Conservatorio di Musica Santa Cecilia

Dipartimento di Nuove Tecnologie e Linguaggi Musicali

Tesi di Laurea Biennale in Musica Elettronica

Sistemi Complessi Adattivi per la performance musicale in Live Electronics

Relatore:

Giuseppe Silvi

Candidato:

Luca Spanedda

Correlatori:

Agostino Di Scipio

Dario Sanfilippo

Anno Accademico 2021/2022

Dichiarazione

Dichiaro solennemente che come autore di questo documento, mi assumo la piena responsabilità del suo contenuto. Qualora parti del testo siano tratte da altre fonti, mi impegno a citarle in modo esplicito per rendere onore al lavoro originale e rispettare i diritti d'autore.

Luca Spanedda

Ringraziamenti

Prima di entrare nel merito desidero esprimere la mia sincera gratitudine a coloro che hanno reso possibile e significativo il mio percorso. La mia famiglia, che mi ha sempre sostenuto in ogni tipo di situazione che ho incontrato. I miei amici, talentuosi artisti e persone di grande valore umano. I miei compagni di corso, compositori straordinari e compagni di vita preziosi. Giuseppe Silvi, mio maestro e amico, che mi ha accompagnato e guidato con amorevoli insegnamenti in questi ultimi anni mio percorso compositivo. E insieme a lui voglio ringraziare tutti i grandi maestri e professori del dipartimento di musica elettronica del conservatorio Santa Cecilia di Roma, che hanno contribuito alla mia crescita e formazione come compositore, tra cui Pasquale Citera, Nicola Bernardini, Marco Giordano, Luigi Pizzaleo, Federico Scalas, ed infine il maestro Michelangelo Lupone, che mi ha aiutato a far maturare il mio pensiero compositivo e mi ha sempre incoraggiato a fare del mio meglio. Infine desidero esprimere la mia profonda gratitudine a Dario Sanfilippo e Agostino Di Scipio, due grandi mentori, maestri e persone straordinarie, che mi hanno accompagnato in questo entusiasmante percorso compositivo. Grazie al loro prezioso supporto ed alla loro collaborazione, ho potuto portare avanti le mie idee visionarie nella composizione musicale elettroacustica, e senza di loro, questo lavoro non sarebbe stato possibile. Sono immensamente grato a tutti voi, perché senza la vostra guida non sarei stato in grado di realizzare i miei sogni, e questa tesi ne è una prima testimonianza della prova del mio sogno realizzato, che è per me solo un punto di partenza verso nuove conquiste. Tuttavia, io credo che la bellezza dell'apprendere stia nell'essere sempre ad un punto di partenza e mai uno di arrivo.

It is not irritating to be where one is. It is only irritating to think one would like to be somewhere else.

(John Cage - Lecture on Nothing - Incontri musicali. Quaderni internazionali di musica contemporanea, Agosto 1959.)

Grazie a tutti con affetto.

Abstract

Il lavoro qui presentato è uno studio di analisi, implementazione e esecuzione di tre sistemi complessi adattivi per la performance musicale in live electronics. La scelta di questi tre sistemi corrisponde a tre diversi casi di studio nell'implementazione di dinamiche non lineari sfruttate per la generazione dei comportamenti emergenti nei sisitemi complessi. Una prima parte del lavoro tratterà dell'implementazione e l'analisi di due brani rispettivamente di Agostino Di Scipio e Dario Sanfilippo. Di Agostino di Scipio un sistema con non linearità provenenti dal mondo fisico, che sfrutta fenomeni generati dalla catena elettroacustica all'interno dell'ambiente e riportati poi all'interno del sistema digitale. Di Dario Sanfilippo invece un sistema che sfrutta delle non linearità appositamente programmate dal compositore nel mondo digitale, controllate tramite agenti di autoregolazione scritti nel software. Infine l'ultima parte del lavoro è dedicata alla composizione di un mio brano, che sfrutta elementi di logiche ibride apprese dai due casi di studio presentati qui, e che andrà a conclusione del lavoro di ricerca sviluppato durante il corso della tesi.

Contents

List of Figures

Listings

1 Introduzione

All'inizio del XX Secolo si manifestò una situazione globale di importanti cambiamenti in tutti gli ambiti.

Nel mondo dell'arte con la nascita delle avanguardie artistiche e in risposta alla problematica del dover trovare nuovi modi di fare che riflettano le tematiche della società attuale. Nelle scienze, con l'esigenza di dover introdurre nuovi paradigmi per far fronte alla grande crisi dei fondamenti e delle certezze. Durante il corso del secolo poi, questi cambiamenti hanno portato ad importanti punti di incontro non più occasionali fra questi due ambiti. La figura dell'artista inizia ad interessarsi alle nuove tecnologie e teorie scientifiche, e questo è stato un importante punto in comune a tutte le avanguardie dell'epoca, inclusa quella musicale.

Nel caso di questa tesi, l'attenzione che ripongo a questo scenario del XX Secolo è in particolare su un cambio di paradigma scientifico nascente durante la seconda guerra mondiale e consolidato al termine di questa, la nascita della cibernetica e la conseguente formulazione di scienze della complessità. La complessità, non è un ambito trattato da una sola scienza, ma più un nuovo modo di pensare e di osservare i fenomeni della realtà.

Se nella cosmologia greca, questa realtà veniva spiegata come caos per l'insieme disordinato e indeterminato degli elementi materiali, contrapposto al cosmos che rappresenta l'ordine, come si può osservare fra i miti delle varie Teogonie fra cui la più famosa quella scritta da Esiodo.[1] Oggi la parola caos ha invece un significato decisamente meno generale. Il caos, anzi il caos deterministico, è la scienza che studia i sistemi dinamici che esibiscono una sensibilità esponenziale rispetto alle condizioni iniziali. O, in termini più rigorosi, è la scienza che studia la dinamica dei sistemi non lineari. Questo cambio di paradigma ha avuto inizio verso la fine del XIX secolo, quando uno studioso nell'ambito della meccanica classica, Henri Poincaré, osservò e analizzò la possibilità di un comportamento fortemente irregolare di alcuni sistemi

[1] Sofia Ranzato. "L'origine del mondo divino Una lettura di Esiodo". ita. Thesis. Facoltà di Lettere e Filosofia C. d. L. S. in Scienze dell'Antichità, 2006/2007, p. 10.

dinamici studiando il problema dei tre corpi, che lo portò alla scoperta del caos matematico.[2] La scoperta di Poincaré segnerà un punto di svolta che verrà ripreso poi solamente a metà del secolo successivo dal meteorologo Edward Norton Lorenz, quando nel '63 Lorenz pubblicherà il suo articolo *Deterministic Nonperiodic Flow*[3], nel quale tratta del comportamento caotico in un sistema semplice e deterministico, con la formazione di un attrattore strano, e aprendo di fatto ufficialmente la strada quella che diverrà poi la famosa teoria del caos. Mostrando come in realtà all'interno dell'ordine emergano forme di disordine, e all'interno del disordine siano presenti forme di ordine.

Prima dell'introduzione di questa tesi, si rende comunque necessario qualche chiarimento e qualche informazione preliminare per comprendere cosa s'intenda con certi termini che ritroveremo nel corso della tesi, e questi sono la nozione di sistema dinamico e di sistema complesso. Un sistema dinamico è un modello matematico che rappresenta un oggetto (sistema), con un numero finito di gradi di libertà che evolve nel tempo secondo una legge deterministica; Mentre per sistema complesso si intende un sistema dinamico a multicomponenti, ovvero composto da diversi sottosistemi che solitamente interagiscono fra loro. Tipicamente un sistema dinamico viene rappresentato analiticamente da un'equazione differenziale, espressa poi in vari formalismi, e identificato da un vettore nello spazio delle fasi: lo spazio degli stati del sistema, dove "stato" 'e un termine che indica l'insieme delle grandezze fisiche, dette variabili di stato, i cui valori effettivi "descrivono" il sistema in un certo istante temporale.

[2] June Barrow-Green. *L'Ottocento: astronomia. Il problema dei tre corpi e la stabilità del Sistema solare*. URL: https://www.treccani.it/enciclopedia/l-ottocento-astronomia-il-problema-dei-tre-corpi-e-la-stabilita-del-sistema-solare_%28Storia-della-Scienza%29/#:~:text=La%20formulazione%20del%20problema%20dei,il%20moto%20negli%20istanti%20successivi.. (accessed: 03.11.2022).

[3] Edward Lorenz. "Deterministic Nonperiodic Flow". In: *Journal of Atmospheric Sciences* 20.2 (1963), pp. 130–148.

1.1 La Cibernetica

Rintracciando l'origine di questi termini è necessario tornare a questo complesso scenario del XX secolo, più precisamente al termine della seconda guerra mondiale e qualche decennio prima della formulazione della teoria del caos, in quel periodo uno dei più importanti avanzamenti nelle scienze che contribuì alla formazione del primo paradigma della complessità, risiedette nell'introduzione della cibernetica.

La cibernetica è la scienza che studia i principi astratti di organizzazione nei sistemi complessi, ed ebbe inizio durante gli anni della seconda guerra mondiale, merito del fisico e matematico Norbert Wiener. Nel '40 Wiener insieme ad altre ad altre prominenti figure provenienti da diversi ambiti scientifici, come Ross Ashby, Margaret Mead, Gregory Bateson, Heinz von Foerster, partecipano ad una serie di conferenze multidisciplinari chiamate "The Macy Conferences", inizialmente intitolate come "Feedback Mechanism in Biology and the Social Sciences" con l'obiettivo comune di andare a definire gli ambiti di interesse della nuova scienza. A seguito nel '48, ispirato dalla meccanica ed i suoi risultati conseguiti durante la guerra e contemporaneamente dallo sviluppo della teoria della comunicazione (o informazione) di Claude Shannon, con la volontà di sviluppare una teoria generalizzata dei principi di organizzazione e controllo nei sistemi emersi durante le conferenze, pubblicherà un libro: La cibernetica, controllo e comunicazione nell'animale e nella macchina; in cui definiva l'ambito di interesse e gli obiettivi della nuova disciplina inaugurando anche l'uso del nuovo termine da lui coniato. A seguito di questo libro che riscuoterà un importante successo, le conferenze presero il nome di "Cybernetics, Circular Causal, and Feedback Mechanism in Biological and Social Systems",[4] riconoscendo Wiener come la principale figura di spicco della nuova scienza.

In particolare come evidenziato fino ad ora dalla sua natura multidisciplinare, la cibernetica non si interessa di individuare in cosa consistano questi sistemi, ma più che altro di comprenderne il loro funzionamento.

[4] Luca Fabbris e Alberto Giustiniano. *CFP18 cibernetica, sistemi, teorie, modelli.* URL: https://philosophykitchen.com/2022/03/cfp18-cibernetica-sistemi-teorie-modelli/. (accessed: 03.11.2022).

Le fortunate premesse iniziali della cibernetica risiedevano in una convinzione da parte di questi scienziati provenienti dai differenti ambiti disciplinari, che esistesse uno "schema processuale" comune ad organismi viventi e macchine, rintracciato attraverso una ricerca uniforme garantita dall'utilizzo di un metodo "sintetico" e "comportamentale". L'aspetto meta-disciplinare del pensiero cibernetico, esplicito nella sua fondazione, godrà però di una fama più popolare che conseguirà in importanti realizzazioni tra l'inizio degli anni '60 e la metà del '70, grazie al contributo degli scienziati: Heinz von Foerster, Margaret Mead, Gregory Bateson, e altri; proprio in quel periodo si compieranno ulteriori passi fondamentali che porteranno il pensiero sistemico verso un consolidamento di una scienza più concreta. Uno dei casi più rilevanti di quel periodo, è il passaggio da una "cibernetica di primo ordine" a una "cibernetica di secondo ordine", anche chiamata come "la cibernetica dei sistemi di osservazione", questa distinzione si deve a Heinz Von Foerster, che è stato uno scienziato e filosofo austriaco noto per i suoi contributi alla cibernetica e alla teoria della comunicazione.[5] La differenza fra cibernetica di primo e di secondo ordine, risiede principalmente nel fatto, che mentre nel primo periodo lo studioso di cibernetica (di primo ordine) studiava un sistema da un punto di vista passivo, da quello dell'osservatore dei comportamenti del sistema, senza tener conto della propria influenza nei casi di studio. Il cibernetico di secondo ordine invece, è consapevole della propria influenza all'interno del sistema, riconoscendo il sistema come un agente con cui interagire, e riconoscendo esso stesso come agente nell'interazione col sistema, e di conseguenza proprio come spiega Heinz Von Foerster nei suoi scritti, di come la realtà per come la percepiamo possa esistere soltanto se si tiene conto del fatto che la sua rappresentazione è fortemente dipendente dall'esistenza di un ambiente che racchiude il sistema stesso.[6] In altre parole, l'ambiente non esiste in sé,

[5] Bernard Scott. "Second-order cybernetics: an historical introduction". In: *Kybernetes* 33.9.10 (2003), pp. 1365–1378. DOI: doi:10.1108/03684920410556007. URL: https://sites.ufpe.br/moinhojuridico/wp-content/uploads/sites/49/2021/10/Ciber-2b-22-out.-second-order-cybernetcs.pdf.

[6] Heinz Von Foerster. *Understanding Understanding: Essays on Cybernetics and Cognition*. en. Vol. 2. 1. 2005. DOI: 10.29173/cmplct8737. URL: https://journals.library.ualberta.ca/complicity/index.php/complicity/article/view/8737 (visited on 11/22/2022).

ma viene creato dalla relazione tra l'osservatore e ciò che viene osservato. Questa teoria ha importanti implicazioni per la cibernetica, poiché sottolinea l'importanza dell'interazione tra il sistema e il suo ambiente nella comprensione del funzionamento del sistema stesso. Inoltre, sottolinea l'importanza di considerare la prospettiva dell'osservatore nell'analisi dei sistemi e dei processi. In conclusione, nella visione di Foerster, l'organizzazione ed eventuale disorganizzazione di alcun sistema può esistere se questo non viene studiato prendendo in considerazione l'ambiente che lo racchiude. A partire dunque da queste importanti premesse, e dalle conseguenze che la cibernetica ha avuto in tutti gli ambiti delle scienze, è doveroso notare che questa ha avuto un ruolo centrale nello sviluppo di molti studi scientifici e la nascita di nuovi ambiti come: l'intelligenza artificiale, la teoria del caos, la teoria della catastrofe, la teoria dei controlli, la teoria generale dei sistemi, la robotica, la psicologia, le scienze sociali, e molto altro (perfettamente in accordo con la sua natura multidisciplinare).

1.2 Le Cibernetiche nella musica

All'inizio degli anni '60, in seno alle nascita delle scienze complesse, l'uso di sistemi di feedback e la rilevanza dei circuiti informativi chiusi nelle strutture organizzate ha goduto di uno slancio popolare anche nel mondo della musica e più in generale dell'arte.

Tuttavia come vedremo, a parte casi popolari di deliberate dichiarazioni formali da parte degli artisti, non bisogna pensare ai lavori che andremo a citare come atti pionieristici che sanciscono una volta per tutte la nascita della cibernetica in musica, ma è più corretto pensare alle questioni sistemiche come ad una sensibilità comune condivisa in un certo periodo da diversi autori provenienti da diverse parti del mondo, che sono stati influenzati e si sono influenzati a vicenda con le stesse idee per un interesse condiviso riguardo le teorie cibernetiche di Weiner e delle Macy Conferences.

In Europa nel '51, Herbert Eimert, Werner Meyer-Eppler e Robert Beyer di comune accordo con il direttore della NWDR, Hanns Hartmann, attrezzarono e crearono

quello che divenne il primo studio per la Musica Elettronica.[7] Questo divenne lo studio più influente al mondo durante gli anni '50 e '60, con ospiti alcuni dei più importanti compositori contemporanei provenienti da tutta Europa, come Franco Evangelisti, Karlheinz Stockhausen, György Ligeti, Roland Kayn, Herbert Brün, Cornelius Cardew, e molti altri. In quel periodo il lavoro di ricerca condotto da Werner-Meyer Eppler, scienziato, musicista, fra gli ideatori e direttori dello studio di Colonia e direttore dell'istituto di fonetica all'università di Bonn, pone una certa attenzione in quelle che furono le prime teorizzazioni della teoria dell'informazione e della cibernetica, che porteranno l'autore alla scrittura di importanti testi di ricerca. Fra le altre cose Werner-Meyer Eppler nel '50 escogita un uso particolare del magnetofono, mediante retroazione *(feedback)* fra le testine di riproduzione e registrazione, creando a tutti gli effetti una prima forma di *tape delay*.[8] Dalle esperienze dello studio di Colonia ne usciranno molti compositori interessati alle teorie cibernetiche.

Un caso importante in questo scenario è quello del compositore Roland Kayn, Il progetto di musica cibernetica di Kayn ha ricevuto il suo impulso iniziale quando nel '53, ad allora giovane musicista e studente universitario, venne in contatto con il filosofo Max Bense professore all'Università Tecnica di Stoccarda. Subito dopo il suo primo incontro con Bense sempre nel '53, Kayn entrò in contatto con Herbert Eimert presso lo studio elettronico della Westdeutscher Rundfunk di Colonia. Roland Kayn era affascinato dal potenziale sonoro offerto dalle nuove tecnologie, ma trovava che l'estetica serialista dominante nello studio in quegli anni era per lui qualcosa di troppo restrittivo, esperienza che lo portò per i successivi dieci anni a concentrarsi principalmente sulla composizione strumentale e le applicazioni delle teorie cibernetiche in modo formale.[9] Sempre in quel periodo, sostanzialmente di-

[7] Agostino Di Scipio. *Circuiti del Tempo Un percorso storico-critico nella creatività musicale elettroacustica e informatica*. it. N. 38. Lucca: Libreria Musicale Italiana, 2021. ISBN: 9788855430685.

[8] Di Scipio, *Circuiti del Tempo Un percorso storico-critico nella creatività musicale elettroacustica e informatica*.

[9] Thomas W. Patteson. "The Time of Roland Kayn's Cybernetic Music". In: (), pp. 1–11. URL: https://kayn.nl/wp-content/uploads/2016/12/The-Time-of-Roland-Kayns-Cybernetic-Music-Thomas-Patteson.pdf.

verso e altrettanto importante è il caso di Franco Evangelisti, che dopo essersi avvicinato alla musica elettronica anche lui sotto la guida e su invito di Herbert Eimert, allo studio elettronico della Westdeutscher Rundfunk di Colonia nel '56, iniziò le sue ricerche che dopo un anno e mezzo di intenso lavoro lo portarono al completamento della sua forse più importante composizione elettronica: Incontri di fasce sonore, trasmessa da Radio Colonia nel '57. Nel '59, Evangelisti è nuovamente in Italia, dove fu tra i promotori della settimana internazionale di nuova musica a Palermo. L'anno seguente, assieme ad altri musicisti fondò l'associazione Nuova Consonanza, con lo scopo di diffondere la musica contemporanea italiana e straniera con concerti convegni ed eventi di vario tipo. Dall'associazione nacque più tardi l'omonimo Gruppo di improvvisazione (GINC: Gruppo d'Improvvisazione Nuova Consonanza) che allora veniva presentato come "il primo ed unico gruppo formato da compositori-esecutori" e che permise ad Evangelisti di mettere in pratica le proprie teorie sull'improvvisazione, riguardo queste citerà più volte deliberatamente in interviste, scritti, e altre documentazioni, il suo approccio sistemico/cibernetico in quelle che saranno le esperienze con il Gruppo.[10]

Nel '60 si trasferisce a Roma Roland Kayn da vincitore del Prix de Rome, dove dal '64 assieme ad Aldo Clementi e Franco Evangelisti prende parte al Gruppo di improvvisazione Nuova Consonanza del quale fece parte sino al '68, ed è in quel periodo che Kayn ispirato dalle teorie della cibernetica iniziò a sperimentare estensivamente con sistemi di autoregolazione basati su feedback loop, non più solo come modelli formali per composizioni strumentali ma anche come reti di generatori di segnale analogici.

In questo scenario, sempre negli anni '50 in Europa, uno dei primi artisti nella storia dell'arte ad evocare l'uso della cibernetica nei propri lavori è stato Nicolas Schoeffer con il suo ciclo di lavori "spazio-dinamici", in acronimo CYSP - Cybernetic Spatiodynamic. In particolare Schoeffer ha creato la prima installazione ad implementare

[10] Theo Gallehr. *Gruppo Nuova Consonanza Azioni. Documentario.* URL: `https://www.youtube.com/watch?v=dqvAhBJ99wA`. (accessed: 06.02.2023).

meccanismi di auto-regolazione, il CYSP-1[11], capace di essere sensibile all'ambiente esterno e a se stesso grazie ad una serie di tecnologie offerte dalla compagnia Philips (fotocellule e microfoni), questa prima scultura spazio-dinamica, è dotata di totale autonomia di movimento (viaggio in tutte le direzioni a due velocità) e di rotazione assiale ed eccentrica (messa in moto delle sue 16 lastre policrome pivotanti), ed era capace di reagire sonoramente a questi stimoli riproducendo una serie di registrazioni composte dal compositore francese Pierre Henry, collaboratore di Pierre Schaeffer ed insieme a lui figura centrale nella nascita della musique concrète. Questa scultura viene celebrata come una prima opera di carattere cibernetico entrata a far parte nel mondo dell'arte.

