collezione di dati. Nel presente contesto, *coll* è stato utilizzato per leggere all'interno dei file di testo generati precedentemente da Python. In questo modo gli oggetti *coll* mandano fuori come output il contenuto della riga corrispondente al frame temporale corrente. I valore di fase e ampiezza entrano all'interno di due rispettivi oggetti *zl*, oggetti che gestiscono liste di elementi. Con l'argomento "queue" passato a *zl* viene implementato un meccanismo di accodamento FIFO, per cui ogni elemento della riga che entra all'interno di *zl* viene fatto uscire, tramite un messaggio di "bang" passato a *zl*, in ordine progressivo. Il valore di frequenza, prima di entrare dentro a *zl* convertito in indice di bin calcolando la divisione arrotondata ad intero con la risoluzione frequenziale. La risoluzione frequenziale si ottiene dall'oggetto *p getres*, un incapsulamento di un'insieme di oggetti, che si può osservare in figura 20. Un incapsulamento consiste nella creazione di una subpatch che continene un insieme di oggetti, precedentemente appartenenti alla patch principale. La risoluzione frequenziale si ottiene dalla divisione della frequenza di campionamento per la dimensione dei frame spettrali, che corrisponde alla metà della finestra di campionamento. Dividere la frequenza per la risoluzione frequenziale fornisce l'indicazione del bin corrispondente nella FFT. L'oggetto *trigger*, dopo aver passato il valore intero relativo al contatore del frame temporale, manda un bang ad un oggetto uzi. L'oggetto uzi una volta sollecitato con un messaggio di "bang" manda in output, nel minor tempo possibile, un quantitativo di messaggi "bang" pari al valore passatogli come argomento, in questo caso 150. Il trigger che riceve i messaggi di "bang" passati da *uzi* a sua volta manda per tre uscite 150 messaggi di "bang" che vanno in ingresso a zl. I tre oggetti zl ricevendo i messaggi di "bang" fanno uscire uno ad uno i valori di indice di frequenza, insieme ai valori di ampiezza e fase. L'accoppiamento indice-ampiezza e indice-fase entra all'interno dell'oggetto *peek~*. L'oggetto peek~ scrive nei buffer "mag" e "phase" all'indice di frequenza il valore corrispondente. Una volta popolati i buffer, l'esecuzione prevede la lettura del contenuto tramite l'oggetto *lookup~*. I valori di ampiezza e fase vengono quindi convertiti da coordinate cartesiane a coordinate polari e passate all'oggetto *fftout~* che effettua la IFFT e manda il risultato all'oggetto *pfft~*, appartenente alla patch di livello superiore, per poter essere ascoltato.