Cambiando continente e passando dall'Europa ad osservare cosa stava accadendo in America in quegli anni, troveremo tanti altri rilevanti atti pioneristici, come ad esempio quelli che sono stati i lavori di Louis e Bebe Barron. I due (marito e moglie) furono compositori e pianisti che si interessarono alla musica elettronica sin dal periodo della sua origine. Intorno al '50 i due si trasferirono al Greewitch Village a New York dove furono attivi in collettivi di musica sperimentale collaborando con persone come John Cage ed altri. I Barron trasformarono la loro casa in una specie di studio di musica elettronica dove scrivevano soundtracks per film sperimentali. Per Louis e Bebe il grande passo arrivò nel '56, quando i due si ritrovarono a scrivere la soundtrack per il film: *Forbidden Planet*, questo sarà il primo film mainstream di Hollywood ad utilizzare una soundtrack composta solamente ed interamente da elettronica. L'elettronica di *Forbidden Planet* è stata costituita a partire da dei circuiti appositamente creati da i due compositori, che deliberatamente ispirati dalle teorie cibernetiche di Wiener dichiareranno:[12]

What we did was pretty elementary: we would attach resistors and capacitors to

[11]Dario Sanfilippo and Andrea Valle. "Feedback Systems: An Analytical Framework". In: *Computer Music Journal* 37.2 (2013), pp. 12–27. DOI: doi:10.1162/COMJa00176. URL: https://direct.mit.edu/comj/article-abstract/37/2/12/94420/Feedback-Systems-An-Analytical-Framework?redirectedFrom=PDF.

[12]Christina Dunbar-Hester. "Listening to Cybernetics: Music, Machines, and Nervous Systems, 1950-1980". en. In: *Science Technology e Human Values* 35.1 (2010), pp. 113–139. URL: http://www.jstor.org/stable/27786196.

activate these circuits... negative and positive feedback was involved - Wiener talks about all that. The same conditions that would produce breakdowns and malfunctions in machines, made for some wonderful music. The circuits would have a "nervous breakdown" and afterwards they would be very relaxed, and it all came through in the sounds they generated.

I circuiti in retroazione erano destinati al corto circuito, e utilizzati appositamente come materiale per la generazione acustica di trame incise su nastro. Altri importanti compositori di quel periodo sono al lavoro su composizioni che sfruttano sistemi di feedback, fra i più rilevanti: John Cage, David Tudor, Robert Ashley, Gordon Mumma e Steve Reich.[13]

Fra questi David Tudor è stato uno dei primi musicisti a montare uno dei primi esempi di un sistema strumentale basato su feedback complessi, per la realizzazione di Variations II (1960) di John Cage, mentre Gordon Mumma nel 1966 si aggrega alla compagnia di Merce Cunningham per il quale compone Mesa, una prima improvvisazione "Cybersonic" per David Tudor; da lì in poi comporrà importanti brani e improvvisazioni cibernetiche negli anni a venire, come ad esempio Hornpipe (1967), che nasce dall'interazione fra performer, uno strumento cibernetico e la risonanza della sala di esecuzione: al corno francese e' collegata una console che si sintonizza sul feedback della sala e reagisce ai suoni emessi dallo strumento.

Un secondo periodo costituito da un approccio sistemico più consapevole che inizia a tracciare la strada per un pensiero ecosistemico della composizione, in un certo senso si può affermare che inizi dal lavoro di Alvin Lucier, che nel '69 scriverà quello che sarà un brano emblematico per le cibernetiche in musica *I'm sitting in a room*, che è un brano importante per quelle che sono le logiche di interazione sistemiche fra uomo/macchina/ambiente che approfondiremo più avanti nel corso della tesi, e che proprio come nel passaggio da cibernetica di primo ordine a quella di secondo ordine di cui parla Heinz Von Foerster, viene sancita l'interazione sistemica dove il musicista l'ambiente e lo strumento divengono parti di un insieme complesso; il

[13]Sanfilippo and Valle, "Feedback Systems: An Analytical Framework".

sistema stesso di *I'm sitting in a room* si osserva tramite l'ambiente nel corso della sua evoluzione, tracciando una storia di relazioni e interazioni con questo. Nel brano di Lucier, un performer recita in un microfono un testo che descrive il fenomeno che avverrà poco a poco, la voce recitante nel microfono viene registrata e poi riprodotta da altoparlanti posti nella stanza, il suono della registrazione riprodotta da questi altoparlanti viene poi registrato nuovamente durante la riproduzione da un secondo supporto, e l'operazione viene ripetuta con questa casualità circolare di volta in volta fino alla fine, dove rimarranno solamente: i contributi provenienti dalle frequenze di risonanza della stanza, dalla voce, e dalla catena elettroacustica, dando vita nel loro insieme ad un processo molto lento di feedback positivo dove la natura stessa non lineare del processo e degli agenti porterà di volta in volta ad un risutato sempre differente. Lucier racconta di essere stato originariamente ispirato a creare I Am Sitting in a Room dopo aver partecipato a una conferenza al MIT, in cui Amar Bose, imprenditore, ingegnere elettrico e tecnico del suono, descrisse come testava le caratteristiche degli altoparlanti che stava sviluppando, ri-diffondendo l'audio prodotto nella stanza e poi riprendendolo tramite i microfoni, in quella che è la stessa casualità circolare ripresa ed utilizzata artisticamente da Lucier.[14] Dopo l'esperienza di Lucier, Nicolas Collins, formatosi nella tradizione compositiva sperimentale con: Alvin Lucier, David Behrman e David Tudor, con i quali ha lavorato a stretto contatto, nel '74 compone il brano *Pea Soup* mentre è studente alla Wesleyan University. In *Pea Soup* una rete auto-stabilizzante di circuiti analogici (originariamente tre Countryman Phase Shifter) sposta il tono del feedback elettoacustico su una frequenza di risonanza diversa ogni volta che questo si inizia a costruire. Il suono familiare del fenomeno è sostituito da schemi instabili che danno vita ad un raga site-specific rispecchiando la personalità acustica della stanza.[15] Anche in questo lavoro appare chiaro come una certa sensibilità nei confronti del fenomeno di feedback si stia avvicinando ad una logica più vicina a quella della cibernetica

[14]Massachusetts Institute of Technology. *Alvin Lucier on "I am sitting in a room"*. URL: https://www.youtube.com/watch?v=v9XJWBZBzq4. (accessed: 04.11.2022).

[15]Nicolas Collins. "'Pea Soup' – A History". In: (September, 2011), pp. 1–23. URL: http://www.nicolascollins.com/texts/peasouphistoryOLD.pdf.

di secondo ordine. Per citare un'ultima esperienza americana rilevante, c'è infine il caso della neural synthesis di David Tudor. David Tudor nel '89 incontrò il progettista e designer Forrest Warthman dopo uno show a Berkeley, che che lo introdusse all'idea di utilizzare reti neurali analogiche per combinare tutta la sua complessa attività live-electronics in un unico computer.[16] Uno dei risultati è apprezzabile nei CD's "Neural Synthesis", in uno di questi sono presenti delle importanti note di copertina di Warthman:

This recording combines the art of music, the engineering of electronics, and the inspiration of biology. In it, David Tudor orchestrates electronic sound in ways analogous to our biological bodies' orchestration of consciousness... The neural-network chip forms the heart of the synthesizer. It consists of 64 non-linear amplifiers (the electronic neurons on the chip) with 10240 programmable connections. Any input signal can be connected to any neuron, the output of which can be fed back to any input via on-chip or off-chip paths, each with variable connection strength. The same floating-gate devices used in EEPROMs (electrically erasable, programmable, read-only memories) are used in an analog mode of operation to store the strengths of the connections. The synthesizer adds R-C (resistance-capacitance) tank circuits on feedback paths for 16 of the 64 neurons to control the frequencies of oscillation. The R-C circuits produce relaxation oscillations. Interconnecting many relaxation oscillators rapidly produces complex sounds. Global gain and bias signals on the chip control the relative amplitudes of neuron oscillations. Near the onset of oscillation the neurons are sensitive to inherent thermal noise produced by random motions of electron groups moving through the monolithic silicon lattice. This thermal noise adds unpredictability to the synthesizer's outputs, something David found especially appealing. The synthesizer's performance console controls the neural-network chip. R-C circuits, external feedback paths and output channels. The chip itself is not used to its full

[16] Mathis Nitschke. *Analog Neural Synthesis*. URL: https://mlure.art/analog-neural-synthesis/. (accessed: 04.11.2022).

potential in this first synthesizer. It generates sound and routes signals but the role of learner, pattern-recognizer and responder is played by David, himself a vastly more complex neural network than the chip Neural Synthesis No.6-9 liner notes by

Forrest Warthman, Palo Alto 1995[17]

Ad oggi svariati compositori a partire dalle trame delineate dalle scienze complesse e dai lavori citati, operano nell'ambito della musica elettronica con un approccio sistemico, e fra questi molti sono italiani.

Ci sono casi particolarmente rilevanti come quello di Agostino Di Scipio, uno dei maggiori compositori con più contributi all'attivo, da prima con i suoi studi sul caos e sui sistemi complessi in modo formale ad inizio anni '90[18], e poi verso metà anni '90 con l'inizio del lavoro sulla composizione ecosistemica con il suo ciclo di lavori - ecosistemico udibile, che come vedremo a seguito riprende quelle che sono le tematiche della cibernetica di secondo ordine dove il sistema si osserva tramite l'ambiente circostante. O dell'(ex)allievo di Di Scipio, Dario Sanfilippo, compositore e ricercatore con all'attivo recenti importanti pubblicazioni e lavori nell'ambito dei sistemi autonomi DSP in musica, tematiche che rimandano alle proposte che vanno dai Barron fino alla Neural Synthesis di Tudor, seppure in forme più primordiali nel caso di questi primi pionieri, e più avanzate nel caso di Sanfilippo e Di Scipio; Di Scipio nella sua sensibilità ecosistemica in effetti sembra rimandarci ad una fase più avanzata di questi lavori pionieristici di Lucier e Collins.

C'è poi il caso di Michelangelo Lupone, che è stato a sua volta maestro di Agostino Di Scipio, e che con i suoi lavori di feedback sulla materia è arrivato durante il corso degli anni '90 allo sviluppo pionieristico di strumenti aumentati in feedback, quale ad esempio il Feed-Drum, innovativo strumento elettroacustico a percussione.

E ci sono poi anche altri compositori internazionalmente riconosciuti con all'attivo composizioni e ricerche rilevanti in lavori con il feedback e i sistemi autonomi, come:

[17] **http://www.lovely.com/albumnotes/notes1602.html**.

[18] Agostino Di Scipio. "Composition by exploration of non-linear dynamic systems". In: *ICMC Glasgow 1990 Proceedings*. C.S.C. University of Padova, 1990, pp. 324–327.

Andrea Valle, Giuseppe Silvi, Simone Pappalardo.

Parlando delle questioni italiane e tornando all'origine delle questioni romane, nonostante la natura frammentaria e sottile della musica elettronica romana, in effetti è comunque possibile individuare e tracciare una sorta di collegamento che ci porta sin dalle prime suggestioni sulla cibernetica avute da Evangelisti con gli altri membri di Nuova Consonanza, fino ad oggi. Michelangelo Lupone ad esempio che abbiamo citato per i suoi lavori e per esser stato maestro di Agostino Di Scipio, studia dal '70 al '79, sotto la guida di Domenico Guaccero per la Composizione e Giorgio Nottoli per la Musica elettronica. Domenico Guaccero è a sua volta fra i fondatori, insieme ad Evangelisti ed altri compositori quali Aldo Clementi, Daniele Paris, Francesco Pennisi, dell'Associazione di Nuova Consonanza. Walter Branchi, noto anche lui per aver preso parte al Gruppo di Improvvisazione Nuova Consonanza, durante gli anni '80 darà vita a degli incontri internazionali su Musica complessità, che radunavano compositori e scienziati di tutto il mondo. Di Scipio e Lupone in questo scenario, si sono interessati alle questioni sul feedback in contemporanea portando avanti il discorso in maniera indipendente intorno alla fine degli anni '80, Di Scipio nel 1989, scrive gli appunti a base di semplici funzioni iterate da cui nacque poi il suo brano *Fractus*, per viola e supporto digitale. Lupone dal 1988, fonda ed inizia con il Centro Ricerche Musicali il suo lavoro con team multidisciplinari di ricerca, con la collaborazione di persone come Lorenzo Seno direttore scientifico del CRM. Tutto questo mette in luce come le problematiche relative alla cibernetica e i sistemi complessi siano stato un argomento molto sentito e vivo nella prassi della composizione elettroacustica in tutto il mondo.

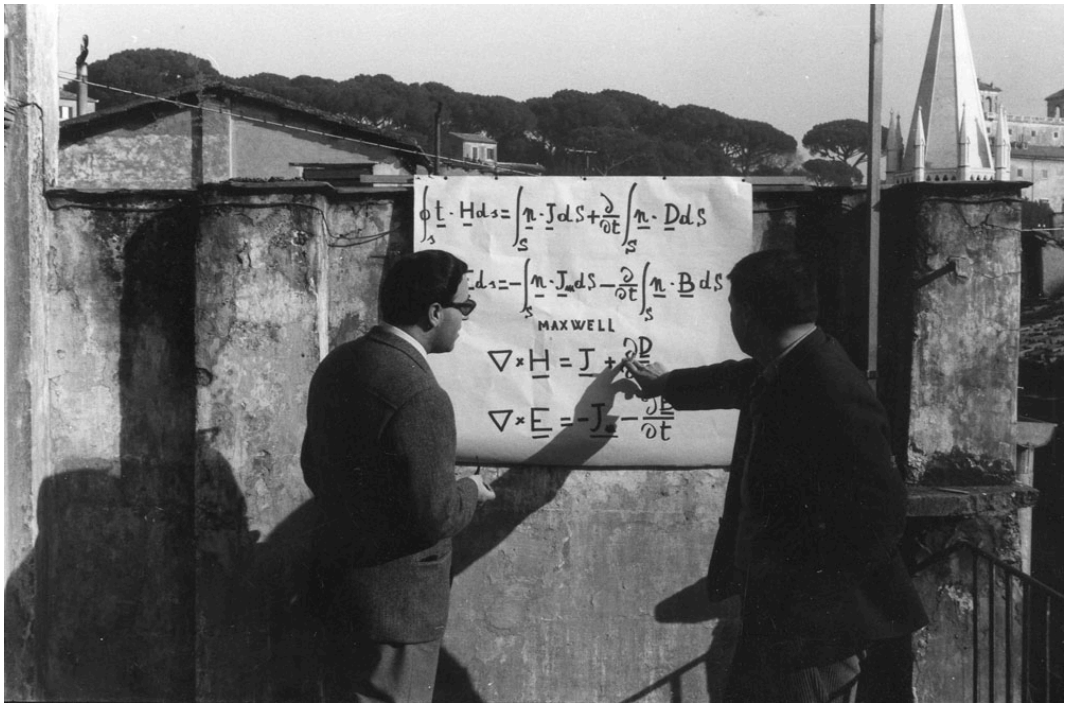

Figure 1: Franco Evangelisti e Franco Nonnis sulla terrazza di Vittorio Consoli che osservano le Equazioni di Maxwell, 1959

1.3 Il Feedback

Il feedback, nella sua espressione inglese, unisce le due parole *Feed* che significa 'alimentare', e *Back* 'all'indietro'. Il primo a introdurre il concetto di feedback nella letteratura scientifica è stato lo scienziato inglese James Clerk Maxwell, il quale nel 1868 pubblicò uno studio[19] sui sistemi automatici, in cui metteva in evidenza come essi fossero in grado di correggersi in maniera automatica proprio grazie al concetto del *ritorno di informazione*. Maxwell anticipò quel concetto che arriverà a maturazione solo molti anni dopo con la nascita della cibernetica e con Norbert Wiener, che diede il nome di cibernetica al campo dei suoi studi proprio in onore del lavoro di Maxwell - dal greco *kybernetiké téchne* sta per 'arte del governare', a cui si collega il termine inglese *governor* dell'articolo di Maxwell. In italiano il significato del termine può essere tradotto come 'retro-alimentazione' o 'retro-iniezione' che ne

[19]Mr. J. C. Maxwell. "on Governors". In: (1868).

preservano in qualche modo il significato originale della lingua inglese, tuttavia in molti preferiscono conservare direttamente la parola inglese feedback.

Nella teoria cibernetica, e in genere in tutte le discipline scientifiche che usano approcci di tipo sistemico, quando il Feedback ha un effetto risultante dall'azione di un sistema (meccanismo, circuito, organismo ecc.) che si riflette sul sistema stesso per variarne e correggerne opportunamente il funzionamento, questo si può chiamare Retroazione.

Esistono sostanzialmente due tipi di retroazione nel feedback: *positiva* e *negativa*. Il feedback è generalmente detto negativo quando agisce in modo che il sistema funzioni sempre allo stesso modo, entro margini controllati di tolleranza e convergendo verso un comportamento, viene invece detto positivo quando tende a mantenere il sistema in uno stato di continuo cambiamento. Tuttavia queste definizioni possono variare da caso a caso, e nel corso della tesi ritorneremo su questi temi riguardo le possibili proprietà del feedback, affrontando queste di volta in volta nei casi specifici. Per anticipare da ora alcuni esempi: ci basti pensare che nel controllo di un sistema complesso come può essere quello del feedback elettroacustico, in un modello dove si possono introdurre delle linearità tramite retroazione all'interno del ciclo di feedback, questo potrebbe ad esempio voler dire costringere la complessità a dei comportamenti più prevedibili, puntando uno stato di convergenza verso l'equilibrio. Un esempio conosciuto è quello dell'intonazione del fenomeno Larsen, che da un comportamento complesso della sorgente e del ricettore, crescendo verso la saturazione, per autoscillazione giunge infine ad uno stato stazionario, di stabilità. Mentre introdurre delle non linearità nel sistema tramite la retroazione, al contrario, può voler dire portare il sistema verso comportamenti non più prevedibili, in divergenza dall'equilibrio dello stato stazionario, e in alcuni casi verso quella che viene chiamata la soglia del caos. Questi due tipi di comportamento possono essere ottenuti per l'appunto sia velocizzando che rallentando questi processi in maniera fortemente dipendente dal caso specifico. Un secondo esempio può essere invece quello dei filtri digitali audio, pensati come un valido strumento di contrasto o avallamento rispetto

a questo tipo di comportamenti, dove se si allineano le fasi si creano dei poli, mentre se si disallineano si punta generalmente alla complessità del sistema. Riguardo i filtri digitali basti pensare anche come questi possono essere utilizzati negli algoritmi di modellazione fisica degli strumenti, come ad esempio quello famoso di Karplus-Strong. Di fatto, concludendo questa introduzione con una visione più generale, la storia delle tecnologie elettroacustiche ha da sempre incorporato il principio del feedback sin dalle sue origini e da prima della nascita della cibernetica. Possiamo pensare a tecnologie come il triodo, chiamato inizialmente valvola audion di Lee De Forest, o i circuiti di feedback negativo - negative feedback amplifier - di Harold Black[20]. In sintesi, il principio del feedback continua a essere un pilastro fondamentale nella storia e nell'evoluzione delle tecnologie elettroacustiche, fino ad essere diventato un aspetto integrato e non più occasionale nella prassi dei compositori musicali elettroacustici.

[20]H.S. Black. "Stabilized Feed-Back Amplifiers". In: *ALEE. committee on communication* (1933).

2 Ecosistemi Udibili

2.1 Audible EcoSystemics n.2 - Feedback Study

Uno degli esiti musicali più importanti che ha portato con sé il paradigma della complessità, riguarda essenzialmente l'aver smesso di scrivere musica per strumenti interattivi con cui farla (e viceversa), ed esser passati invece al comporre le interazioni tramite gli strumenti.

Da un punto di vista più generale, Heinz Von Foerster in tal senso sviluppò una teoria dell'informazione secondo cui l'informazione è un costrutto relazionale, e non solo un singolo oggetto o un solo segnale, e che di conseguenza la quantità di informazione in un sistema è determinata quindi dalla sua capacità di generare varietà. Le implicazioni e le conseguenze di questo pensiero applicato alla musica sono enormi. Nella nostra tradizione musicale occidentale, nell'insieme delle parole: Sistema Tonale, che sta a significare "quell'insieme delle regole di organizzazione ed uso dei suoni maggiormente utilizzati per produrre musica" (sempre rimanendo nel contesto della tradizione musicale Occidentale), è frequente che passi in secondo piano la parola sistema, che assume invece un significato centrale nel contesto di questa tesi; se si pensa che di conseguenza passa in secondo piano l'insieme di quelle relazioni che compongono nella sua interezza un sistema, di cui parla Von Foerster, si riduce la pratica musicale ai minimi termini. Non è un caso che molti compositori nel corso del XX Secolo, siano passati non solo al comporre i sistemi stessi con cui fare la propria musica, ma al comporne anche le interazioni. Si pensi ai sistemi formalizzati come partiture da: Iannis Xenakis, John Cage, o Roland Kayn, arrivando fino alle partiture comportamentali come quelle di: Domenico Guaccero e del contesto generale della Roma del dopoguerra, per concludere con lavori di Karlheinz Stockhausen come Plus-Minus, dove i processi sono udibili. Tornando al tema principale, come accennato nell'abstract, le interazioni che ho ritenuto importante affrontare all'interno dei sistemi creati dai compositori cibernetici sono di due tipi: di sistemi che interagiscono con l'ambiente circostante appartenente al mondo fisico, e di sis-

temi che interagiscono con il proprio ambiente nel mondo digitale. Prima di entrare nel merito del primo caso qui, è doveroso spendere ancora qualche parola su cosa si intenda per ambiente.

Nessun sistema è separabile e isolabile dall'ambiente circostante, a prescindere dal fatto che il suo spazio vitale, sia nel mondo fisico o digitale. Proprio come ha mostrato Heinz Von Foerster non si può parlare di auto-organizzazione se non ci si riferisce essenzialmente a un ambiente che racchiuda il sistema al suo interno. E nella pratica musicale, la sensibilità nel comporre le interazioni è spesso lontana dall'idea del voler interagire con una consapevolezza effettiva del puntare ai cambiamenti di stato all'interno di un sistema con cui si interagisce; a maggior ragione viene trascurata la possibilità di interagire con sistemi che abbiano la capacità di auto-osservarsi non dipendendo più direttamente dal controllo dell'esecutore secondo delle modalità prettamente lineari, ma dall'ambiente, con la capacità di guardare al proprio stato interno per creare autonomamente le interazioni. In particolare, nei sistemi auto-osservanti quello che si manifesta quando un sistema entra in interazioni non distruttive con il proprio ambiente circostante, che coincide nella sostanza anche nel suo spazio vitale, è un Ecosistema.

In tal senso, i lavori del ciclo Ecosistemico Udibile di Agostino di Scipio, trovano fondamento a partire da fenomeni e relazioni che possono esistere e manifestarsi solamente nell'ambiente circostante da cui prende vita il sistema, che nel suo caso diventa proprio lo spazio acustico reale. Lavori come lo studio sul feedback, lo studio sul rumore di fondo, lo studio sulle risposte all'impulso, o lo studio sul silenzio, sono nella loro essenza dei sistemi che hanno come principio nella loro morfogenesi, un determinato comportamento appartenente allo spazio acustico reale, delimitato in questi studi da una stanza che ne racchiude al suo interno tutti gli agenti e le relazioni possibili, per costruirne una storia di relazioni dove il sistema si osserva attraverso lo spazio fisico, (ambiente) e si manifesta e vive solo attraverso di esso. In questo senso l'interprete, lo spazio, e gli ascoltatori, divengono essi stessi parte integrante del sistema, dove l'ascolto, diviene anche esso parte dell'insieme delle cose

e delle relazioni che lo costituisce.

2.2 L'interazione Uomo-Macchina-Ambiente

Qual'è dunque l'esigenza alla base del comporre le interazioni invece che limitarsi al comporre musica per strumenti interattivi? Superare la classica relazione uomo-macchina dominante nella tradizione musicale, iniziando a pensare invece allo sconfinato universo delle nuove possibilità della complessità. Secondo Agostino Di Scipio, questo avviene in primo luogo grazie alla possibilità che la macchina possa rappresentarsi senza mediazione umana, e attraverso l'ambiente circostante,[21] e dunque poi alla possibilità di stabilire un flusso di relazioni macchina-ambiente. Consentendo in secondo luogo al musicista e performer la possibilità di potersi aprire ad un flusso di relazioni complesso fra uomo-macchina-ambiente dove le tre sono fortamente connesse e interdipendenti l'una dall'altra.

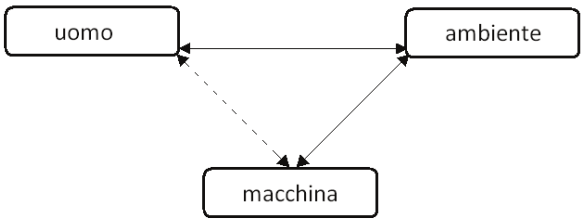

Figure 2: Interazione Uomo-Macchina-Ambiente

la mossa decisiva è: passare da un lavoro che mira a usare mezzi interattivi per creare forme sonore desiderate ad un lavoro che mira a creare le interazioni desiderate e ad ascoltarne le tracce udibili. Nel secondo caso, si tratta di progettare, implementare, e rendere operativo un reticolo di componenti interconnesse il cui comportamento sonoro emergente si può chiamare musica.[22]

[21] Agostino Di Scipio. "Polveri Sonore - Una prospettiva ecosistemica della composizione". it. In: *La Camera Verde* (2014). Publisher: La Camera Verde, 17–42 Pages. (Visited on 11/22/2022).
[22] Di Scipio, "Polveri Sonore - Una prospettiva ecosistemica della composizione".

La modalità con cui andrò a discutere in questo capitolo la composizione di interazioni ecosistemiche, è ponendo il focus su un lavoro di Agostino Di Scipio, l'Ecosistemico Udibile n.2, studio sul feedback. Partendo dalla partitura e dai suoi scritti, ed implementando e analizzando il ruolo ed il compito dei singoli agenti all'interno del sistema che compongono nella loro totalità l'intero ecosistema del brano.

2.3 Il Meccanismo LAR

Il feedback elettroacustico, che abbiamo già detto esser stata una delle risorse fondamentali dei primi compositori cibernetici, è la condizione di partenza su cui Agostino Di Scipio opera per la costruzione dei suoi Ecosistemi Udibili. Addentrandoci ora verso una spiegazione più tecnica del feedback elettroacustico, possiamo citare una definizione che Di Scipio ha esposto in un suo articolo pubblicato presso la rivista Online: ECHO dell' Orpheus Institute, in Ghent.[23]

A condenser microphone (M1) and a dynamic loudspeaker (L1) stand in the performance place (S), few or several meters apart, maybe not too far from walls (or curtains, or other larger surfaces). They are connected (through one or more amplification stages) to realize a very basic electroacoustic chain: M1→L1→S. There's no sound M1 should capture, though, no sound source save the minimal, barely audible turbulence of the background noise, in a situation of 'silence'. This 'sound-of-nothing' is amplified and heard through L1, whence it comes back in S. If amplification suffices, the L1 sound feeds back into M1 and the chain design closes onto itself, making a 'reinjection' circuit – a feedback loop. The amplitude level, the transductive technical features of M1 and L1, their relative distance, the distance from walls, etc. – all of that (and much more) sets the actual feedback loop gain. With not-too-high gain levels, what is engendered is an audible nuisance,

[23] Agostino Di Scipio. *A Relational Ontology of Feedback*. en. Jan. 2022. DOI: 10.47041/TKUL7223. URL: https://echo.orpheusinstituut.be/article/a-relational-ontology-of-feedback (visited on 11/22/2022).

a kind of 'halo': the sound reinjection decays more or less rapidly, in a kind of badly sounding, spectrally uneven reverb effect. With higher gain levels, the loop eventually enters a self-oscillatory regime, it may 'ring' or 'howl', as is often said. Because of the iterated reinjection, the barely audible but spectrally wide background noise accumulates in the loop and finally (quickly) yields an increasingly louder sustained sound of narrower spectrum – this is often heard as a peaking tone of definite pitch, or a tone cluster. That's the Larsen effect: a self-sustaining feedback resonance occasioned by a positive feedback loop (FB+) ('positive' here means greater than unit gain).

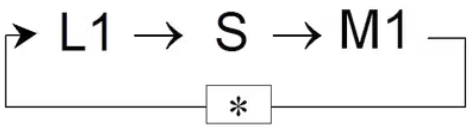

Figure 3: Schema M1→L1→S

L'effetto Larsen: (dal nome del fisico Søren Absalon Larsen che per primo ne scoprì il principio), detto anche feedback elettroacustico, come abbiamo appena letto è un fenomeno di retroiniezione che tende idealmente ad un'accumulazione infinita, che viene poi limitata in realtà dalla saturazione dei sistemi che la generano (relativi alla potenza massima, all'amplificazione, nonché alla sensibilità dei trasduttori e all'elasticità delle membrane). Che può anche essere oltre al microfono, un pick-up di uno strumento musicale elettrico, come una chitarra o un basso, o un trasduttore di altra natura... Esistono anche principi di feedback acustico oltre che elettroacustico, come la risonanza acustica o la risonanza per simpatia, che sono i principi su cui si basa il funzionamento di quasi tutti gli strumenti musicali.
Tornando quindi all'articolo di Agostino Di Scipio:

In common sound engineering practice, audible feedback phenomena are a nuisance,

a problem one should get rid of or substantially minimize. When direct level manipulation is not enough, one resorts to hard-limiting circuits, 'feedback killers' and alike devices... In a different attitude, one may instead consider feedback as a resource, a deliberately designed sound-making mechanism one can play with.[24]

per utilizzare il feedback come una risorsa, ora appare chiara la necessità di un intervento sulla sua retroiniezione a guadagno infinito che porta alla saturazione dei sistemi coinvolti, in tal senso una delle possibili soluzioni è quella di poter far calcolare al computer in tempo reale tramite diverse tecniche di *amplitude following* la stima dell'ampiezza del segnale in ingresso, ed utilizzare conseguentemente la *feature extraction* come segnale di controllo in retroazione negativa al sistema di feedback elettroacustico.

Questo tipo di meccanismi, provenienti dalla tradizione della Computer Music, appartiene all'ambito del *Music Information Retrieval* (MIR). Il MIR è un campo di ricerca multidisciplinare, che nell'area dell'informatica musicale si occupa di studiare i metodi e gli strumenti per l'estrazione dell'informazione musicale per mezzo di un computer.

Oltre alle analisi in tempo differito, che sono tipicamente trattate dal MIR per le sue applicazioni di ricerca dove viene presa in considerazione l'intera lunghezza del segnale su cui si effettua un analisi, oggi grazie ad alcuni linguaggi di programmazione adatti alle implementazioni per il tempo reale e utili alle performance in Live Electronics, è possibile operare anche alcune di queste analisi in tempo reale, fra cui per l'appunto la stima dell'ampiezza di un segnale in ingresso. L'applicazione in tempo reale di una *feature extraction* Nel contesto di una performance di live electronics, viene chiamata col nome di *Audio Information Processing*; tanto più un sistema è in grado di raccogliere informazioni riguardo il suo stato e il suo rapporto con l'ambiente, tanto più sarà in grado di processare "cognitivamente" tramite l'informazione il suo prodotto.

Rimandando al prossimo capitolo approfondimenti più dettagliati riguardo il tema

[24] Di Scipio, *A Relational Ontology of Feedback*.

dell'*Audio Information Processing*, mi fermerò per il momento a discutere nel dettaglio alcuni fondamentali meccanismi presi da questo brano che sono paradigmatici, come il singolo meccanismo di controbilanciamento del guadagno del fenomeno di feedback elettroacustico, che viene chiamato da Agostino Di Scipio col nome di LAR: *Audio feedback with self-regulated Gain*, e che può essere implementato in DSP in diverse modalità e configurazioni, ognuna con le sue diverse caratteristiche e che porta a differenti esiti.

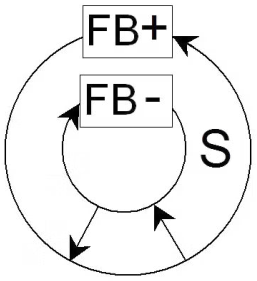

Figure 4: Schema del meccanismo LAR

Ci sono in effetti diversi modi per ottenere una *feature extraction* in tempo reale capace di realizzare un algoritmo di controbilanciamento del feedback in tempo reale. Alcuni di questi metodi possono riguardare controbilanciamenti applicati nel dominio della frequenza, con tecniche di filtraggio automatizzate (adattive) che hanno il compito di eliminare dallo spettro la presenza dell'autoscillazione prodotta dal Larsen *Larsen Suppressors*, o come nel nostro caso d'interesse possono riguardare controbilanciamenti in ampiezza automatizzati, che non permettano al feedback di avere un guadagno troppo troppo elevato e giungere conseguentemente allo stato di saturazione del sistema. Algoritmi di *amplitude-following* possono essere implementati con la media *RMS*, con finestre variabili o fisse, filtri *Peakholder* che mantengano il valore di picco, o di altra natura.

Tratterò a partire da ora nella tesi anche delle parti più operative, discutendo ed illustrando l'implementazione di alcune di queste tecniche nel linguaggio di program-

mazione FAUST (Grame), che è uno dei linguaggi di programmazione che consente di fare *Audio Information Processing* in tempo reale. Faust è l'ambiente in cui ho scritto i codici di tutti i lavori trattati in questa tesi e le relative compilazioni, diagrammi e softwares. [25][26]

Vorrei qui esprimere la mia profonda gratitudine a Dario Sanfilippo per il prezioso supporto che mi ha fornito durante lo sviluppo di molti di questi codici, e poiché nell'ultimo anno ha amorevolmente condiviso con me il suo sapere e le sue tecniche sviluppate nel corso dei suoi studi e delle sue ricerche, spiegandomi sempre con molta pazienza ogni dettaglio e applicazione di queste nei auoi sistemi complessi.

Il modo più semplice per mantenere l'effetto Larsen in uno stato stazionario, è attraverso un valore costante ricavato dalla crescita del segnale in ingresso, che controbilanci l'ampiezza del circuito di feedback come in figura del meccanismo LAR; la stima di questa costante avviene tramite un algoritmo di analisi. Il modo più semplice per implementare un algoritmo di analisi di questo tipo è attraverso un *Peakholder* che nella sua forma basilare consiste in una finestra di campioni idealmente infinita IIR *Infinite Impulse Response*, e che può essere espresso matematicamente come

$$y_1(t) = \max\left(y_1(t-1), |x(t)|\right)$$

dove x è il segnale in ingresso, e y il segnale sia in uscita che reiterato nella funzione.

E utilizzato come algoritmo di controbilanciamento può essere espresso matematicamente come

[25] FAUST (Grame) (Functional Audio Stream), è un linguaggio di programmazione specifico per il Digital Signal Processing sviluppato da Yann Orlarey, Dominique Fober, e Stephane Letz nel 2002. Nello specifico, FAUST è un linguaggio di programmazione ad alto livello scritto in C++, che permette di tradurre delle istruzioni date e create appositamente per il digital signal processing (DSP), in un largo raggio di linguaggi di programmazione non specifici per il dominio dell'Audio Digitale.

[26] **https://faust.grame.fr/**.

$$y_1(t) = (1 - (\max(y_1(t-1), |x(t)|))) * x(t)$$

questo algoritmo ha il compito di confrontare il valore assoluto del campione in ingresso con il suo precedente, e Il maggiore fra i due nella comparazione viene mandato sia in uscita che in ingresso in retroiniezione alla funzione stessa di comparazione, in questo modo si sfrutta un principio di feedback per l'accumulazione del valore massimo, in un'analisi campione per campione.

```
// import faust standard library
import("stdfaust.lib");

// Peak Max with IIR filter and max comparison
peakmax = loop
    with{
        loop(x) = \(y).((y , abs(x)) : max) ~ _ ;
    };

process = _ : peakmax;
```

Listing 1: Algoritmo del *Peakholder* ad 1 campione di ritardo

```
// import faust standard library
import("stdfaust.lib");

// LAR with Peak Max - IIR filter and max comparison
peakmax = loop
with{
    loop(x) = \(y).((y , abs(x)) : max)~_;
};

LARpeakmax = _ <: (_ * (1 - (_ : peakmax)));
```

```
process = _ : LARpeakmax;
```

Listing 2: Algoritmo del LAR con *Peakholder* ad 1 campione di ritardo

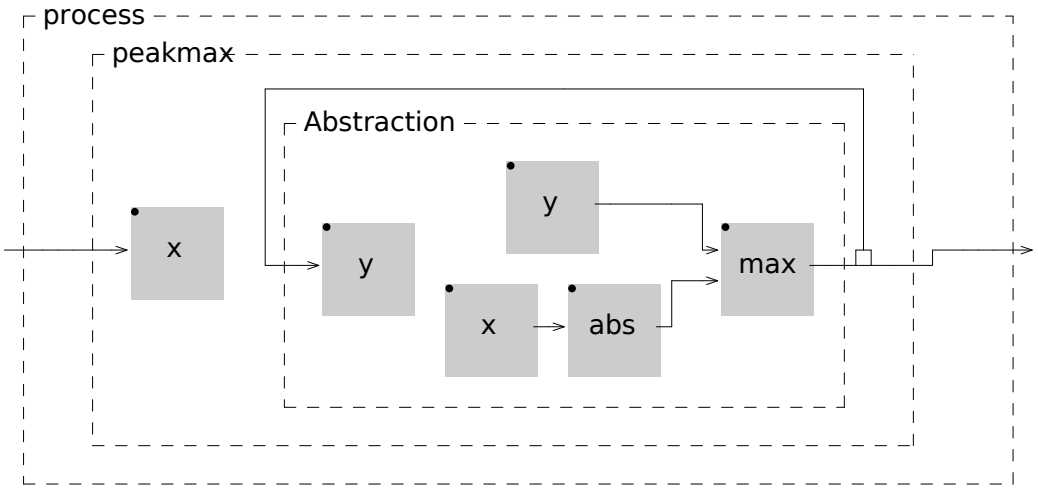

Figure 5: Topologia del *Peakholder* ad 1 campione di ritardo

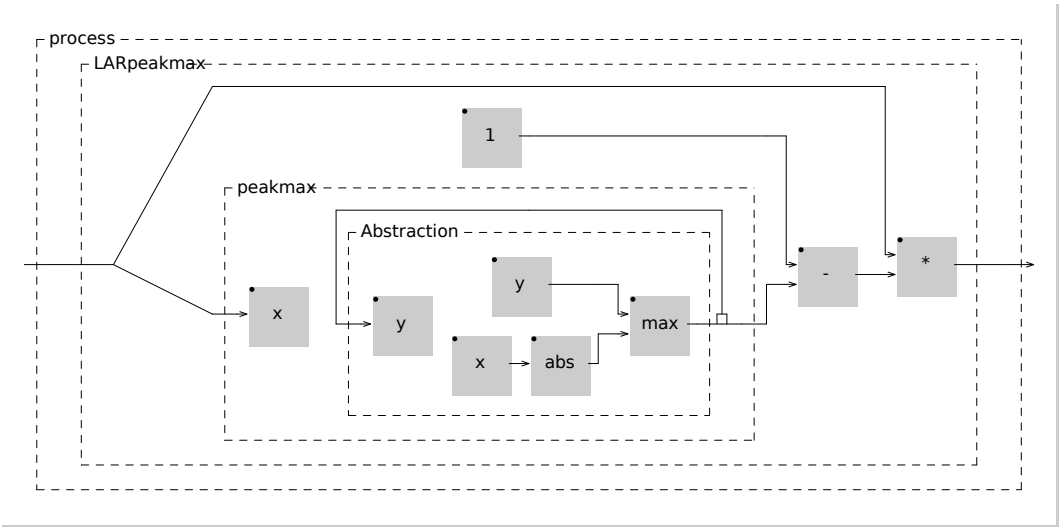

Figure 6: Topologia del LAR con *Peakholder* ad 1 campione di ritardo

Questo tipo di algoritmo presenta comunque alcuni problemi: non avendo una funzione di smoothing del segnale, si verificano problemi di segnali di differenza a banda molto larga che possono generare aliasing e contributi spuri che tendono a permanere nel segnale complessivo. Oltre a questo, variazioni del comportamento

del feedback sono dipendenti dalla grandezza della finestra di osservazione, da eventuali cofficenti di feedback inseriti nella retroazione del *Peakholder* e da altri tipi di implementazioni di tecniche *amplitude-following*. Per questi motivi si sceglie più frequentemente di utilizzare algoritmi adattivi, proprio come nel contesto del meccanismo LAR implementato nel feedback study, dove il comportamento adattivo dell'*amplitude-follower* è cruciale per la dinamica adattiva ed auto-regolatoria del sistema.