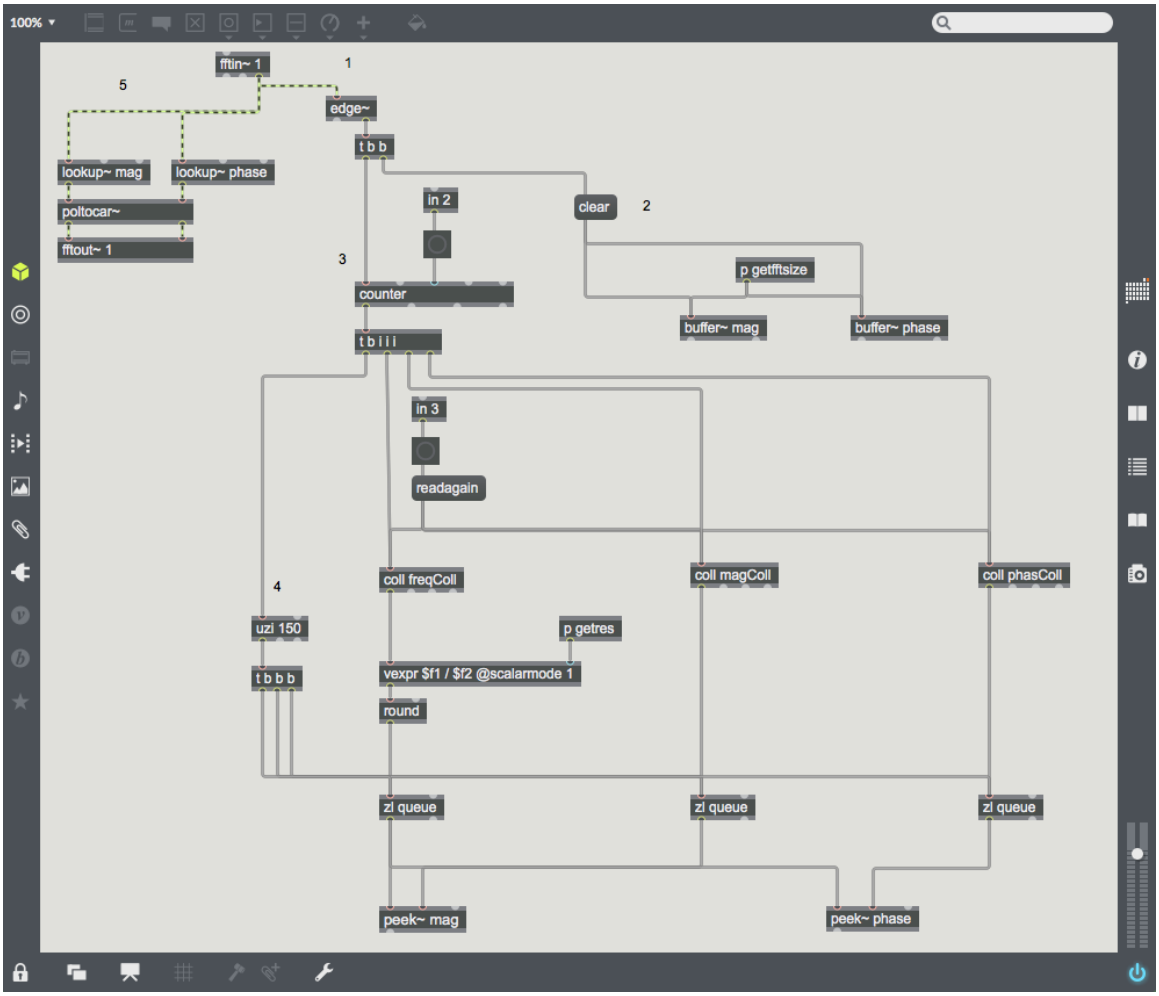

Figura 19: subpatch di sintesi.

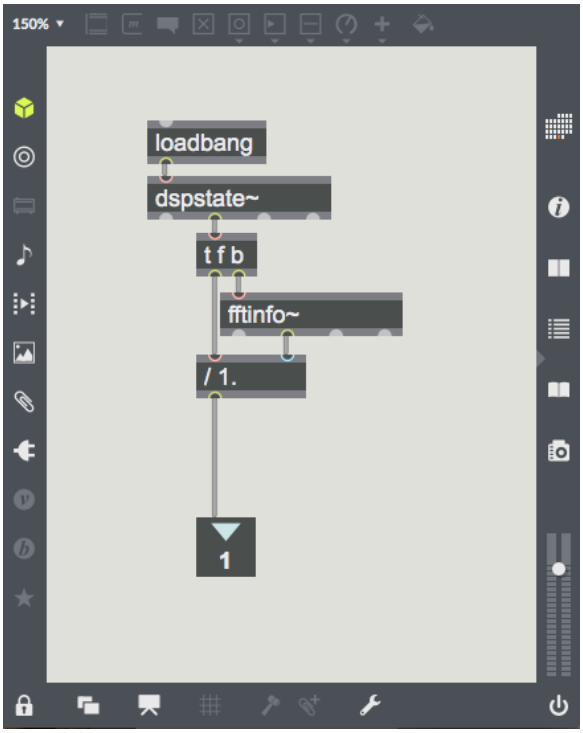

Figura 20: Calcolo della risoluzione frequenziale.