Il meccanismo LAR è essenzialmente nel cuore della live performance del feedback study, senza di questo non sarebbe possibile creare una condizione favorevole per procedere alle conseguenti trasformazioni del suono, e conseguentemente ad una condizione favorevole affinché il sistema possa osservarsi tramite l'ambiente circostante. In tal senso vale la pena procedere osservando da vicino come viene richiesto in partitura di implementare questo meccanismo.

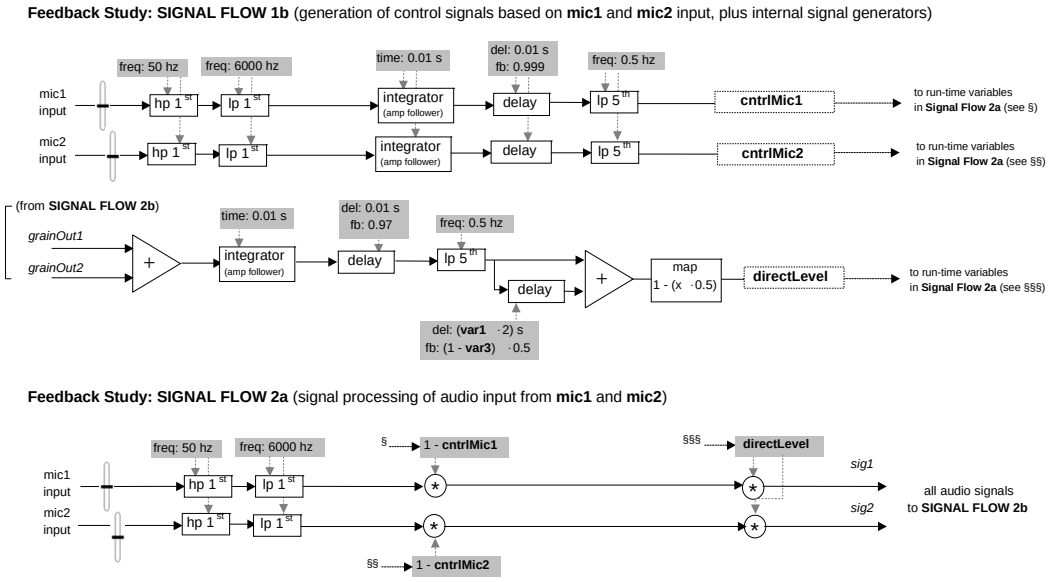

Figure 7: Estratto dalla partitura di *Audible Ecosystemics n.2/ feedback study (2003)* revisione del 2017 - Agostino Di Scipio

Come scritto in partitura, è da notare in prima istanza come il microfono serva sia da sorgente per l'alimentazione del Larsen che da canale che riceve l'informazione necessaria per il meccanismo auto-regolatorio, fin qui è tutto uguale rispetto allo

Schema del meccanismo LAR. La sostanziale differenza consiste invece nell'aggiunta di un secondo microfono, quando viene introdotta questa condizione di disaccoppiamento si permette al meccanismo di essere più responsivo ai piccoli cambiamenti nell'ambiente e di rispondere diversamente agli stessi stimoli da diversi punti di osservazione del sistema. Nel Signal Flow 1b, due microfoni in ingresso (mic1 e mic2) vengono limitati in banda da un filtro Highpass ed un filtro Lowpass, rispettivamente fra 50 e 6000 Hz; dei due microfoni che escono dal cntrlMic1 e cntrlMic2 ne viene ricavato l'inviluppo d'ampiezza tramite l'oggetto *integrator*, poi la curva di questo inviluppo viene amplificata da un delay con feedback, e ne viene estratta solamente la componente energetica presente fra i 0 e i 0.5Hz, limitandone il comportamento dell'oscillazione in quel range. Più in basso, nel Signal Flow 2a, i stessi microfoni (mic1 e mic2) sempre limitati in banda fra 50 e 6000 Hz, vengono attenuati in ampiezza dal segnale proveniente dal cntrlMic1 e cntrlMic2, realizzando nella sostanza una versione più sofisticata del controbilanciamento del meccanismo LAR. La parte restante: ovvero, la generazione del segnale directLevel e la conseguente attenuazione dei segnali audio in uscita (sig1 e sig2), consiste in un secondo controbilanciamento ricavato da un processo di granulazione attivo alla fine del sistema, finché il granulatore non produce segnale in output l'ampiezza dei segnali audio diretti in uscita dal sistema rimane inalterata.
L'inviluppo d'ampiezza ricavato tramite l'oggetto *integrator*, in partitura viene richiesto come segue:

integrator = returns the average absolute value over a specific time frame (one may use RMS measures, instead, or other amplitude-following methods); output range is [0, 1]

dove sostanzialmente, viene calcolato il valore assoluto di una specifica finestra temporale. A seguito una implementazione di questo tipo di algoritmo, dove *fi.pole* è un semplice onepole filter nella forma

$$y(n) = x(n) + py(n-1)$$

e *ma.SR* una costante che rappresenta la frequenza di campionamento.

```
// import faust standard library
import("stdfaust.lib");

movingAverage(seconds, x) = x - (x @ N) : fi.pole(1.0) / N
    with {
        N = seconds * ma.SR;
    };

movingAverageRMS(seconds, x) = sqrt(max(0, movingAverage(seconds, x * x)));

process = movingAverageRMS(1);
```

Listing 3: Algoritmo d'integrazione a media mobile

In partitura viene indicato all'esecutore di poter utilizzare diversi metodi per poter calcolare il valore desiderato, come ad esempio l'RMS, tuttavia da analisi dirette secondo l'implementazione del compositore del sistema sul sistema hardware-software KYMA con il suo linguaggio di programmazione dedicato, i filtri utilizzati risultano essere di tipo IIR, così da avere nella sostanza una risposta più lenta rispetto ad altre implementazioni come ad esempio quella del calcolo a media mobile. La risposta del filtro nell'implementazione originale è di tipo *tau* e l'inviluppo è assoluto.

Se si desidera utilizzare questo tipo d'implementazione, nella libreria di Faust è possibile trovarla nell'oggetto `an.abs_envelope_tau`. Nella loro interezza i meccanismi di cntrlMic1 e cntrlMic2, e la conseguente attenuazione del segnale diretto provenente da mic1 e mic2 possono essere quindi implementati in Faust come segue:

```
// import faust standard library
    import("stdfaust.lib");

    LARmechanismAE2(mic1, mic2) = sig1, sig2
    with{
        Mic_1B_1 = hgroup("Mixer", hgroup("Signal Flow 1B",
            gainMic_1B_1(mic1)));
        Mic_1B_2 = hgroup("Mixer", hgroup("Signal Flow 1B",
            gainMic_1B_2(mic2)));

        // cntrlMic - original version
        cntrlMic(x) = x : HP1(50) : LP1(6000) :
            integrator(.01) : delayfb(.01, .995) : LP5(.5);
        cntrlMic1 = Mic_1B_1 : cntrlMic;
        cntrlMic2 = Mic_1B_2 : cntrlMic;

        // from Signal Flow 2a
        Mic_2A_1 = hgroup("Mixer", hgroup("Signal Flow 2A",
            gainMic_2A_1(mic1)));
        Mic_2A_2 = hgroup("Mixer", hgroup("Signal Flow 2A",
            gainMic_2A_2(mic2)));
        micIN1 = Mic_2A_1 : HP1(50) : LP1(6000) *
            (1 - cntrlMic1);
        micIN2 = Mic_2A_2 : HP1(50) : LP1(6000) *
            (1 - cntrlMic2);
        // in the full system this this is a secondary counterbalance
        directLevel = 1;
        sig1 = micIN1 * directLevel;
        sig2 = micIN2 * directLevel;
    };

    process = LARmechanismAE2;


    //------- ------------- ----- -----------
```

```
//-- LIBRARY ------------------------------------------------------------
//------- --------
// selected objects from "aelibrary.lib"


//---------------------------------------------------------- UTILITIES --
// limit function for library and system
limit(maxl,minl,x) = x : max(minl, min(maxl));


//------------------------------------------------------------ FILTERS --
onePoleTPT(cf, x) = loop ~ _ : ! , si.bus(3)
with {
    g = tan(cf * ma.PI * (1/ma.SR));
    G = g / (1.0 + g);
    loop(s) = u , lp , hp , ap
    with {
        v = (x - s) * G;
        u = v + lp;
        lp = v + s;
        hp = x - lp;
        ap = lp - hp;
    };
};

LPTPT(cf, x) = onePoleTPT(limit(20000,ma.EPSILON,cf), x) : (_, !, !);
HPTPT(cf, x) = onePoleTPT(limit(20000,ma.EPSILON,cf), x) : (!, _, !);

// Order with filters in series
LP1(CF, x) = x :LPTPT(CF);
HP1(CF, x) = x :HPTPT(CF);
LP2(CF, x) = x :LPTPT(CF) :LPTPT(CF);
HP2(CF, x) = x :HPTPT(CF) :HPTPT(CF);
LP3(CF, x) = x :LPTPT(CF) :LPTPT(CF) :LPTPT(CF);
HP3(CF, x) = x :HPTPT(CF) :HPTPT(CF) :HPTPT(CF);
LP4(CF, x) = x :LPTPT(CF) :LPTPT(CF) :LPTPT(CF) :LPTPT(CF);
HP4(CF, x) = x :HPTPT(CF) :HPTPT(CF) :HPTPT(CF) :HPTPT(CF);
```

```
LP5(CF, x) = x :LPTPT(CF) :LPTPT(CF) :LPTPT(CF) :LPTPT(CF) :LPTPT(CF);
HP5(CF, x) = x :HPTPT(CF) :HPTPT(CF) :HPTPT(CF) :HPTPT(CF) :HPTPT(CF);

//-------------------------------------------------------- INTEGRATOR --
integrator(sec, x) = an.abs_envelope_tau(limit(1000,.001, sec), x);

//------------------------------------------------------------ DELAYS --
delayfb(delSec,fb,x) = loop ~ _ : mem
with{
    loop(z) = ( (z * fb + x) @(ba.sec2samp(delSec)-1) );
};

//------------------------------------------------ INPUTS/OUTPUTS MIXER --
gainMic_1B_1(x) = x *
    si.smoo( ba.db2linear(
           vslider("SF_1B_1 [unit:db]", 0, -80, 80, .001) ) ) <:
               attach(_, VHmetersEnvelope(_) :
                   vbargraph("VM1B1 [unit:dB]", -80, 80));
gainMic_1B_2(x) = x *
    si.smoo( ba.db2linear(
           vslider("SF_1B_2 [unit:db]", 0, -80, 80, .001) ) ) <:
               attach(_, VHmetersEnvelope(_) :
                   vbargraph("VM1B2 [unit:dB]", -80, 80));
gainMic_2A_1(x) = x *
    si.smoo( ba.db2linear(
           vslider("SF_2A_1 [unit:db]", 0, -80, 80, .001) ) ) <:
               attach(_, VHmetersEnvelope(_) :
                   vbargraph("VM2A1 [unit:dB]", -80, 80));
gainMic_2A_2(x) = x *
    si.smoo( ba.db2linear(
           vslider("SF_2A_2 [unit:db]", 0, -80, 80, .001) ) ) <:
               attach(_, VHmetersEnvelope(_) :
                   vbargraph("VM2A2 [unit:dB]", -80, 80));

VHmetersEnvelope = abs : max ~ -(1.0/ma.SR) :
```

```
        max(ba.db2linear(-70)) : ba.linear2db;
```

Listing 4: Algoritmo LAR nel feedback study

Gli algoritmi degli oggetti richiamati in questa implementazione sono provenienti dalla libreria scritta per questo brano. (sono stati riportati qui solo alcuni oggetti della libreria, per rendere eseguibile il codice appena scritto) Questo codice, che rappresenta il cuore del sistema, permette grazie al meccanismo adattivo una condizione favorevole per mantenere il Larsen in uno stato stazionario senza che giunga a saturazione, e consentendo al sistema di fatto una serie di nuove possibilità di elaborazione e trasformazione del suono.

2.4 Trasformazioni del suono

l'Ecosistemico Udibile n.2, studio sul feedback, è costituito da due principali meccanismi di feedback, uno nello spazio acustico reale, e uno interno al sistema.
Il suono proveniente dal meccanismo LAR appena discusso nella precedente sezione, passa per una serie di elaborazioni numeriche del segnale prima di essere restituito e diffuso dagli altoparlanti nello spazio acustico reale. Queste elaborazioni che vengono poi nuovamente intercettate dal microfono e riportate nel Sistema, consentono una continua ed incessante trasformazione del suono, che permette di cambiarne la morfologia ad ogni istante di tempo e in modo continuo, permettendo al sistema di avere "vita propria" ascoltandosi tramite l'ambiente anche senza il contributo apportato da un esecutore.
Le elaborazioni del suono interne al sistema sono a loro volta di due tipologie, *sample read* e *granular sampling.*
L'implementazione di questi due oggetti viene richiesta in partitura come segue:

sample write = write samples into a memory buffer, in cyclical fashion (wrap-around)

sample read = read samples off the memory buffer, with controls over frequency shift ratios and actual buffer segment being read

granular sampling = read sample sequences off subsequent buffer memory chunks, and envelopes the signal chunk with a pseudo-Gaussian envelope curve; the particular implementation should allow for time-stretching (slower memory pointer increments at grain level), as well as for "grain density" controls and slight random deviations ("jitter") on grain parameters; no frequency shift necessary

dove *sample write* rappresenta una tabella (buffer o dispositivo di memoria) che viene riscritta in modo continuo da un puntatore alla memoria; ciclicamente, e in tutta la sua interezza. La grandezza di questa tabella è data da una delle quattro variabili definite in partitura utilizzate per inizializzare il sistema, *var1* che definito in questo contesto sta ad indicare la grandezza della tabella in secondi. Le 4 variabili definite per inizializzare il sistema ad ogni performance sono le seguenti:

var1 = distance (in meters) between the two farthest removed loudspeakers on the left-right axis.
var2 = rough estimate of the center frequency in the spectrum of the room's background noise (spectral centroid): to evaluate at rehearsal time, in a situation of "silence".
var3 = subjective estimate of how the room revereberance, valued between 0 ("no reverb") and 1 ("very long reverb").
var4 = distance (in meters) between the two farthest removed loudspeakers on the front-rear axis.

queste 4 variabili hanno un ruolo molto importante, poiché non solo vanno a determinare la velocità del comportamento del sistema, ma ne vanno a determinare la sua sensibilità rispetto all'ambiente in cui viene eseguito, come ad esempio nel caso del *sample read* la velocità di lettura dalla tabella *sample write* e la porzione di tabella che ne viene letta.
I *sample read* sono 5 indicati in partitura nella configurazione che segue.

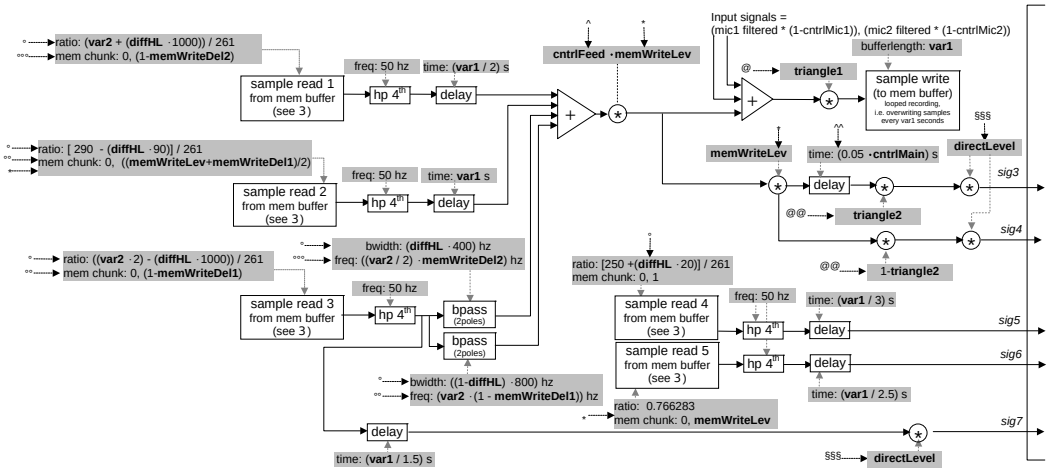

Figure 8: Estratto dalla partitura di *Audible Ecosystemics n.2/ feedback study (2003)* revisione del 2017 - Agostino Di Scipio

Il suono del Larsen proveniente da due microfoni e stabilizzato con il meccanismo LAR come illustrato in precedenza, viene a sommarsi per poi venire moltiplicato per un oscillatore a bassa frequenza che ne modifica l'inviluppo d'ampiezza. Il risultato di questa operazione viene scritto in memoria nella tabella *sample write* e viene letto dai 5 campionatori distribuiti In questa sezione del sistema.

I parametri esterni e tempo varianti che utilizza il campionatore sono: *ratio* e *mem chunk*, che corrispondono rispettivamente alla velocità di lettura e al frammento di buffer che viene letto dalla tabella *sample write*. A seguito di una conversazione con il compositore è risultato essere fondamentale che il campionatore aggiorni i valori di *ratio* ad ogni campione, permettendo modulazioni di frequenza derivate dal cambiamento delle velocità di lettura, che si riflettono in cambiamenti di frequenza del Larsen registrato nel *sample write*. i *ratio* vengono indicati in partitura con delle espressioni diverse per ognuno dei 5 *sample read*, ad esempio: *ratio: (var2 + (diffHL * 1000)) / 261*, dove

ratio value sets the sample-read rate as a function of var2 and the control signal diffHL: ratio = 1 causes no rate change, hence no frequency shift of the sampled signal; values major of 1 determine shifts to higher frequencies; values between 0

and 1 determine shifts to lower frequencies

e dove *diffHL* è un segnale di controllo variabile in un range fra 0 e 1, che a run time a microfoni chiusi parte da una costante di 0.5. La lettura del *ratio* è sensibile rispetto all'ambiente, poiché il valore di *diffHL* che ne determina le modulazioni di frequenza, viene ricavato da una stima della centroide spettale nei segnali di controllo tramite una differenza fra filtri Highpass e Lowpass a cui viene passata come frequenza di taglio la variabile *var2*. Altrettanto sostanziale nel contribuire alle trasformazioni del Larsen è il comportamento del *mem chunk*, che deve permettere una lettura di frammenti di memoria che può andare dal silenzio: *mem chunk: 0, 0* all'intera lunghezza del buffer *mem chunk: 0, 1*. Anche in questo caso la maggiorparte dei campionatori implementa un *mem chunk* determinato dai segnali di controllo variabili in un range fra 0 e 1, segnali di controllo che a run time e a microfoni chiusi partono da costanti distribuite di 0 e di 1. Per questo motivo è molto importante che il comportamento del *mem chunk* possa tollerare soglie ai due estremi. In partitura altre importanti indicazioni sul campionatore che riguardano lettura e scrittura sono le seguenti:

all sample-read processes should include short fade-in and fade-out ramps when sampling pointers wrap around the buffer, to avoid discontinuities
more discontinuities may occur because the buffer is being written as it is also being read: these can be avoided by various means of one's own design

In Faust non è possibile separare la scrittura e la lettura da un buffer, difatti esistono solo 2 tipi di oggetti per questo scopo: la primitiva *rdtable*, che può essere utilizzata per leggere da una tabella di sola lettura (predefinita prima del tempo di compilazione), e la primitiva *rwtable*, che può essere utilizzata per implementare una tabella di lettura/scrittura. Quest'ultima prende in input un segnale che può essere scritto nella tabella utilizzando un indice di scrittura e letto utilizzando un secondo

indice destinato alla lettura. Per implementare in Faust l'architettura del *sample write* richiesta in partitura, si possono sincronizzare gli indici di scrittura di tutte le tabelle con un unico segnale (rampa) destinato a questo scopo e utilizzato dai *sample read* e dai *granular sampling* impostando una grandezza comune, che risulta essere in questo caso una costante espressa in *var1* in secondi, condivisa da tutti i buffer in questione. In Faust la grandezza della tabella deve essere espressa in campioni. Per ottenere quindi una grandezza espressa in secondi, bisogna dunque moltiplicare il numero dei secondi desiderati per la frequenza di campionamento, definite entrambi come una costante (predefinita prima del tempo di compilazione). Infine per completare le operazioni come richiesto nella partitura, si può sostituire l'oggetto *rwtable* con una mandata (send) del segnale destinato ad entrare in feedback ai *sample read* ed in feedforward al *granular sampling*. A seguito una implementazione del campionatore come richiesto in partitura.

```
// import faust standard library
    import("stdfaust.lib");
    // hard-coded: change this to match your samplerate
    SampleRate = 44100;



    sampler(lengthSec, memChunk, ratio, x) =
    it.frwtable(3, bufferLen, .0, writePtr, x, readPtr) * window
    with {
        memChunkLimited = max(0.100, min(1, memChunk));
        bufferLen = lengthSec * SampleRate;
        writePtr = ba.period(bufferLen);
        grainLen = max(1, ba.if(writePtr > memChunkLimited * bufferLen,
            memChunkLimited * bufferLen, 1));
        readPtr = y
            letrec {
                'y = (ratio + y) % grainLen;
            };
```

```
        window = min(1, abs(((readPtr + grainLen / 2) % grainLen) -
            grainLen / 2) / 200);
    };

    process = sampler(4, hslider("memChunkLimited", 0.100, 0, 1, .001),
        hslider("ratio", 5, .1, 10, .001), os.osc(100)) <: _, _;
```

Listing 5: Algoritmo del *sample write/read* per il feedback study

In questo campionatore, il *ratio* è continuamente variabile, rendendo possibili le modulazioni di frequenza desiderate. Mentre il *mem chunk* ricomincia la sua lettura del buffer dall'inizio ogni volta che la rampa di lettura incontra il valore del modulo determinato dalla sua variabile. Per risolvere le discontinuità come richiesto in partitura, è necessario utilizzare un design personalizzato, e nel design di questo campionatore, per evitare clicks ricorrenti ogni volta che la lettura del campionatore incrocia la sua scrittura, (visto che la lettura del *mem chunk* parte sempre da inizio buffer) la lettura parte solo quando la scrittura è di dimensioni maggiori rispetto al frammento *mem chunk* che dovrebbe essere letto. Infine, per evitare discontinuità ogni volta che la lettura ricomincia a leggere il buffer da 0, è stata utilizzata una finestratura trapezoidale con un fade-in e fade-out per un valore di campioni costante.