4.4 Terzo caso di studio

Nel caso di studio precedentemente presentato i parametri di analisi vengono raccolti in Max/MSP, l'analisi viene effettuata tramite codice Python, e la sintesi viene realizzata in una patch Max/MSP. L'architettura che permette lo scambio dei dati forniti in uscita dall'algoritmo di analisi e necessari per guidare la sintesi, prevede la formattazione di un file di interscambio di testo. Il funzionamento della patch è compromesso a causa flusso di operazioni all'interno della patch di sintesi. La discrepanza temporale tra gli oggetti che lavorano ad audio-rate e quelli che lavorano ad un rate inferiore non permette alla patch di essere performante. Il risultato presentato risulta utile sotto un profilo teorico, come caso di studio.
Il terzo caso di studio parte dai risultati appena esposti e si pone in un ottica di graduale raffinamento.

Il risultato proposto in questa sezione si compone di:

- Una patch Max/MSP per raccogliere i parametri di analisi.

- Una collezione di file Python che esegue l'analisi e scrive il risultato su dei file *.wav*.
- Una patch Max/MSP che implementa la sintesi.

4.4.1 Max/MSP: Patch di avvio

La patch di avvio del terzo caso di studio si presenta identica al caso di studio appena esposto e per tale ragione si rimanda al 4.3.1 per una completa spiegazione del funzionamento. Quando l'utente inserisce tutti i parametri di analisi, l'oggetto *shell* manda in esecuzione il codice Python, passandogli come argomenti da linea di comando i valori inseriti dall'utente. Il funzionamento prosegue dunque con la parte di codice che esegue l'analisi e genera il file di interscambio.

4.4.2 Python: analisi e generazione del file di interscambio

Come nel caso precedente, il codice Python prende i parametri di input forniti dall'utente e li utilizza per eseguire la funzione di analisi. La funzione di analisi è la stessa dei casi di studio precedentemente illustrati. Svolta l'analisi, vengono restituiti quattro array riguardarti la frequenza, la magnitudine, la fase e il modello stocastico generato. La funzione di analisi divide la fase per 3.15 in modo tale che i valori non eccedano -1 e 1 e risultino quindi troncati a causa del fenomeno di *clipping* nel file *.wav*. I valori presenti nell'array riguardante la magnitudine vengono convertiti in valori di ampiezze lineari. A differenza del caso precedente, viene scelto un file di interscambio di tipo audio, per poter facilitare la lettura ad audio-rate all'interno dell'ambiente Max/MSP. Il codice Python successivo implementa la scrittura dei file di interscambio relativi ad ampiezza e fase.

```
1  Ns = 512
2  fftbins = (Ns // 2) + 1
3  totlen = numrows * fftbins
4  magtowav = np.zeros(totlen)
5  phasetowav = np.zeros(totlen)
6  res = fs / Ns
7
8  for k in range (0, numrows):
9          for l in range (0, numcol):
10                 j = int((tfreq[k,l] // res))
11                 m = int((k * fftbins) + j)
12                 magtowav[m] = tmag[k,l]
```

```
13
14 for k in range (0, numrows):
15         for l in range (0, numcol):
16                 j = tfreq[k,l] // res
17                 m = int((k * fftbins) + j)
18                 phasetowav[m] = tphase[k,l]
19
20 UF.wavwrite(magtowav, fs, "/Users/SMSuser/caso_di_studio_3/Max
   /magtowav.wav")
21 UF.wavwrite(phasetowav, fs, "/Users/SMSuser/caso_di_studio_3/
   Max/phasetowav.wav")
```

I file di interscambio sono formattati come una sequenza di frame temporali all'interno dei quali viene scritto, per ciascuna frequenza convertita in indice, il rispettivo valore di ampiezza e fase. Per tale ragione non risulta necessario generare un file di interscambio relativo alle frequenze. Il codice Python salva il valore del numero di bins totali all'interno di una variabile. Viene poi moltiplicato il numero di bins per il numero di righe presenti negli array generati dalla funzione di analisi, che corrisponde al numero di frame totali, ottenendo la lunghezza che i file finali dovranno avere. Vengono dichiarati due file con la lunghezza appena calcolata e si determina la risoluzione in frequenza. Il codice prosegue con due cicli: il ciclo esterno lavora su un frame temporale alla volta. Il secondo ciclo lavora per ogni sinusoide. Viene trasformato il valore di frequenza in indice. Il valore di indice relativo ad un determinato frame temporale diventa il valore di indice assoluto rispetto all'intero file e, calcolata la posizione viene scritto il valore di ampiezza. Il procedimento viene ripetuto per la fase. Gli array risultanti da queste operazioni sono scritti su due file *.wav*. Il proseguimento delle operazioni per generare la sintesi avviene all'interno di una patch Max/MSP.