Mentre alcuni campionatori fanno parte di un circuito di feedback interno al sistema, sia i granulatori che tutti i campionatori utilizzano lo spazio acustico reale per il feedback, i granulatori in particolare sono l'ultima trasformazione del suono del sistema, prima di andare a due sommatori con il resto delle uscite e all'output. In partitura vengono indicati come a seguito.

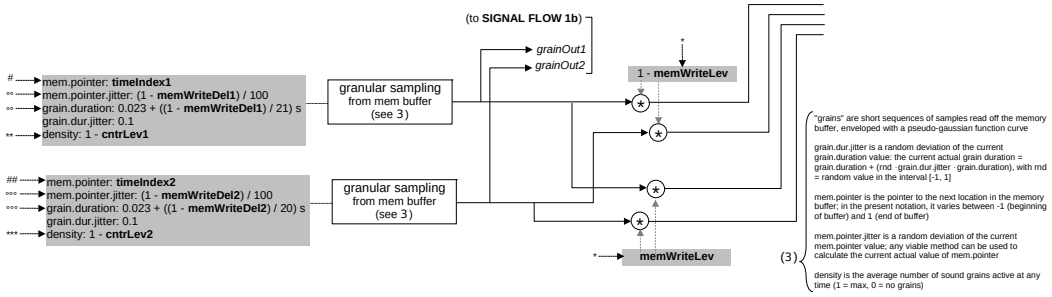

Figure 9: Estratto dalla partitura di *Audible Ecosystemics n.2/ feedback study (2003)* revisione del 2017 - Agostino Di Scipio

Anche in questo caso, oltre alle indicazioni dei segnali di controllo che vengono ricevuti dai due *granular sampling*, in partitura sono presenti delle richiste specifiche riguardo l'implementazione dell'oggetto che ne determinano il risultato dei timbri che produce:

"grains" are short sequences of samples read off the memory buffer, enveloped with a pseudo-gaussian function curve

*grain.dur.jitter is a random deviation of the current grain.duration value: the current actual grain duration is: grain.duration + (rnd * grain.dur.jitter * grain.duration), with rnd is: random value in the interval (- 1, 1)*

mem.pointer is the pointer to the next location in the memory buffer; in the present notation, it varies between - 1 (beginning of buffer) and 1 (end of buffer)

mem.pointer.jitter is a random deviation of the current mem.pointer value; any viable method can be used to calculate the current actual value of mem.pointer

density is the average number of sound grains active at any time (1 is: max, 0 is: no grains)

I due granulatori leggono dunque dalla tabella *sample write* dove vengono scritti al contempo i Larsen diretti e i campionatori in feedback. Questa letura avviene con due puntatori alla memoria *(mem.pointer)* determinati dai segnali di controllo *timeIndex1* e *timeIndex2*, la grandezza dei grani *(grain.duration)* viene invece deter-

minata dai segnali di controllo *memWriteDel1* e *memWriteDel2*, infine il parametro *.jitter* presente sia nei puntatori alla memoria che nelle durate (e calcolato in modo differente per ognuna delle due), ha il compito di decorrelare le fasi dei grani, accedendo alla memoria in punti differenti, e le durate di questi, dando vita in questo modo a delle tessiture timbriche *"clouds"* più interessanti. Anche in questo caso, a seguito di una conversazione con il compositore è emerso che le voci di polifonia (istanze) risultano essere 10 per ogni *granular sampling*. Un problema che sorge nell'implementazione di un granulatore polifonico in un linguaggio di programmazione come Faust (che non contempla segnali di controllo ma lavora in DSP sul singolo campione), è che in necessità di creare delle *"clouds"*, è necessario che i puntatori di lettura siano decorrelati in qualche modo, così da non non avere la stessa fase e sovrapporsi sistematicamente. Nella tradizione della computer music, esistono diverse implementazioni techniche e design di algoritmi di granulazione per risolvere questo tipo di problema: alcune tipologie di granulatori sono dette "asincrone" poiché sono particolari design dove ogni istanza ha un personale *trigger* di partenza che ne avvia la lettura del grano, decorrelato rispetto alle altre istanze, invece e altre tipologie chiamate "quasi sincrone" e "sincrone" si concentrano sul mantenere una coerenza di fase del segnale e una certa continuità. In questo caso per il design di questo oggetto in Faust, si è optato per un tipo di granulazione PSOLA (acronimo di Pitch-Synchronous Overlap and Add, sovrapposizione e aggiunta a toni sincroni), poiché come abbiamo visto precedentemente in partitura, seppur sono presenti funzioni di *jittering* del segnale per ottenere certe tessiture timbriche, viene esplicitamente richiesto che il *timestretching* del suono sia possibile. A seguito una implementazione del granulatore come richiesto in partitura.

```
// import faust standard library
    import("stdfaust.lib");
    // hard-coded: change this to match your samplerate
    SampleRate = 44100;
```

```
//------------------------------------------------ GRANULAR SAMPLING --
grain(L, position, duration, x, trigger) = hann(phase) *
    buffer(readPtr, x)
with {
    maxLength = L * SampleRate;
    length = L * SampleRate;
    hann(ph) = sin(ma.PI * ph) ^ 2.0;
    lineSegment = loop ~ si.bus(2) : _ , ! , _
    with {
        loop(yState, incrementState) = y , increment , ready
        with {
            ready = ((yState == 0.0) | (yState == 1.0)) & trigger;
            y = ba.if(ready, increment, min(1.0, yState + increment));
            increment = ba.if(ready, ma.T / max(ma.T, duration),
                incrementState);
        };
    };
    phase = lineSegment : _ , !;
    unlocking = lineSegment : ! , _;
    lock(param) = ba.sAndH(unlocking, param);
    grainPosition = lock(position);
    grainDuration = lock(duration);
    readPtr = grainPosition * length + phase * grainDuration * ma.SR;
    buffer(readPtr, x) =
        it.frwtable(3, maxLength, .0, writePtr, x, readPtrWrapped)
    with {
        writePtr = ba.period(length);
        readPtrWrapped = ma.modulo(readPtr, length);
    };
};

// works for N >= 2
triggerArray(N, rate) = loop ~ si.bus(3) : (! , ! , _) <:
    par(i, N, == (i)) : par(i, N, \(x).(x > x'))
with {
```

```
    loop(incrState, phState, counterState) = incr , ph , counter
    with {
        init = 1 - 1';
        trigger = (phState < phState') + init;
        incr = ba.if(trigger, rate * ma.T, incrState);
        ph = ma.frac(incr + phState);
        counter = (trigger + counterState) % N;
    };
};

grainN(voices, L, position, rate, duration, x) =
    triggerArray(voices, rate) :
    par(i, voices, grain(L, position, duration, x));

process = os.osc(200) * .5 <: grainN(10, 4,
    hslider("Grain Position", -1, -1, 1, .001),
    hslider("Grain Rate", 1, 1, 100, .001),
    hslider("Grain Duration", 0.100, 0, 1, .001)) :> _;

// in the full system this this is the granular sampling function
granular_sampling(var1, timeIndex, memWriteDel, cntrlLev, divDur, x) =
    grainN(10, var1, position, rate, duration, x) :> _
with {
    rnd = no.noise;
    memPointerJitter = rnd * (1.0 - memWriteDel) * .01;
    position = timeIndex * (1.0 - ((1.0 - memWriteDel) * .01)) +
        memPointerJitter;
    density = 1.0 - cntrlLev;
    rate = 50 ^ (density * 2.0 - 1.0);
    grainDuration = .023 + (1.0 - memWriteDel) / divDur;
    duration = grainDuration + grainDuration * .1 * rnd;
};
```

Listing 6: Algoritmo del *granular sampling* per il feedback study

2.5 Conclusioni

In conclusione, abbiamo avuto modo di approfondire alcuni dei principali meccanismi utilizzati da Agostino Di Scipio per uno dei più importanti lavori del suo ciclo Ecosistemico Udibile, prendendo questo lavoro come caso di studio con il fine di illustrare alcune modalità e relazioni possibili di un sistema con l'ambiente circostante e lo spazio acustico. I meccanismi contenuti all'interno del brano ovviamente non si limitano a questi, e ci sono altre modalità d'interazione con l'ambiente circostante prodotte da altri meccanismi di elaborazione digitale del suono; così come ne esistono delle altre tipologie in altri lavori del ciclo tanto quanto nell'operato di altri compositori, ma si è cercato qui di illustrare alcuni passaggi essenziali così da poter lasciar spazio ad ulteriori approfondimenti dedicati ad altri sistemi di altro tipo, lasciando comunque aperta la possibilità di poter approfondire il brano a partire dai testi citati in bibliografia e dal codice di un porting dell'intero sistema in Faust riportato nell'appendice.

3 Sistemi Autonomi

3.1 Order From Noise (Homage to Heinz Von Foerster)

Nel capitolo precedente, attraverso la composizione e gli studi di Agostino Di Scipio, abbiamo potuto osservare che cosa si intende, e come viene organizzato, un sistema con non linearità provenienti dal mondo fisico; aperto al suo ambiente esterno che diviene in questo caso lo spazio stesso della performance. Sempre in riferimento alla cibernetica di secondo ordine, abbiamo già parlato del fatto che nessun sistema è separabile e isolabile da un suo ambiente circostante, ma abbiamo osservato anche come il concetto di spazio di un sistema sia qualcosa di molto complesso, poiché il concetto stesso di "complessità" sta a significare in un sistema che questo sia composto da una rete d'interazioni dinamiche, ma che il comportamento dell'insieme potrebbe non essere prevedibile in base al comportamento dei diversi componenti tipicamente in relazione fra loro. Mentre, nel caso di Agostino Di Scipio, lo spazio del sistema che ne contiene le sue relazioni è costituito principalmente dal mondo fisico, in questo capitolo osserveremo come, per lo spazio di un sistema, si possa intendere anche uno spazio latente, che traccia le sue relazioni tra le parti in uno spazio virtuale interamente programmato nel software. In questo caso, l'ambiente digitale che racchiude il sistema all'interno e le sue relazioni tra gli agenti è interamente costituito solo dal DSP, inclusi i metodi per la generazione delle relative non linearità. Ho voluto approfondire questo metodo attraverso il lavoro compositivo e gli studi di Dario Sanfilippo, specialista di sistemi di feedback, esecutore e compositore. I suoi lavori riguardano principi di autopoiesi, evolvibilità e costruttivismo radicale nella progettazione di reti di feedback audio complesse.

La modalità con cui andrò a discutere in questo capitolo la composizione di questi sistemi autonomi, appartenenti anche questi alla più vasta categoria dei CASes (Complex Adaptive Systems), è ponendo un focus su un particolare lavoro di Dario Sanfilippo, Order From Noise *(Homage to Heinz Von Foerster)*.

Order From Noise è un progetto che implementa uno dei primi prototipi di Dario

Sanfilippo basati sull'idea dell'*adattattività dinamica*. Questo lavoro del compositore si basa sull'idea che i sistemi adattivi complessi, che sono emergenti per definizione, siano essenziali per raggiungere innovazioni formali, performative e tecniche, nel contesto della performance in live electronics e nella composizione musicale in generale.[27] I CASes si dicono adattivi quando sono composti da una rete dinamica di interazioni dove i singoli agenti sono interconnessi fra loro, interagendo individualmente o collettivamente, possono cambiare il loro stato in risposta a variazioni nell'ambiente o degli altri agenti connessi. Questo tipo di relazioni nel sistema è comune sia nell'opera di Agostino Di Scipio, in cui abbiamo visto precedentemente come lo stato dei singoli agenti nel sistema cambi al variare delle condizioni nell'ambiente circostante portando a cambiamenti di stato globali del sistema, che nel lavoro di Dario Sanfilippo. Altro importante aspetto in comune qui con l'opera di Agostino Di Scipio, è il tipo di rapporto che ha l'uomo con la macchina, che viene respinto nella visione in cui è il primo ad essere totalmente a controllo del secondo, per essere sostituito da un concetto d'interazione Uomo-Macchina-Ambiente, chiamato da Dario Sanfilippo Ibridazione: una condizione in cui umano e macchina cooperano per far emergere la performance. Nel caso delle opere di Dario Sanfilippo, tuttavia, l'estetica dei suoi lavori è interamente subordinata al disegno del sistema stesso, che di volta in volta, emerge da fenomeni musicali che sono il risultato del tipo d'interazione che viene a crearsi fra il performer e il disegno del sistema che si era reso necessario in partenza.

In alcuni casi, questa idea viene portata alle sue estreme conseguenze con opere in cui la macchina è l'unica ente che esegue, come accade per il brano presentato qui.

> *Order from noise (2017) is based on a time-variant feedback delay network containing a set of entangled nonlinear processing algorithms for audio and information signals. The work is an example of autonomous self-performing*

[27] Dario Sanfilippo. "Time-variant infrastructures and dynamical adaptivity for higher degrees of complexity in autonomous music feedback systems: the Order from noise (2017) project". en. In: *Musica/Tecnologia* (Aug. 2018). Artwork Size: 119-129 Pages Publisher: Musica/Tecnologia, 119–129 Pages. DOI: 10.13128/MUSIC_TEC-23804. URL: https://oajournals.fupress.net/index.php/mt/article/view/7482 (visited on 11/22/2022).

system and it is realised by feeding the network with one millisecond of background noise from the performance environment. The initial recirculating noise impulse is what entirely determines the formal evolutions of the system which have substantially different long-term developments for each different noise impulse. An approach to present the work live is that of reinitialising the system a number of times for a period of about three-five minutes to show its sensitivity to initial conditions and the long-term divergence between the formal structures.[28]

Rispetto al sistema discusso nel capitolo precedente, come già accennato c'è una sostanziale differenza nel concetto di spazio o ambiente del sistema. Come appena detto, Order From Noise è basato su una *feedback delay network* tempo-variante, vale a dire che ci ritroviamo di fronte ad un un sistema il cui funzionamento stesso cambia nel tempo, a differenza dei sistemi tempo-invarianti dove seppur l'output del sistema può cambiare nel tempo, il loro funzionamento e l'ambiente che contiene il sistema rimane inalterato. In parole semplici: lo spazio del sistema in Order From Noise può variare nel tempo rendendo imprevedibile il suo funzionamento a partire da una determinata condizione iniziale. Per questo motivo il performer non avrà un grande margine di prevedibilità nel comportamento del sistema a differenza dei sistemi tempo-varianti, ma saprà che per le stesse condizioni iniziali, il sistema qui totalmente deterministico, riporterà sempre le stesse evoluzioni formali. Proprio come nei sistemi caotici emersi dalla teoria del caos introdotti all'inizio di questa tesi, che seppur governati da leggi deterministiche, sono in grado di esibire un'empirica casualità nell'evoluzione delle variabili dinamiche.

3.2 Audio Information Processing

Come già accennato nello scorso capitolo, l'*Audio Information Processing*; ovvero la capacità di un sistema di elaborare le informazioni tremite la *feature extraction*,

[28]Sanfilippo, "Time-variant infrastructures and dynamical adaptivity for higher degrees of complexity in autonomous music feedback systems".

è un dato fondamentale nella creazione di un CAS, che ne determina le sue capacità "congitive" e di auto-osservazione rispetto all'ambiente. L'uso creativo dei CASes in Musica, e la loro stretta connessione con l'informazione, ha favorito lo sviluppo di techniche algoritmiche per l'*Audio Information Processing* e l'analisi di comportamenti, sia ad alto e basso livello.

> *The low-level algorithms provide a continuous measure of the features and can operate with short analysis frames. The high-level algorithms, on the other hand, are original designs informed both perceptually and by complexity theory for the analysis of musically meaningful information, both in short sounds or articulated streams with long-term nontrivial variations.*[29]

Alcuni primi esempi storici, sono ad esempio le performance in live electronics di Gordon Mumma, come: Diastasis, as in Beer (1966), e Hornpipe (1967), dove vengono implementate techniche di amplitude following utilizzate poi per pilotare parametri nelle unità di generazione del suono.[30] Arrivando fino ai sistemi di Agostino Di Scipio, che nello studio del suo feedback study ci ha dato modo di approfondire e scoprire tutti quei meccanismi e tutte quelle particolari funzioni che hanno la responsabilità di generare segnali di controllo nei suoi sistemi.
Dario Sanfilippo ha discusso in più articoli l'utilizzo dell'*Information Processing* nei CASes, sia in senso generale[31], parlando di implementazioni più tipiche come: centroide spettrale, rumorosità, diffusione spettrale a basso livello, eterogeneità e complessità per l'alto livello. Che come in questo caso nell'utilizzo specifico di queste tecniche in un suo sistema:

[29]Dario Sanfilippo. "Time-Domain Adaptive Algorithms for Low- and High-Level Audio Information Processing". en. In: *Computer Music Journal* 45.1 (Mar. 2021), pp. 24–38. ISSN: 0148-9267, 1531-5169. DOI: 10.1162/comj_a_00592. URL: https://direct.mit.edu/comj/article/45/1/24/110389/Time-Domain-Adaptive-Algorithms-for-Low-and-High (visited on 11/22/2022).

[30]Sanfilippo, "Time-Domain Adaptive Algorithms for Low- and High-Level Audio Information Processing".

[31]Sanfilippo, "Time-Domain Adaptive Algorithms for Low- and High-Level Audio Information Processing".

These signals, often based on their perceptual characteristics and their relationship with the domains of the variables in the processing units, are mapped to certain ranges and then used to control the state of the components in a large network. (See Di Scipio (2003) for a detailed discussion on this method). Using infrasonic signals to pilot these variables is highly desirable if not necessary, for high-rate, sudden changes in the DSP parameters would produce an output with a continuously large and homogeneous spectral band, so it would not be possible to perceive the state variations in the long-term. The information and audio processing algorithms implemented, the specific connections between the control signals and the variables, the linear and nonlinear mapping strategies used as well as the network topologies, all these elements determine the infrastructure of a system. In a large network, these elements can already provide a high number of configurations and an even larger number of possible states that a system can reach. That, theoretically, could be considered as something that guarantees a good variety and complexity in the long-term behaviour of a system, albeit the practical case tends to be much different from the ideal scenario.[32]

Nel suo articolo su Order From Noise, Sanfilippo discute tutte le unità utilizzate nel suo sistema e il loro ruolo all'interno della rete. Partendo dalla descrizione di queste nell'articolo, tratterò ora una parte più operativa implementando queste tecniche nel linguaggio di programmazione FAUST (Grame), con un particolare focus su tutte tutte quelle *feature extraction* che nell'implementazione della rete permettono al sistema (nonostante questo sia chiuso rispetto all'ambiente esterno) di mantenersi in uno stato "emergente" manifestando comportamenti nuovi, e di "stabilità" rimanendo sempre in dei range controllati tramite feedback positivi e negativi.

[32]Sanfilippo, "Time-variant infrastructures and dynamical adaptivity for higher degrees of complexity in autonomous music feedback systems".

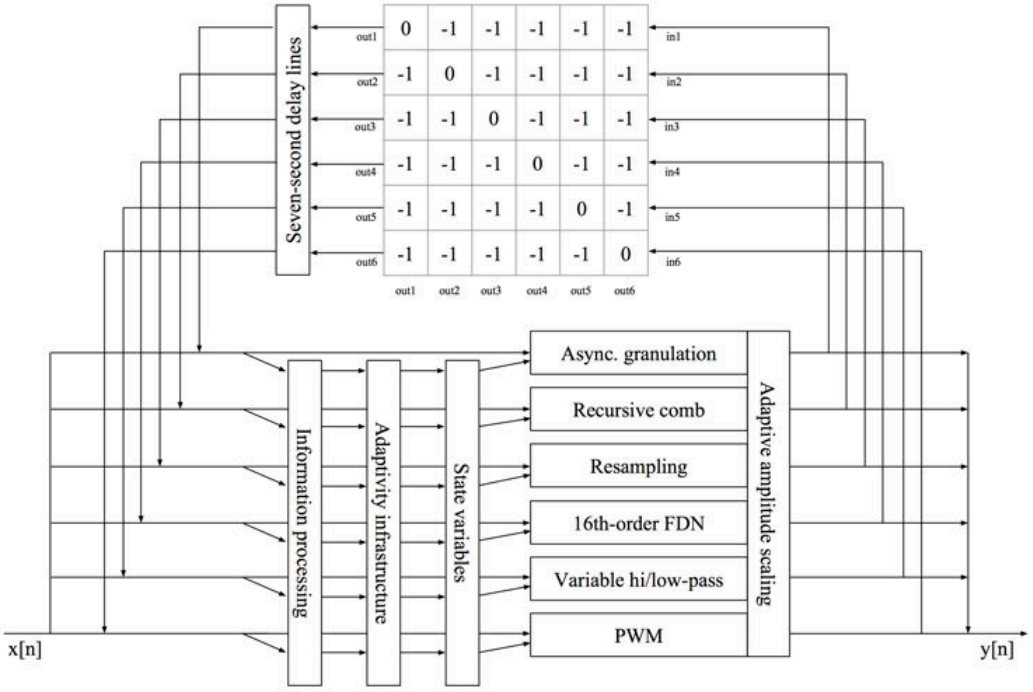

Figure 10: Schema della rete di Order From Noise

La rete di Order From Noise ha sei nodi, ognuno dei quali contiene due unità in cascata, contenenti i seguenti algoritmi di elaborazione audio: *asynchronous granulation, recursive comb filtering, variable high-pass/low-pass filtering, pulse-width modulation (PWM), resampling and 16th-order feedback delay network (FDN) processing.*

A parte le unità di elaborazione del suono, in questo sistema, l'*Information Processing* utilizzato per generare i segnali di controllo è diverso dalle implementazioni tipiche riportate precedentemente nel corso di questa tesi; piuttosto che calcolare *feature extraction* come l'RMS, la diffusione spettale, ecc. Dario Sanfilippo utilizza in questo sistema una sorta di Low Frequency Oscillator non lineare dipendente dal segnale in ingresso. Questo comportamento viene ottenuto utilizzando un filtro Lowpass per far passare solo le componenti energetiche di un segnale sotto la soglia di 1Hz, come ad esempio segnali a 0.01Hz, il segnale risultante viene poi processato tramite normalizzazione dinamica per essere riportato ad una soglia in ampiezza in un range fra [-1; 1], viene elevato a una potenza relativamente grande per forzarlo

intorno a 0, ed infine utilizzato per pilotare la frequenza di un fasore.

```
nonLinearity(exponent, refPeriod, y) = y : lowFreqNoise <:
    (nonlinearsig(exponent, ma.tanh(_ * normRMS)))
with{
    lowFreqNoise(x) = x : seq(i, 4, LPTPT(1.0 / refPeriod));
    noiseRMS(x) = x * x : LPTPT(1.0 / (10.0 * refPeriod)) : sqrt;
    normRMS(x) = 1.0 / max(ma.EPSILON, noiseRMS(x));
    nonlinearsig(exponent, x) = ma.signum(x) * pow(abs(x), exponent);
};

process = no.noise : nonLinearity(1, 1);

// Zavalishin's Onepole TPT Filter
// reference : (by Will Pirkle)
// http://www.willpirkle.com/Downloads/AN-4VirtualAnalogFilters.2.0.pdf
onePoleTPT(cf, x) = loop ~ _ : ! , si.bus(3) // Outs: lp , hp , ap
with {
    g = tan(cf * ma.PI * (1.0/ma.SR));
    G = g / (1.0 + g);
    loop(s) = u , lp , hp , ap
    with {
        v = (x - s) * G; u = v + lp; lp = v + s; hp = x - lp; ap = lp - hp;
    };
};

// Lowpass TPT
LPTPT(cf, x) = onePoleTPT(cf, x) : (_ , ! , !);
```

Listing 7: non linearità in Order From Noise

La funzione `lowFreqNoise(x)` accetta un segnale in ingresso facendolo passare per uno stadio di 4 filtri Lowpass a cascata, dove la stessa frequenza di taglio viene passata esternamente dalla variabile `refPeriod`. Il segnale passa poi per il normal-

izzatore dinamico composto dalle funzioni `noiseRMS(x)` e `normRMS(x)`, questo tipo di normalizzazione dinamica si basa sulla stima RMS e ha due ingressi, uno è il segnale di riferimento, e l'altro è il segnale che deve essere normalizzato sulla base del segnale di riferimento fornito, ovviamente, la finestra RMS deve essere abbastanza grande a seconda della lentezza del segnale che deve essere normalizzato.
Per comprendere meglio i meccanismi alla base della normalizzazione dinamica, andrò a discutere ora il caso più generico di una normalizzazione di picco in DSP.
Partendo dall'algoritmo basilare del *Peakholder* in IIR discusso nel precedente capitolo. L'ampiezza del picco di un segnale può essere normalizzata secondo la seguente espressione

$$y_1(t) = (1/(\max(y_1(t-1), |x(t)|))) * x(t)$$

dove x è il segnale in ingresso, e y il segnale sia in uscita che reiterato nella funzione.
Per prima cosa si calcola con il *Peakholder* il picco massimo del segnale, tracciandone il profilo di ampiezza campione per campione, e a seguito si divide poi il risultato per un fattore per cui si desidera riscalare il segnale, in questo caso una costante 1. Ed infine il profilo di ampiezza risultante da questa operazione verrà moltiplicato per lo stesso segnale in ingresso, incrementando o riducendo l'ampiezza di questo in base al risultato dell'operazione, e ridimensionando grazie a questo processo l'ampiezza di tutti i campioni del segnale, in modo tale che l'ampiezza del picco abbia un valore pari a [1, -1].

```
// import faust standard library
import("stdfaust.lib");

// Peak Max with IIR filter and max comparison
peakmax = loop
with{
```

```
    loop(x) = \(y).((y , abs(x)) : max) ~ _ ;
};

peaknormalization(x) = 1/(peakmax(x)) * x;

process = _ : peaknormalization;
```

Listing 8: Algoritmo di normalizzazione di picco tramite *Peakholder* ad 1 campione di ritardo

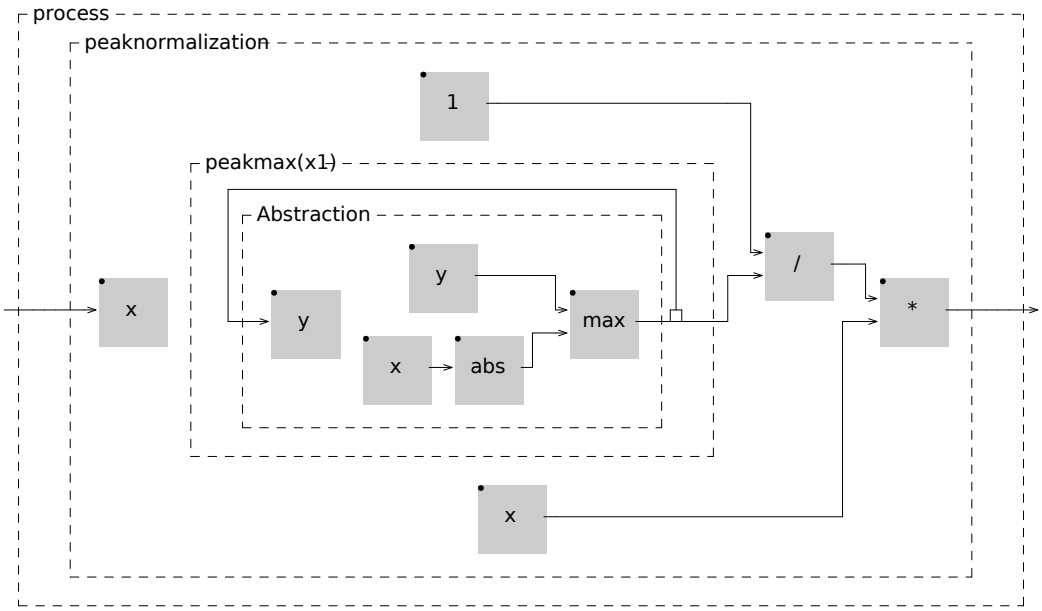

Figure 11: Topologia del normalizzatore di picco tramite *Peakholder* ad 1 campione di ritardo

Proprio come nell'algoritmo del *Peakholder* ad 1 campione di ritardo, il problema di questa applicazione dipende dal fatto che nel segnale complessivo possano permanere contributi spuri che alterino il risultato della normalizzazione, e che si possa generare aliasing a causa di una mancata funzione di smoothing del segnale. Per questi motivi nelle unità della rete di Order From Noise che implementano al proprio interno meccanismi di feedback, e per raggiungere quindi una stabilità globale della rete, è presente un tipo di limiter dinamico chiamato da Dario Sanfilippo col nome di *Lookahead Limiter*, che a differenza dell'algoritmo appena discusso ha

un comportamento adattivo. Questo tipo di Algoritmo viene descritto da Dario Sanfilippo sia nell'articolo di Musica/Tecnologia dell' 11-12 (2017-2018) che parla di Order From Noise citato precedentemente,[33] che nel suo blog, dove ne riporta un implementazione fatta nel linguaggio di programmazione Pure Data.

Recently, I have decided to try the design of a limiter based on a post by IOhannes Zmölnig who, in turn, has based his design on the thesis project by Peter Falkner: Entwicklung eines digitalen Stereo-Limiters mit Hilfe des Signalprozessors DSP56001. This design is based on a peak-hold module with an exponential decay curve... The first step in a lookahead limiter is to delay the input signal so that the attenuating curve can anticipate fast peaks. The amplitude profile of the input signal is obtained by combining a peak-hold module – something that holds a peak for a certain time until a new peak is detected – with a peak envelope one to have a smooth decay when signals transition to lower peaks. Here, after, the peak-hold module, I am also using a one-pole low-pass filter with a period matching that of the input delay so that peaks and input signal are synchronised and the attack is slightly smoothed out... The peak envelope can easily be implemented as a single feedback loop through which the detected peaks recirculate

Lookahead limiting in Pure Data - July 2, 2017[34]

Per implementare il *Lookahead Limiter* sono dunque richieste due unità distinte combinate fra loro in feedforward con un filtro onepole Lowpass fra le due, un *Peakholder* con una finestra che mantiene il picco massimo del segnale e che viene resettata da un timer, e un *Peakenvelope* che ha invece un decadimento basato su *tau* all'interno della retroazione; sono entrambe due unità ottimizzate del nostro algoritmo *Peakholder* ad 1 campione di ritardo, e che combinate fra loro consentono di rilevare sia i transienti di attacco rapidi, che al contempo di avere una lunga fase

[33] Sanfilippo, "Time-variant infrastructures and dynamical adaptivity for higher degrees of complexity in autonomous music feedback systems".

[34] **https://www.dariosanfilippo.com/blog/2017/lookahead-limiting-in-pure-data/**.

di decadimento per i suoni sostenuti e lenti.

Il *Peakenvelope* può essere ottenuto modificando la retroazione dell'algoritmo *Peakholder*, moltiplicandola per un fattore di decadimento come nella seguente formula

$$y_1(t) = \max\left(k_1 * y_1(t-1), |x(t)|\right)$$

dove

$$k_1 = 0.001^{\frac{T}{ST}}$$

è la formula del decadimento RT60, e dove T rappresenta il tempo di decadimento desiderato in secondi, mentre ST rappresenta la durata di un campione in secondi per la frequenza di campionamento corrente. Come nelle altre formule del *Peakholder*, x è il segnale in ingresso, e y il segnale sia in uscita che reiterato nella funzione stessa.

A seguito l'algoritmo in Faust del *Peakenvelope*.

```
// import faust standard library
import("stdfaust.lib");

// Peak Max IIR filter with max comparison and RT60 Decay
peakenvelope(t, x) = abs(x) <: loop ~ _ * rt60(t)
with{
    loop(y, z) = ( (y, z) : max);
    rt60(t) = 0.001^((1/ma.SR) / t);
};

decayFactor = 10;
process = _ : peakenvelope(decayFactor);
```

Listing 9: Algoritmo di *Peakenvelope* con Decay RT60

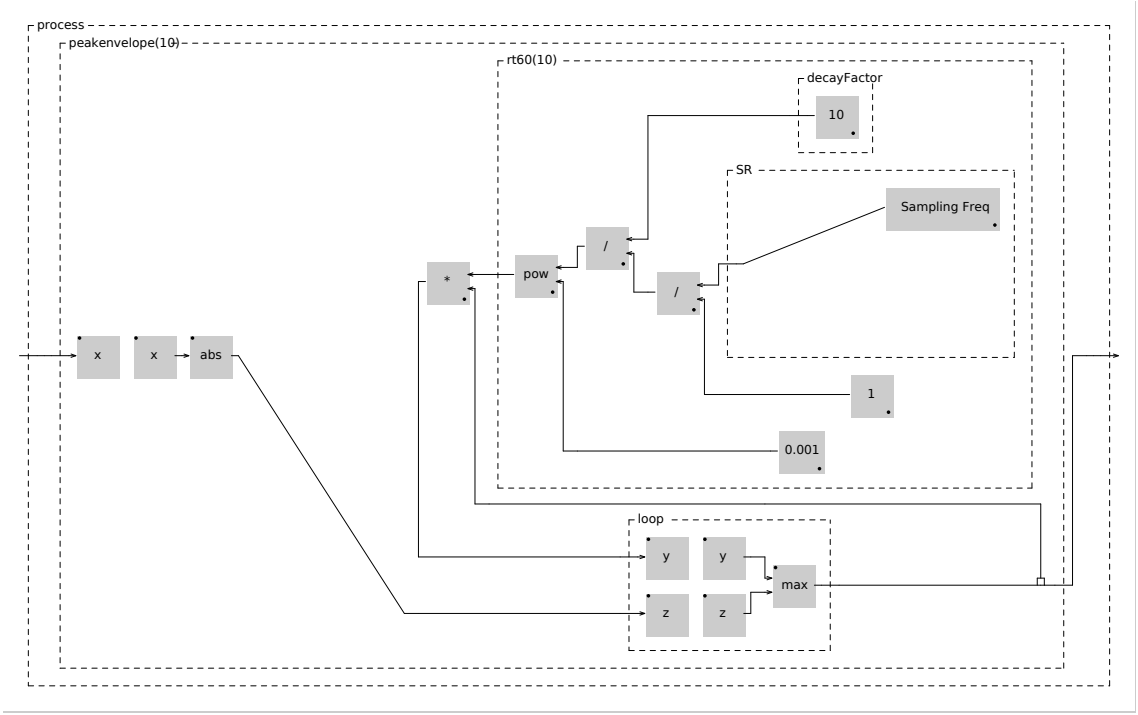

Figure 12: Topologia del *Peakenvelope* con Decay RT60

per quanto riguarda invece l'algoritmo del *Peakholder* adattivo, è un po' meno semplice. Il *Peakholder* verificherà se l'ingresso è maggiore o uguale all'uscita e, se la condizione è vera, aggiornerà il picco e ripristinerà un timer. Quando la condizione invece è falsa, inizierà il suo countdown e, allo scadere del tempo, se non è stato rilevato alcun nuovo picco, qualsiasi valore nell'ingresso corrente verrà impostato come nuovo picco.

```
// import faust standard library
import("stdfaust.lib");

peakHolder(holdTime, x) = loop ~ si.bus(2) : ! , _
with {
    loop(timerState, outState) = timer , output
    with {
        isNewPeak = abs(x) >= outState;
        isTimeOut = timerState >= (holdTime * ma.SR - 1);
        bypass = isNewPeak | isTimeOut;
```

```
        timer = ba.if(bypass, 0, timerState + 1);
        output = ba.if(bypass, abs(x), outState);
    };
};

process = _ : peakHolder(1);
```

Listing 10: Algoritmo del *Peakholder* (adattivo) con Timer

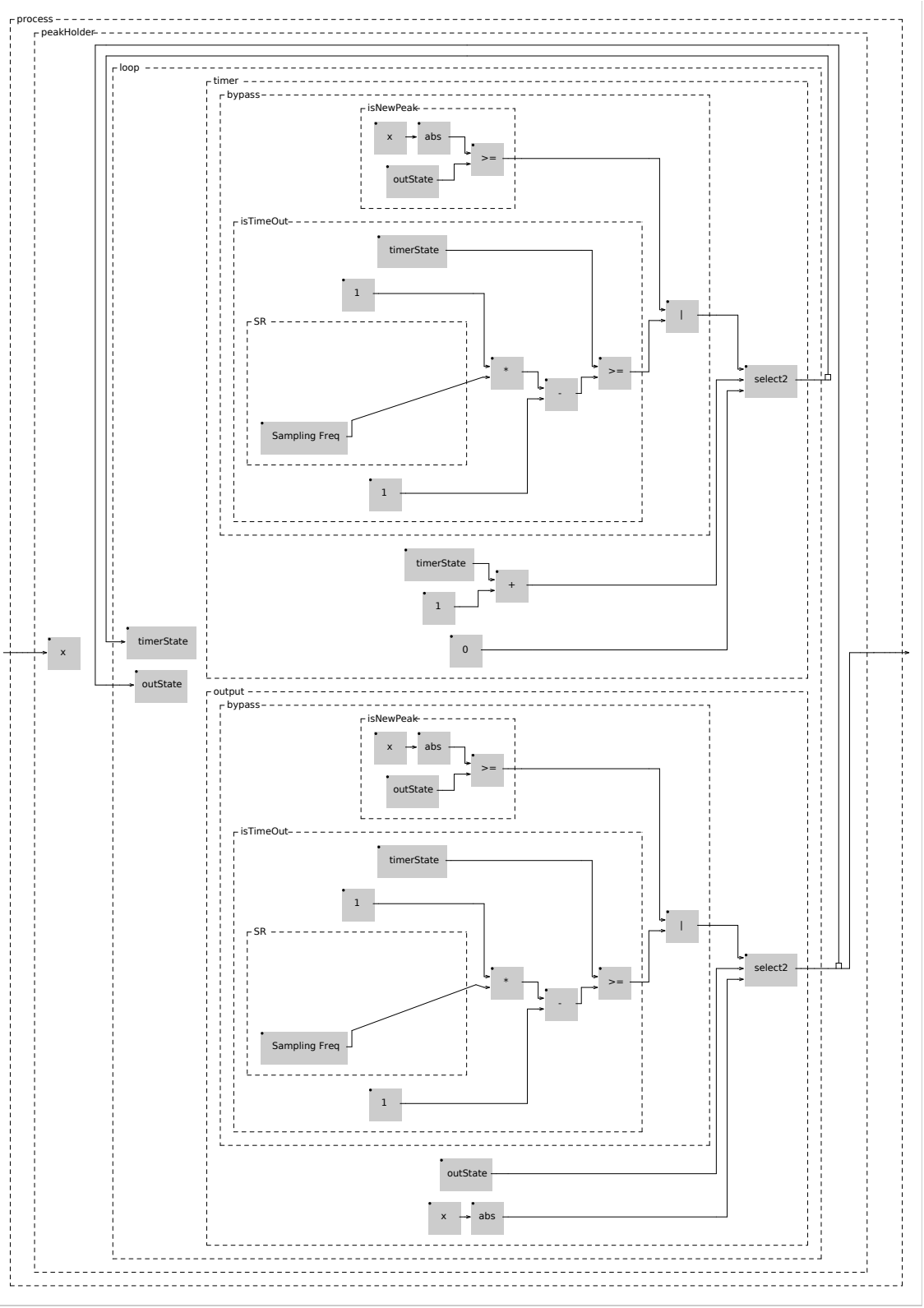

Figure 13: Topologia del *Peakholder* (adattivo) con Timer

Per concludere infine le unità sono combinate fra loro come a seguito, formando nel loro insieme il *Lookahead Limiter*.

```
// import faust standard library
import("stdfaust.lib");

// Peak Max IIR filter with max comparison and RT60 Decay
peakenvelope(t, x) = abs(x) <: loop ~ _ * rt60(t)
with{
    loop(y, z) = ( (y, z) : max);
    rt60(t) = 0.001^((1/ma.SR) / t);
};
// process = _ : peakenvelope(decayFactor);

// PeakHolder with Timer
peakHolder(holdTime, x) = loop ~ si.bus(2) : ! , _
with {
    loop(timerState, outState) = timer , output
    with {
        isNewPeak = abs(x) >= outState;
        isTimeOut = timerState >= (holdTime * ma.SR - 1);
        bypass = isNewPeak | isTimeOut;
        timer = ba.if(bypass, 0, timerState + 1);
        output = ba.if(bypass, abs(x), outState);
    };
};
// process = _ : peakHolder(1);

// PeakHold module with an exponential decay curve
peakHoldwDecay(holdSeconds, frequencyCut, decayT60, x) = x :
    peakHolder(holdSeconds) : LPTPT(frequencyCut) : peakenvelope(decayT60);

// Zavalishin's Onepole TPT Filter
onePoleTPT(cf, x) = loop ~ _ : ! , si.bus(3) // Outs: lp , hp , ap
with {
```

```
    g = tan(cf * ma.PI * (1.0/ma.SR));
    G = g / (1.0 + g);
    loop(s) = u , lp , hp , ap
    with {
        v = (x - s) * G; u = v + lp; lp = v + s; hp = x - lp; ap = lp - hp;
    };
};
// Lowpass TPT
LPTPT(cf, x) = onePoleTPT(cf, x) : (_ , ! , !);
// Highpass TPT
HPTPT(cf, x) = onePoleTPT(cf, x) : (! , _ , !);

// Lookahead Limiter
LookaheadLimiter(threshold, x) = ( x : peakHoldwDecay(.1, 500, 10) ) :
    ( threshold / max(ma.EPSILON, _) : min(1.0) ) *
        ( x @ (ms2samp(1)));

process = _ : LookaheadLimiter(1);
```