4.4.3 Max/MSP: Patch di sintesi

La patch Max/MSP che implementa la sintesi si sviluppa su più livelli. La patch principale, illustrata nella figura 21, consiste in un oggetto *pfft~* a cui è collegato un *live.gain~*, che gestisce il volume di uscita. L'output viene infine passato ad *ezadac~*, che implementa la conversione digitale-analogico per emettere il segnale sonoro. Come nel caso esposto in 4.3.3, anche in questo caso, il cuore algoritmico della patch risiede nella subpatch inserita come argomento obbligatorio dell'oggetto *pfft~*. Oltre al nome della subpatch, *Sintesi_core3*, a *pfft~* prende come argomenti la dimensione della FFT e il fattore di sovrapposizione. *Pfft~* possiede tre inlet, poichè all'interno della subpatch sono previsti tre box di "input". Ad un messaggio

di “bang” passato al secondo inlet, corrisponde un reset dei buffer contenuti nella subpatch. Un “bang” passato al terzo inlet comporta la rilettura degli ultimi file di interscambio caricati.
Nella parte successiva la trattazione si concentra sul funzionamento della subpatch “Sintesi_core3” come argomento alla patch di sintesi.

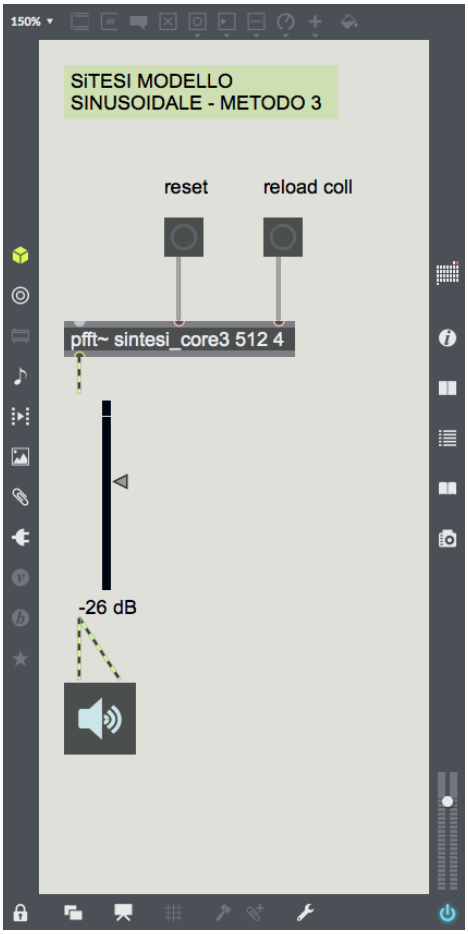

Figura 21: patch di sintesi.

La figura 22 mostra il contenuto della subpatch di sintesi, il cui funzionamento prevede di leggere i valori di ampiezza e fase contenuti in due buffer temporanei, per poterli sintetizzare tramite IFFT.
La subpatch carica all’interno di due buffer chiamati “mag” e “phase” il contenuto dei file di interscambio. Questa operazione avviene quando la patch viene caricata,