Listing 11: Algoritmo del *Lookahead Limiter*

Prima di concludere queste considerazioni a seguito qualche doverosa aggiunta. Anche nell'Ecosistemico Udibile n.2, studio sul feedback, di Agostino Di Scipio è presente una versione ottimizzata dell'algoritmo *Peakholder*, che a differenza del *Peakholder* con il timer discusso precedentemente, viene chiamato come *Local Max*, e riporta l'ampiezza massima solamente al termine del *time frame* che corrisponde al reset del timer, senza necessità della complessa condizione vero/falso discussa precedentemente, ma a discapito di un tempo di ritardo dovuto all'analisi del valore massimo che viene riportato solo al termine del *time frame*. In partitura viene descritto come segue:

local max = returns the maximum signal amplitude (absolute value) in a given time frame; frame duration is dynamically adjusted: the next frame duration is set at the end of the previous frame

A seguito l'implementazione del *Local Max* in Faust.

```
// import faust standard library
import("stdfaust.lib");

localMax(seconds, x) = loop ~ si.bus(4) : _ , ! , ! , !
with {
    loop(yState, timerState, peakState, timeInSamplesState) =
        y , timer , peak , timeInSamples
    with {
        timeInSamples = ba.if(reset + 1 - 1', seconds *
            ma.SR, timeInSamplesState);
        reset = timerState >= (timeInSamplesState - 1);
        timer = ba.if(reset, 1, timerState + 1);
        peak = max(abs(x), peakState * (1.0 - reset));
        y = ba.if(reset, peak', yState);
    };
};

process = os.osc(.1245) : localMax(1);
```

Listing 12: Algoritmo del *Local Max*

3.3 Conclusioni

In conclusione, abbiamo avuto modo di approfondire alcuni dei principali meccanismi utilizzati da Dario Sanfilippo nel suo brano Order From Noise *(Homage to Heinz Von Foerster)*, prendendo questo lavoro come caso di studio, con il fine di illustrare alcune modalità e relazioni possibili per rendere un sistema autonomo e capace di manifestare comportamenti emergenti e complessità anche all'interno di uno spazio deterministico interamente costituito solo dal DSP nel software (che abbiamo chiamato qui col nome di spazio latente); inclusi metodi e strategie per la generazione delle relative non linearità, e meccanismi adattivi per mantenere il sistema in una condizione di stabilità controllata tramite algoritmi di feedback positivo e negativo.

Anche qui i meccanismi contenuti all'interno del brano ovviamente non si limitano solamente a questi, ma si è cercato di approfondire le parti del sistema che riguardano l'*Audio Information Processing*; così da fornire alcuni strumenti che rendano possibile affrontare il problema dell'adattattività e dell'*adattattività dinamica* all'interno dei CASes, lasciando comunque aperta la possibilità di poter approfondire il brano a partire dai testi citati in bibliografia, che riportano anche le formule matematiche e la descrizione per gli altri moduli e le altre unità del brano.

4 Sistemi Caotici

Ciò che accomuna un sistema complesso ad un sistema caotico è la non linearità. In questa visione della complessità, i sistemi caotici sono considerati un sottoinsieme appartenente alla macrocategoria dei sistemi complessi: la complessità si manifesta infatti sulla soglia della caos. Mentre nel mondo fisico la non linearità è insita nella natura delle cose, nel mondo digitale, queste non linearità, come abbiamo visto anche nei capitoli precedenti, devono essere accuratamente programmate ed introdotte all'interno degli algoritmi per raggiungere la complessità.

La teoria del caos postula che esista una classe di fenomeni naturali che possano essere modellati da sistemi deterministici non lineari. Quando si parla di caos ci si riferisce ad un certo tipo di comportamento di un sistema dinamico: determinato dalla sensibilità alle condizioni iniziali, imprevedibilità a lungo termine, e orbite periodiche dense.

In altri termini un sistema caotico amplifica le piccole differenze: porta i fenomeni microscopici a un livello macroscopico. Ed è dunque nell'amplificazione di queste piccole deviazioni delle condizioni iniziali che si annida il caso. Nonostante siano perfettamente individuate le equazioni differenziali che descrivono questi sistemi: al variare delle condizioni iniziali varia il comportamento del sistema stesso negli stadi successivi a quello iniziale. L'ipotesi che i sistemi deterministici possano sviluppare comportamenti impredicibili, come già detto all'inizio di questa tesi fu teorizzata da Henri Poincaré e portata alla fama da Edward Norton Lorenz con il suo articolo.[35]

In questo capitolo della tesi, illustrerò attraverso la composizione di un mio brano, come a partire dalla risoluzione delle equazioni differenziali del sistema di Lorenz, si possa creare un CAS che possa essere utilizzato per la performance musicale in live electronics.

[35] Lorenz, "Deterministic Nonperiodic Flow".

4.1 RITI : Room Is The Instrument

RITI: Room Is The Instrument è un brano che si basa su un Sistema Complesso *site-specific*, in grado di manifestare comportamenti emergenti e caotici, dove la personalità acustica della stanza in cui viene eseguito il brano può essere riflessa in variazioni del comportamento del sistema stesso. L'idea dell'acronimo RITI deriva dal famoso paper di Agostino Di Scipio "Sound is the interface" (SITI), che ha introdotto per la prima volta la prospettiva sistemica sull'auto-organizzazione e sulla capacità di un sistema di auto-osservarsi tramite l'ambiente circostante in musica.[36] Il sistema è costruito utilizzando alla base una soluzione delle equazioni differenziali del sistema di Lorenz come motore di sintesi del suono in DSP; Le esigenze di questo tipo di applicazioni nel mondo della Computer Music sono generalmente motivate dal desiderio di esplorare i comportamenti complessi ed emergenti che questo tipo di sistemi manifesta. Il modello delle Equazioni di Lorenz che ho seguito per questo tipo di sistema, si basa su delle costrizioni matematiche aggiunte alle risoluzioni per esplorare il sistema in regioni instabili, seguendo il metodo descritto nel paper di Dario Sanfilippo "Constrained Differential Equations as Complex Sound Generators".[37] Dove vengono illustrate una serie di risoluzioni per alcune equazioni differenziali caotiche, modificate inserendo opportunamente all'interno di queste un DC Blocker e un Waveshaper (soft clipping) utilizzando la funzione della tangente iperbolica. Oltre alle costrizioni proposte da Dario Sanfilippo, ispirato dall'idea che ha avuto Tom Mudd nel suo paper "Between Chaotic Synthesis and Physical Modelling: Instrumentalising with Gutter Synthesis",[38] di aggiungere alla risoluzione dell'equazione differenziale dell'Oscillatore di Duffing un banco di filtri Bandpass che permettesse al sistema di manifestare risonanze modali che ricordino il compor-

[36] Agostino Di Scipio. "'Sound is the interface': from *interactive* to *ecosystemic* signal processing". en. In: *Organised Sound* 8.3 (Dec. 2003), pp. 269–277. ISSN: 1355-7718, 1469-8153. DOI: 10.1017/S1355771803000244. URL: https://www.cambridge.org/core/product/identifier/S1355771803000244/type/journal_article (visited on 11/22/2022).

[37] Dario Sanfilippo. "CONSTRAINED DIFFERENTIAL EQUATIONS AS COMPLEX SOUND GENERATORS". en. In: (2021), p. 8.

[38] Tom Mudd. "Between chaotic synthesis and physical modelling: Intrumentalizing with Gutter Synthesis". en. In: *Proceedings of the Seventh Conference on Computation, Communication, Aesthetics X* (), p. 12.

tamento di uno strumento musicale. Ho deciso di aggiungere alla risoluzione delle equazioni di Lorenz modificate, un banco parallelo di 32 filtri Bandpass informati con 4 liste ricavate dall'analisi di 4 registrazioni di note diverse di un violoncello, per ottenere una complessità costretta dalle risonanze modali di questi filtri. Per concludere, ho aggiunto infine alle tre equazioni l'ingresso di un segnale esterno, affidato ai microfoni, che permettesse di perturbare il sistema e generare un ulteriore stato di feedback proveniente dall'ambiente della performance, rendendo così il sistema di sintesi sensibile alle condizioni esterne. L'equazione di Lorenz modificata con le costrizioni esposte, costituisce nel suo insieme un singolo agente, a cui ci riferiremo a partire da questo momento con il nome di CSGA *Complex Sound Generator Agent*, e che verrà richiamato a seguito all'interno di una superiore *Feedback Delay Network* (FDN) che ne contiene un numero finito voci determinato a tempo di compilazione. Prima della performance, in fase di compilazione del sistema, è quindi possibile scegliere il numero di CSGA che andranno a costituire la FDN, e di conseguenza il numero di altoparlanti da utilizzare per ascoltare l'output da ogni agente della rete. Tuttavia, si può anche decidere di ascoltare solo un certo numero di output della rete a partire da un numero di altoparlanti ridotti nello spazio della performance, senza rinunciare alla complessità derivata da una compilazione con un numero di agenti che costituisce la rete superiore al numero di ouput che si vogliono ascoltare, ad esempio: 16 CSGA che costituiscono la rete ma ascoltandone solo 4 dagli Altoparlanti. Il feedback globale della rete proveniene da ogni singola voce che va a sommarsi con diversi tempi di ritardo prima della retroiniezione, rientrando poi conseguentemente con opportuni tempi di ritardo nelle altre voci del sistema. Questo processo crea una decorrelazione ed una perturbazione nei singoli agenti all'interno del sistema, ma offre anche la possibilità con opportuni valori di Gain del feedback della FDN, di lasciar ricirolare del materiale nella rete. La performance consiste nell'impostare un insieme iniziale di valori di inizializzazione del sistema, come Dt, Rho, Beta, Sigma, tangente iperbolica, un fattore di Shift in frequenza del banco di filtri, Tempi di ritardo della FDN, ecc. E a partire da questi, l'obiettivo è di

esplorare il sistema finché non si ha l'impressione di aver esaurito tutte le possibilità, con l'obiettivo fondamentale di bilanciare i tre feedback principali: quelli provenienti dalle retroazioni delle tre equazioni differenziali di Lorenz, quello della FDN, e quello proveniente dal contributo dell'ambiente di performance. Nelle sezioni successive della tesi, discuteremo in dettaglio tutti questi aspetti passo per passo come fatto fino ad ora con i lavori precedenti.

4.2 Complex Sound Generator

La nozione di Complex Sound Generator è stata ripresa qui a partire dal paper di Dario Sanfilippo[39], tuttavia nella letteratura della Computer Music, primi esempi di applicazioni di sistemi caotici per la generazione di suoni nella musica risalgono ai primi anni '90 con gli esempi portati da Agostino Di Scipio in suoi articoli come "Composition by exploration of non-linear dynamic systems".[40] O nel corso degli anni '90 con gli studi di Lorenzo Seno al Centro Di Ricerche musicali di Roma sui modelli fisici ed il Caos, con applicazioni nelle composizioni di Michelangelo Lupone. Le radici della sintesi caotica in generale sono multiple e dipendono dalla definizione del campo di applicazione, si potrebbe far risalire a David Tudor con i suoi esperimenti cibernetici sul feedback elettrico del 1960 così come alle implementazioni specifiche esplorate da molti artisti delle equazioni di Rössler nella sintesi video e audio.[41] Il modello delle Equazioni utilizate alla base del CSGA si basa sul modello di Lorenz che fu il primo esempio di un sistema di equazioni differenziali a bassa dimensionalità in grado di generare un comportamento caotico. Venne scoperto da Edward N. Lorenz, del Massachusetts Institute of Technology, nel 1963. Semplificando le equazioni del moto alle derivate parziali che descrivono il movimento termico di convezione di un fluido, Lorenz ottenne un sistema di tre equazioni differenziali del primo ordine. La versione continua dell'equazione differenziale che

[39] Sanfilippo, "CONSTRAINED DIFFERENTIAL EQUATIONS AS COMPLEX SOUND GENERATORS".

[40] Di Scipio, "Composition by exploration of non-linear dynamic systems".

[41] Mudd, "Between chaotic synthesis and physical modelling: Intrumentalizing with Gutter Synthesis".

descrive il modello di Lorenz è descritta come segue:

$$\frac{\partial x}{\partial t} = \sigma(y - x)$$
$$\frac{\partial y}{\partial t} = \rho x - xz - y$$
$$\frac{\partial z}{\partial t} = xy - \beta z$$

Dove x è proporzionale all'ampiezza della circolazione della velocità del fluido nell'anello del fluido, il positivo rappresenta il moto in senso orario ed il negativo il senso antiorario, dove y è la differenza di temperatura tra i fluidi superiori ed inferiori, e dove z è la distorsione dalla linearità del profilo della temperatura verticale. Mentre σ, ρ, β governano la relazione fra le quantità (e sono maggiori di 0), e ∂t rappresenta lo step d'integrazione.
Mentre per una sua rappresentazione in forma di modello discreto, può essere scritto un algoritmo in Faust come segue:

```
// import faust standard library
import("stdfaust.lib");

// Lorenz System
LorenzSystem(x0, y0, z0, dt, beta, rho, sigma) = LorenzSystemEquations ~
    si.bus(3) :
    par(i, 3, _ * 0.002)
    with {
        x_init = x0-x0'; y_init = y0-y0'; z_init = z0-z0';
        LorenzSystemEquations(x, y, z) =
            (x + (sigma * (y - x)) * dt + x_init),
            (y + ((rho * x) - (x * z) - y) * dt + y_init),
            (z + ((x * y) - (beta * z)) * dt + z_init);
    };

// Lorenz System Parameters
```

```
X = 1.2;

Y = 1.3;

Z = 1.6;

DT = .002;

BETA = 8/3;

RHO = 100;

SIGMA = 10;


process = LorenzSystem(X, Y, Z, DT, BETA, RHO, SIGMA);
```

Listing 13: Algoritmo del Sistema di Lorenz Discreto

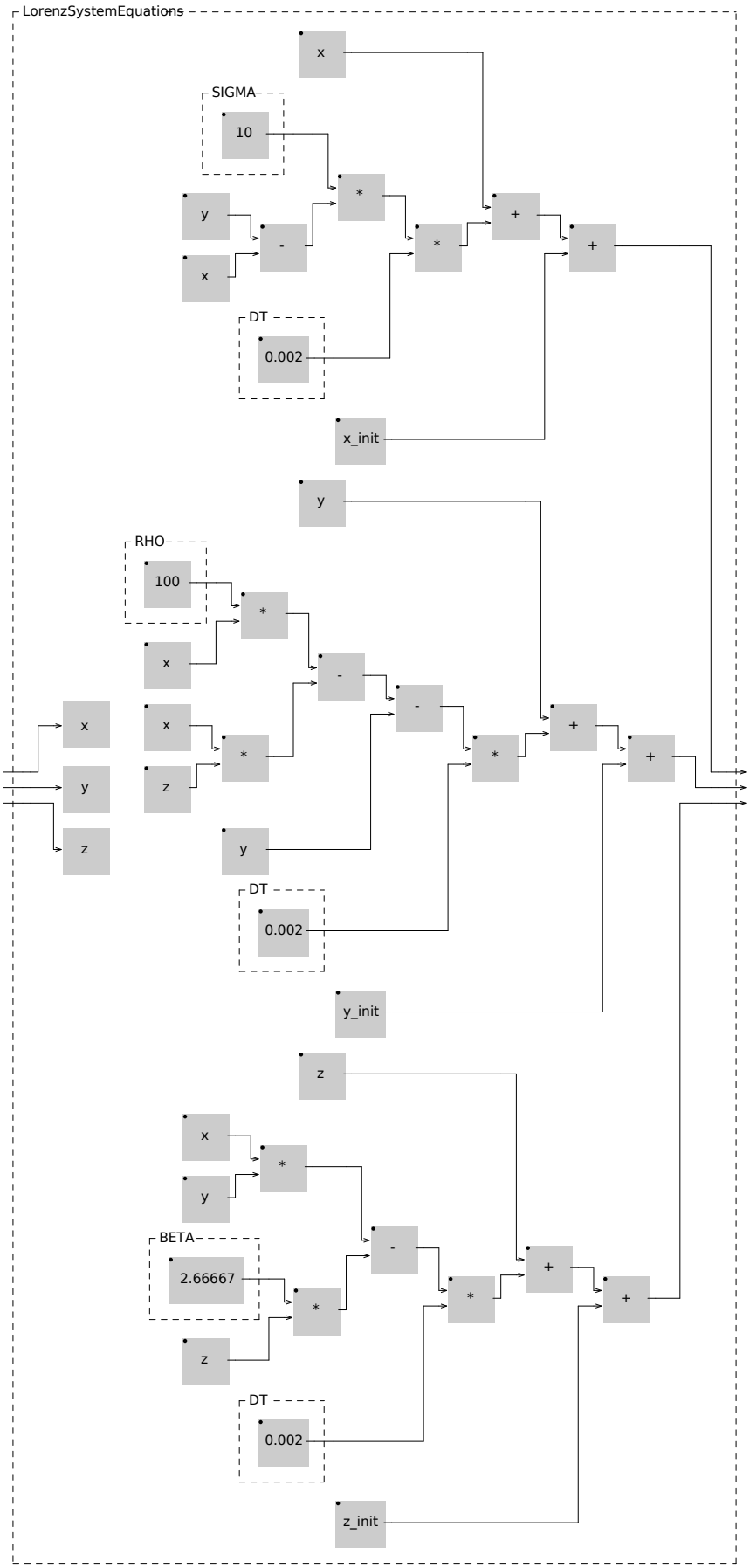

Figure 14: Topologia dell'Algoritmo del Sistema di Lorenz Discreto

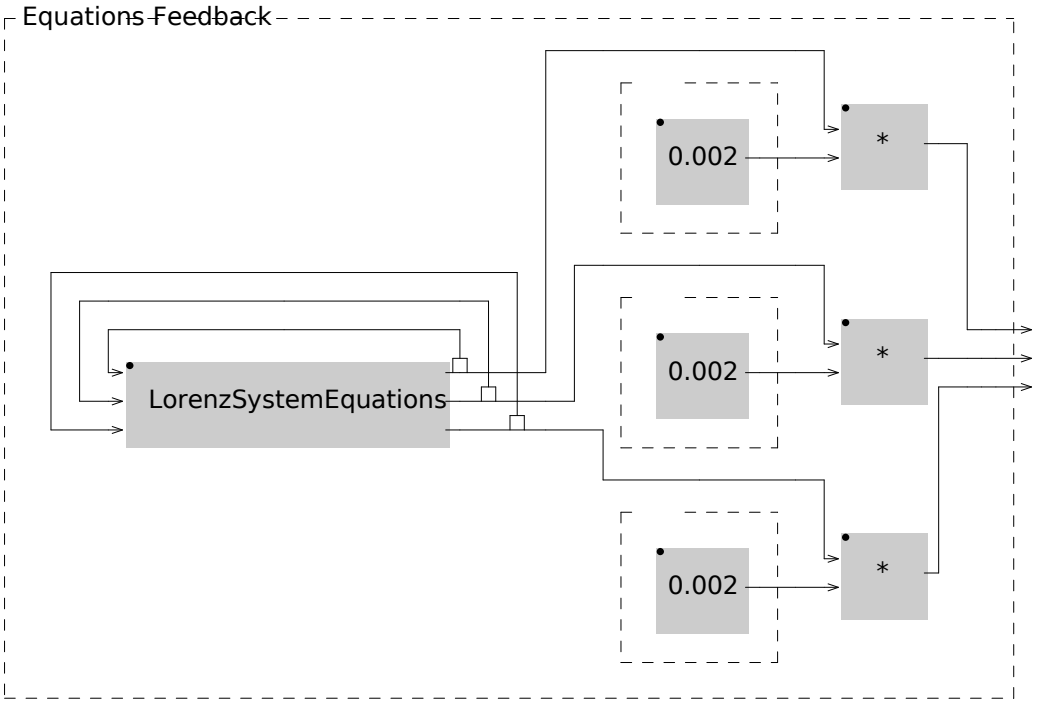

Figure 15: Topologia del Feedback delle Equazioni del Sistema di Lorenz Discreto

Questo algoritmo permette già di produrre sintesi attraverso la risoluzioni delle Equazioni del Sistema di Lorenz, ma per il momento in modo acritico. Come detto in precedenza, esistono ricerche che si pongono il problema di un adattività dei sistemi caotici con il fine di poterli utilizzare per determinati scopi, andrò adesso ad illustrare nel dettaglio i modelli che ho preso a riferimento in questo brano per poter utilizzare i sistemi caotici per fini musicali di sintesi del segnale.