tramite *loadbang*. Durante l'esecuzione, per recuperare i valori corretti ed in successione ad audio-rate all'interno dei buffer, si utilizza l'informazione relativa al bin correntemente processato. Dal momento che dal terzo outlet di *fftin~* esce, per ogni frame temporale, l'indice di bin relativo bisogna trasformare questa informazione in un indice di bin che tenga conto del passaggio dei frame temporali. Viene in un primo momento preso il valore di indice bin corrente e lo si somma al un valore che indica il numero totale di bin trascorsi dall'ultimo cambio di frame temporale. Per recuperare l'informazione di numero di bin trascorsi rispetto all'ultimo cambio di frame temporale viene utilizzato un sommatore che, rilevando il passaggio da un indice di bin diverso da zero ad un indice di bin uguale a zero, incrementa di uno. Questo valore viene moltiplicato all'informazione del numero totale di bin presenti nella FFT, informazione che si ricava grazie all'incapsulamento *p getnumbin*. L'oggetto *lookup~* funziona in modo tale che quando legge nei buffer i valori di indici su cui scorre l'indice interno del buffer va da -1 a 1. L'informazione relativa al numero di bin trascorsi dall'avvio della sintesi viene passata all'interno di un incapsulamento denominato *p scale*, che formatta gli indici che passano in modo tale che il valore sia scalato tra -1 e 1, rispetto alla dimensione dei buffer. A questo punto di esecuzione *lookup~* legge le informazioni all'interno del buffer, facendo uscire dal suo outlet i valori di magnitudine e fase. Dal momento che, come espresso in 4.4.2, la fase viene divisa per 3.15 nel codice Python, nella patch la fase viene moltiplicata per 3.15, per riportare i valori di fase al corrispondente originale fornito dalla funzione di analisi. Il risultato passa all'interno di *poltocar~* che prende le informazioni di ampiezza e fase, e restituisce le informazioni di parte reale e parte immaginaria. Tali dati vanno in ingresso a *fftout~* che realizza la IFFT e manda il risultato alla patch di livello superiore.

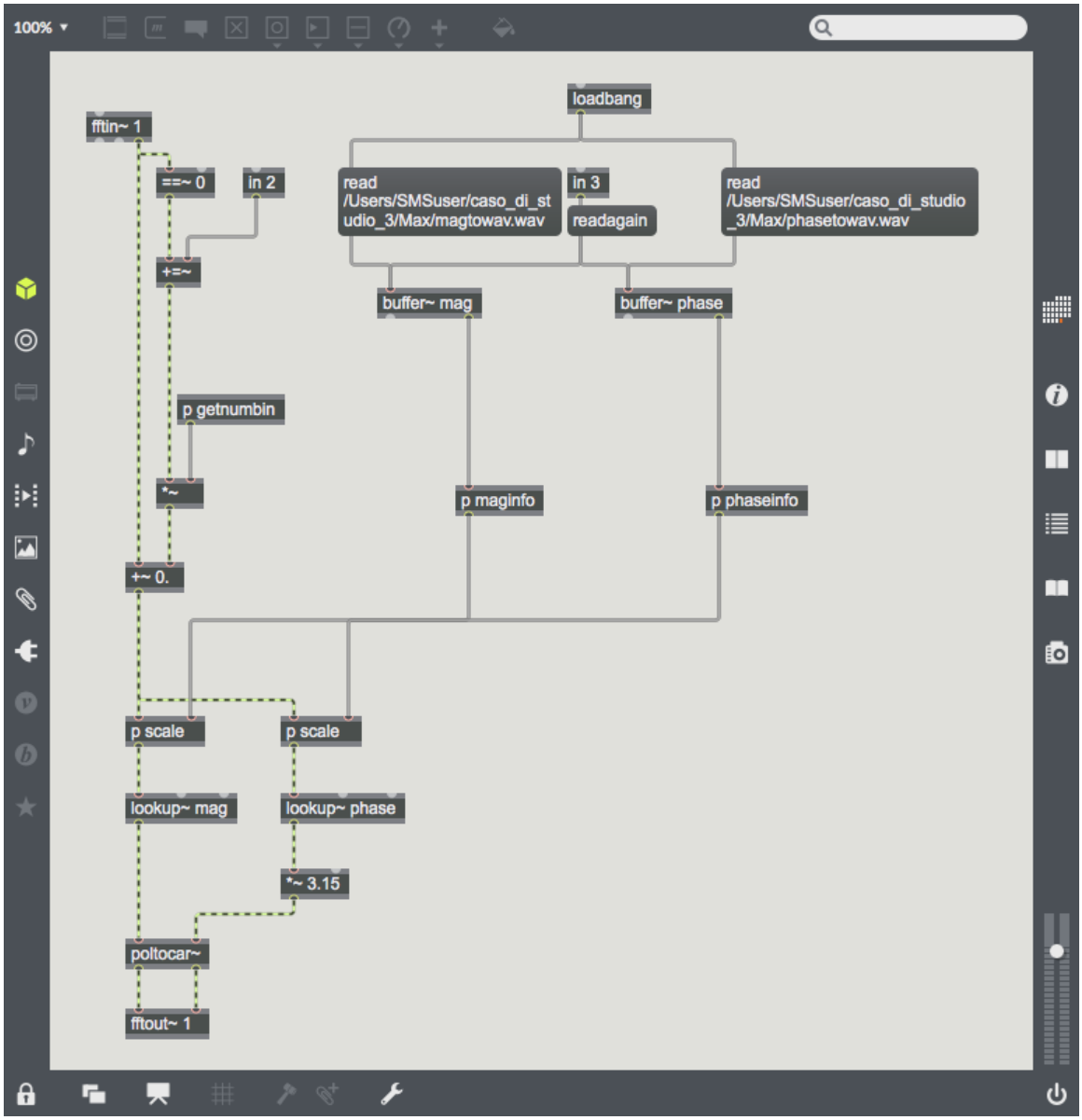

Figura 22: subpatch di sintesi.

4.5 Osservazioni finali

Dei tre metodi esposti nel presente capitolo utilizzati per implementare la Sintesi per Modello Sinusoidale, il primo rappresenta il risultato che maggiormente soddisfa gli intenti della ricerca. Costituisce un implementazione dotata di solidità a livello performativo. Lascia spazio a margine di miglioramento ed ampliamento dell'ambiente. Il secondo ed il terzo metodo esposto invece sono stati adottati al fine di esplorare le possibilità di Max/MSP nell'ottica di una maggiore coinvolgimento dell'ambiente nei processi computazionali, legati alla sintesi. Entrambi risultati come punto di partenza per successive elaborazioni, queste due strade intraprese necessitano di revisione e integrazioni con le parti che mancano, ad esempio la sintesi del rumore. Tra i due, il terzo caso si avvicina di più ad uno schema utilizzabile da un utente. Il secondo caso soffre l'incongruenza temporale che si crea nel funzionamento a rate diversi degli oggetti Max e del modulto MSP utilizzati. Il terzo caso, dal momento che l'implementazione dell'algoritmo di sintesi prevede l'utilizzo esclusivo di oggetti appartenenti al modulo MSP risulta, seppur non in una maniera sufficiente ad essere utilizzato, maggiormente performante.

Capitolo 5

Conclusioni e sviluppi futuri

Il capitolo 4 illustra i risultati del lavoro di ricerca, che hanno portato alla generazione di tre schemi implementativi. Ciascuno schema condivide con gli altri l'intenzione di realizzare un'implementazione della Sintesi per Modello Sinusoidale, utilizzando un ambiente di sviluppo ibrido, costituito dall'interazione tra Max/MSP e Python. I risultati ottenuti non hanno precedenti noti in letteratura, e costituiscono una base per successive elaborazioni. Negli sviluppi futuri per il progetto presentato nel presente elaborato, sono previste ottimizzazioni negli schemi di comunicazione e interazione tra Python e Max/MSP. Come esemplificato, l'interazione tra Python e Max/MSP è in una fase di sviluppo poco più che embrionale, ma si immagina che in futuro i due ambienti godranno di una maggiore documentata interazione, tramite lo sviluppo di mezzi software che permettono metodi più flessibili per incorporare script Python dentro Max/MSP. Parallelamente, l'altro aspetto che offre margini di ottimizzazione è la parte algoritmica che si occupa di analisi e sintesi rispetto alla Sintesi per Modello Sinusoidale. Le patch Max/MSP che si occupano di sintesi sono oggetto di revisione e saranno integrate con le componenti mancanti.

Il primo metodo presentato risponde in maniera esaustiva all'obiettivo posto. Il punto di contatto tra i due ambienti, l'external di Max/MSP *shell* permette il lancio e la gestione dell'applicativo Python. Una possibilità per ampliare questo modello, alla luce di un maggior coinvolgimento dell'ambiente Max/MSP, potrebbe essere quella di raccogliere tutti i parametri di analisi all'interno di Max/MSP ed implementare un lettore di file *.wav* che permette, ad analisi svolta, di riprodurre i brani all'interno della patch Max/MSP, eliminando l'interfaccia grafica Python. Il codice Python in questo modo costituirebbe il motore computazionale dello schema, ma all'utente sembrerebbe, in fase di utilizzo, di non lasciare mai l'ambiente Max/MSP.

Gli sforzi documentati con il secondo ed il terzo risultato proposto, che utilizzano

Max/MSP per implementare la sintesi, lasciano spazio ad ampi margini di miglioramento. Tra questi due schemi, l'ultimo costituisce una base di partenza in cui operare dei raffinamenti e integrazioni, mentre il suo predecessore è risultato un utile caso di studio per la sintesi della parte periodica del suono. Vi è la necessità di revisionare e migliorare quanto effettuato, per pareggiare la funzionalità di questi schemi con quello che impiega codice Python per la sintesi. Oltre al naturale proseguimento della ricerca, con l'integrazione della sintesi della componente stocastica, il progetto deve essere oggetto di riqualificazione attraverso una fase di debugging che miri a migliorarne le prestazioni, e a offrire un risultato competitivo con quanto già realizzato in Python, per potersi porre come alternativa.

In ciascuno schema coesistono elementi dotati di validità ed elementi che lasciano spazio a successivi miglioramenti. Il percorso che ha portato allo sviluppo degli schemi proposti nel presente elaborato è aperto alla ridefinizione, alla manipolazione e all'ampliamento degli ambienti. Nell'attività di revisione di quanto realizzato fino ad ora risiede dunque il naturale proseguimento del percorso intrapreso.

Capitolo 6

Ringraziamenti

Ringrazio Sylviane Sapir, che nelle aule del Conservatorio ha saputo proporre un argomento di ricerca conciliante verso gli interessi che dentro di me erano motivo di divisione. Un caloroso ringraziamento va a Federico Avanzini per avermi instradato sul percorso, per le puntuali osservazioni sul lavoro svolto e per le chiacchiere spese tra i corridoi del Conservatorio Giuseppe Verdi di Milano e quelli del nuovo edificio di Informatica Musicale. Il presente progetto di ricerca non sarebbe stato possibile senza i confronti con Giorgio Presti, che ringrazio per la sua pazienza e per avermi stimolato costantemente al ragionamento. Ringrazio infine Giovanni Verrando che ha educato la mia curiosità senza sterilizzarla, affichè diventasse un delicato strumento di composizione.

Bibliografia

[1] Richard Cann. «An Analysis/Synthesis Tutorial: Part 1». In: *Computer Music Journal* (1979), pp. 6–11.

[2] Dodge Charles e Thomas A Jerse. *Computer Music: Synthesis, Composition and Performance*. 1985.

[3] John M Grey e James A Moorer. «Perceptual evaluations of synthesized musical instrument tones». In: *The Journal of the Acoustical Society of America* 62.2 (1977), pp. 454–462.

[4] Reinier Plomp. «Experiments on tone perception». Tesi di dott. TNO, 1966.

[5] Miller Puckette. «Combining event and signal processing in the MAX graphical programming environment». In: *Computer music journal* 15.3 (1991), pp. 68–77.

[6] Juan G Roederer. «The psychophysics of musical perception». In: *Music Educators Journal* 60.6 (1974), pp. 20–30.

[7] Xavier Serra. «A system for sound analysis/transformation/synthesis based on a deterministic plus stochastic decomposition». In: (1989).

[8] Xavier Serra et al. «Musical sound modeling with sinusoids plus noise». In: *Musical signal processing* (1997), pp. 91–122.

[9] Xavier Serra e Julius Smith. «Spectral modeling synthesis: A sound analysis/-synthesis system based on a deterministic plus stochastic decomposition». In: *Computer Music Journal* 14.4 (1990), pp. 12–24.

[10] Julius O Smith e Xavier Serra. «PARSHL: An analysis/synthesis program for non-harmonic sounds based on a sinusoidal representation». In: *Proceedings of the 1987 International Computer Music Conference, ICMC; 1987 Aug 23-26; Champaign/Urbana, Illinois.[Michigan]: Michigan Publishing; 1987. p. 290-7.* International Computer Music Conference. 1987.

[11] WA Yost e DW Nielsen. *Fundaments of hearing*. 1977.

[12] Udo Zölzer. *DAFX: digital audio effects*. John Wiley & Sons, 2011.