Per poter esplorare un sistema caotico con il fine di creare un segnale udibile in tutte le sue regioni, è necessario dunque porsi dunque un problema di ottimizzazione; è necessario chiedersi quali problemi possano verficarsi e quali strumenti possano contrastarli. I problemi nell'utilizzare Lorenz come segnale di sintesi sono in sostanza Il DC Offset, e i valori più grandi del range [1, -1]. Riguardo le soluzioni sappiamo che il DC blocker, è uno strumento indispensabile nella modellazione della guida d'onda digitale e in altre applicazioni, e che può rimuovere la componente Continua del segnale che circola in una linea di ritardo. Questo consiste sostanzialmente in un filtro FIR highpass, poiché le correnti continue possono essere viste come frequenze

a 0Hz, ed infine un onepole in serie che recupera parte della cancellazione del FIR. Mentre per la funzione della tangente iperbolica, sappiamo che è una funzione non lineare appartenente alle funzioni iperboliche, che costituiscono una famiglia di funzioni elementari dotate di alcune proprietà analoghe e corrispondenti alle proprietà delle ordinarie funzioni trigonometriche. utilizzando la tangente iperbolica come un Waveshaper (soft clipping) possiamo esplorare il sistema caotico in tutte le sue regioni rimanendo sempre in un range [1, -1]. A tal proposito Dario Sanfilippo propone:[42]

Considering first-order differential equations, we can obtain a generalisation of the modified systems discussed here. Let y be a vector of functions, let x be an input vec- tor, let F be a vector of functions of y and x ; let C be a vector of constraining functions. Written in differential form with respect to time, we have that:

$$\frac{\partial y(t)}{\partial t} = C(F(x(t), y(t))) \qquad \text{where} \qquad C(z) = B(l * S(z(l)))$$

with B and S being, respectively, vectors of first-order DC-blockers and saturating nonlinearities with arbitrary saturation threshold piloted by the parameter l. The saturation threshold becomes a key parameter for the interaction with the oscillators, while the overall output can be normalised to unity peak amplitudes for digital audio by merely dividing by l. While several types of bounded saturators are available, here, we will focus on the well known hyperbolic tangent function. Note that the input vector can be used to set the system's initial conditions or as continuous perturbation through signals. In fact, these systems can also be deployed as nonlinear distortion units when operating under non-self-oscillating conditions.

Secondo il metodo appena esposto, le costrizioni possono essere rappresentate

[42] Sanfilippo, "CONSTRAINED DIFFERENTIAL EQUATIONS AS COMPLEX SOUND GENERATORS".

come

$$\text{where B} \qquad y(n) = x(n) - x(n-1) + Ry(n-1)$$

$$\text{where S} \qquad y(t) = tanh(x(t))$$

Mentre per la rappresentazione di Lorenz in forma di modello discreto modificato, può essere scritto un algoritmo in Faust come segue:

```
// import faust standard library
import("stdfaust.lib");

// Hyperbolic Tangent Saturator Parameter
THRESHOLD = 1000;
// Hyperbolic Tangent Saturator Function
saturator(lim, x) = lim * ma.tanh( x / (max(lim, ma.EPSILON)) );

// DC Blocker Parameters
ZERO = 1;
POLE = .995;
// DC Blocker Filter Function
dcblocker(zero, pole, x) = x : _ <: _, mem : _, * (zero) : - : + ~ * (pole);

// Costrained (Modified) Lorenz System
LorenzSystem(x0, y0, z0, dt, beta, rho, sigma, tanHrange) =
    (LorenzSystemEquations : par(i, 3, dcblocker(ZERO, POLE)) : par(i, 3,
        saturator(tanHrange))) ~
        si.bus(3) : par(i, 3, _ / (tanHrange)) :> (_ / 3)
    with {
        x_init = x0-x0'; y_init = y0-y0'; z_init = z0-z0';

        LorenzSystemEquations(x, y, z) =
            (x + (sigma * (y - x)) * dt + x_init),
            (y + ((rho * x) - (x * z) - y) * dt + y_init),
```

```
            (z + ((x * y) - (beta * z)) * dt + z_init);
    };


process = LorenzSystem(1.2, 1.3, 1.6, .150, 2, 3.4, 1.9, THRESHOLD);
```

Listing 14: Algoritmo del Sistema di Lorenz Modificato con DC-blocker e TanH

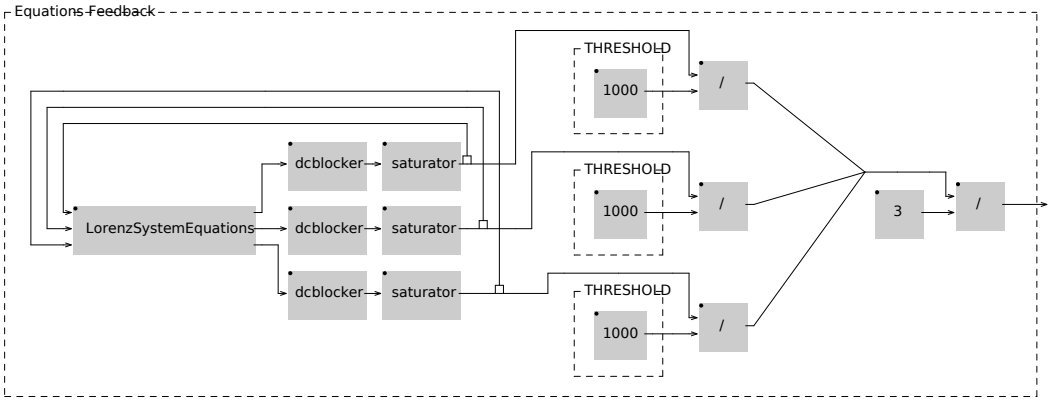

Figure 16: Topologia del feedback delle Equazioni del Sistema di Lorenz Modificato con DC-blocker e TanH

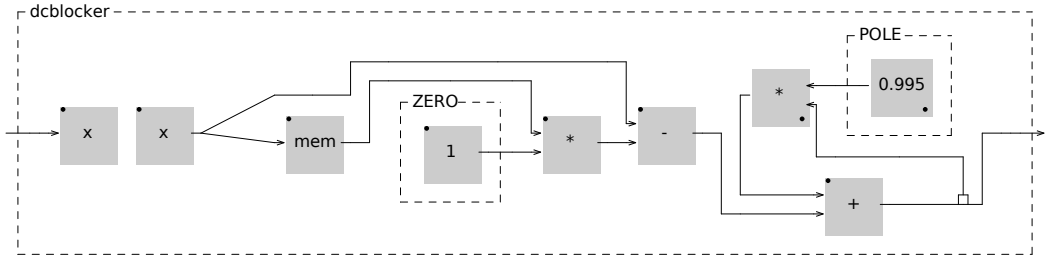

Figure 17: Topologia dell'Algoritmo DC-blocker

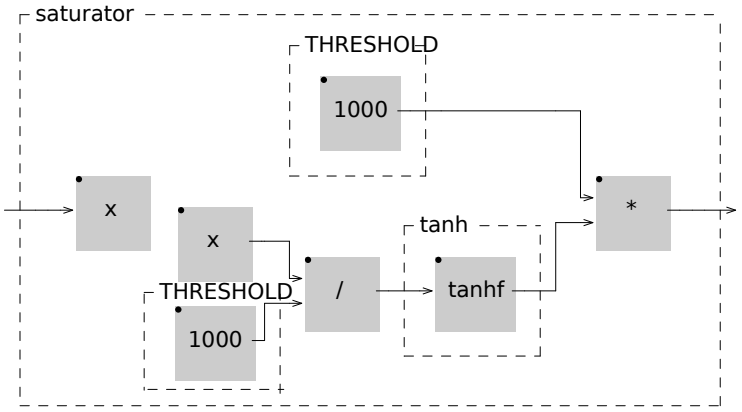

Figure 18: Topologia dell'Algoritmo TanH

Ora il sistema di Lorenz è invece ottimizzato per la sintesi sonora. Per concludere, il sistema di Lorenz Modificato passa infine per la sua terza costrizione, che consiste in un Banco di Filtri Bandpass dove per un solo termine in ingresso il segnale viene diviso per il numero di banchi di filtri e restituito in uscita.

Il banco di filtri Bandpass è costituito da 32 voci parallele informate da 4 liste di ricavate da analisi di 4 file audio di note del violocello, esplorate durante il corso della performance. L'algoritmo dei filtri Bandpass utilizzato come costrizione per il sistema di Lorenz Modificato, consiste in un implementazione *Virtual Analog* di un filtro del 2° ordine basato sulla struttura SVF *State Variable Filter*. La particolarità di questo tipo d'implementazione di SVF che ho utilizzato, è che preserva matematicamente la topologia del filtro analogico, includendo nella maggiorparte dei casi *zero-delay loops*. Questi filtri sono chiamati *Topology Preserving Transform filters* o filtri TPT, e provengono dal libro "The Art of VA Filter Design" di Vadim Zavalishin,[43] pubblicato e reperibile gratuitamente in rete.

A seguito riporto l'implementazione matematica di questo filtro SVF TPT di Vadim Zavalishin che ha esposto Will Pirkle in un suo articolo chiamato "Virtual Analog (VA) Filter Implementation and Comparisons v2.0"[44] del 2013 e reperibile dal suo

[43] Vadim Zavalishin. *THE ART OF VA FILTER DESIGN*. en. rev. 2.1.0 (October 28, 2018), p. 520.

[44] Will Pirkle. "Virtual Analog (VA) Filter Implementation and Comparisons v2.0". en. In: *www.willpirkle.com* (), p. 24.

sito.

$$y_{hp}(n) = \frac{x(n) - 2Rs_1(n) - gs_1(n) - s_2(n)}{1 + 2Rg + g^2}$$

in questa formula viene prima calcolata l'equazione differenziale per l'uscita yHP (Highpass), e dopo, applicata all'ingresso del primo integratore, utilizzando il metodo il *read-before-write* (ovvero s_1 e s_2 vengono letti per primi e aggiornati per ultimi). Vengono ricavate da qui le equazioni alle differenze per tutti gli altri filtri:

$$\begin{aligned}
y_{bp}(n) &= gy_{hp}(n) + s_1(n) \\
y_{lp}(n) &= gy_{bp}(n) + s_2(n) \\
y_{ubp}(n) &= 2Ry_{bp}(n) \\
y_{bshelf}(n) &= x(n) + 2KRy_{bp}(n) \\
y_{notch}(n) &= x(n) - 2Ry_{bp}(n) \\
y_{apf}(n) &= x(n) - 4Ry_{bp}n) \\
y_{peak}(n) &= y_{lp}(n) - y_{hp}(n)
\end{aligned}$$

prendendo quindi la funzione y_{bp}, e assumendo ancora una volta C come vettore delle costrizioni per le tre equazioni differenziali del sistema di Lorenz, e aggiungendo ora Q come la funzione che rappresenta il banco di filtri, avremo infine che:

$$\begin{aligned}
\text{for} \qquad & \frac{\partial y(t)}{\partial t} = C(F(x(t), y(t))) \\
\text{where} \qquad & C(z) = Q(B(l * S(z(l)))) \\
\text{Q is} \qquad & y(n) = x_{1bp}(n) + x_{2bp}(n) + x_{3bp}(n) + ...x_{32bp}(n)
\end{aligned}$$

il CSGA in forma di modello discreto modificato, può infine essere scritto in un algoritmo in Faust come segue:

```
// import faust standard library
import("stdfaust.lib");

// FFT analysis - List
Cello1_D2_frequencies = (
368.0, 221.0, 147.0, 515.0, 736.0, 74.0, 589.0, 810.0, 883.0, 295.0, 442.0,
    662.0, 1693.0, 956.0, 1030.0, 2282.0, 1914.0, 1546.0, 1104.0, 1472.0, 2356.0,
    2503.0, 2135.0, 1251.0, 1398.0, 1325.0, 1178.0, 2061.0, 2429.0, 2209.0,
    1619.0, 2572.0, 1840.0, 2655.0, 1766.0, 2865.0) ;
Cello1_D2_amplitudes = (
1.0, 0.883508750371294, 0.6105272358722508, 0.25009984168236576,
    0.2004373971446242, 0.19782531572463796, 0.15986608084324763,
    0.15408867955329245, 0.15388399774271314, 0.12206611629626225,
    0.07568694217870818, 0.0749419685950924, 0.04993743413946004,
    0.04296305701601583, 0.03695324484801036, 0.035190168499954565,
    0.034299901561064564, 0.027814420294695608, 0.025519908386620865,
    0.025451275008886335, 0.024970356245165314, 0.02376619002590769,
    0.02359690477324046, 0.02108580284017779, 0.020955020563014883,
    0.02080994218773682, 0.020502590513884294, 0.019359564369447416,
    0.01934635943322834, 0.018745547950376563, 0.014962909727368317,
    0.011249397635411025, 0.00849049090994134, 0.0070851384197025405,
    0.006322712098150642, 0.0062269201300692465) ;
Cello1_D2_bandwidths = (
1.0
) ;
Cello2_D2_frequencies = (
129.0, 782.0, 260.0, 521.0, 391.0, 1042.0, 1949.0, 912.0, 1558.0, 2079.0, 651.0,
    2351.0, 1699.0, 2221.0, 1433.0, 1819.0, 442.0, 1308.0, 1178.0, 569.0, 2481.0,
    2275.0, 2861.0, 2532.0, 2403.0, 2991.0, 2600.0, 2142.0, 3116.0, 1753.0,
    2667.0, 3377.0, 3637.0, 3446.0, 3523.0, 3268.0) ;
Cello2_D2_amplitudes = (
1.0, 0.5539841329657997, 0.5527755241243919, 0.3798690322097816,
```

```
    0.24385623120426272, 0.1858573818584144, 0.1718924484719696,
    0.13549203046918346, 0.09550335625230665, 0.06326345700196957,
    0.05721228382207657, 0.046121609704541516, 0.04311865463771355,
    0.032964828166694375, 0.02880281110676074, 0.02747889262701017,
    0.02513218851591026, 0.02498359541143432, 0.024351519031547093,
    0.022320475851666407, 0.020022022408488896, 0.01947464397996532,
    0.018274828098453825, 0.017151821615801758, 0.016455897816434754,
    0.014466403589882717, 0.011927332037780307, 0.010669371414647293,
    0.008807995446126208, 0.007513211416567189, 0.007349012756623662,
    0.0044685740115062425, 0.004297015920414301, 0.004118966421804356,
    0.003062546865966815, 0.002902563274463849) ;
Cello2_D2_bandwidths = (
1.0
) ;
Cello3_D2_frequencies = (
98.0, 783.0, 294.0, 587.0, 685.0, 196.0, 489.0, 1566.0, 2055.0, 391.0, 1957.0,
    1468.0, 881.0, 1076.0, 2153.0, 1664.0, 2544.0, 2349.0, 2251.0, 1370.0,
    2446.0, 1272.0, 2642.0, 978.0, 2936.0, 1761.0, 1174.0, 3033.0, 2740.0,
    3229.0, 832.0, 2838.0, 3327.0, 3621.0, 3717.0, 3816.0, 3425.0, 3523.0) ;
Cello3_D2_amplitudes = (
1.0, 0.5246811846335828, 0.4012585768972376, 0.22942034106512663,
    0.20079641370293627, 0.18876983281899867, 0.12502443350296757,
    0.0933584257472202, 0.07841217109023603, 0.07786536078021235,
    0.06732374792084482, 0.06398941035139119, 0.06048111263906231,
    0.05915731723672648, 0.058731470170536024, 0.05118398937740636,
    0.046391926665833076, 0.040066228067825026, 0.03988990722982784,
    0.03345736827139034, 0.032867690289697674, 0.026159698256227056,
    0.02450082955507741, 0.022720013057591623, 0.01673071239995854,
    0.01386228359792361, 0.011125330538207747, 0.01045539895797359,
    0.008912949330652495, 0.008614981571685254, 0.007903782014397535,
    0.006904461116857643, 0.005737821129793645, 0.004468696578275683,
    0.004458611785686046, 0.004255083332897215, 0.004082351498449864,
    0.003974921717750587) ;
Cello3_D2_bandwidths = (
1.0
```

```
) ;
Cello4_D2_frequencies = (
165.0, 247.0, 82.0, 412.0, 329.0, 576.0, 824.0, 906.0, 741.0, 494.0, 659.0,
    1560.0, 1725.0, 1071.0, 1658.0, 2152.0, 1313.0, 994.0, 1488.0, 1807.0,
    2564.0, 1905.0, 1230.0, 2296.0, 1972.0, 2214.0, 2070.0, 2399.0, 1395.0,
    1153.0, 2950.0, 2873.0, 2708.0) ;
Cello4_D2_amplitudes = (
1.0, 0.40463907988091696, 0.24305538012575184, 0.09564215197843938,
    0.07882101859578412, 0.07272871966988338, 0.06616610763986651,
    0.05372820415795432, 0.03670485927598048, 0.027388497371127554,
    0.021635361017390872, 0.010607924019265583, 0.009233410589260653,
    0.008987446999479275, 0.008735131399976701, 0.007861982586356843,
    0.007105490361846338, 0.006690209866887384, 0.005720588228210853,
    0.005655438992297758, 0.00564938401160071, 0.0054292290841770575,
    0.00528525120649931, 0.0051118414293123745, 0.004984368543658675,
    0.004299506779823275, 0.004149143995148571, 0.003924111275742912,
    0.003601811817177141, 0.002426805651034355, 0.001983449226846196,
    0.0017614827121850407, 0.0015088640939487819) ;
Cello4_D2_bandwidths = (
1.0
) ;

// Take FFT Lists
FrequenciesListCH1(index) = ba.take(index, Cello1_D2_frequencies) ;
AmplitudesListCH1(index) = ba.take(index, Cello1_D2_amplitudes) ;
BandwidthsListCH1(index) = ba.take(1, Cello1_D2_bandwidths) ;
FrequenciesListCH2(index) = ba.take(index, Cello2_D2_frequencies) ;
AmplitudesListCH2(index) = ba.take(index, Cello2_D2_amplitudes) ;
BandwidthsListCH2(index) = ba.take(1, Cello2_D2_bandwidths) ;
FrequenciesListCH3(index) = ba.take(index, Cello3_D2_frequencies) ;
AmplitudesListCH3(index) = ba.take(index, Cello3_D2_amplitudes) ;
BandwidthsListCH3(index) = ba.take(1, Cello3_D2_bandwidths) ;
FrequenciesListCH4(index) = ba.take(index, Cello4_D2_frequencies) ;
AmplitudesListCH4(index) = ba.take(index, Cello4_D2_amplitudes) ;
BandwidthsListCH4(index) = ba.take(1, Cello4_D2_bandwidths) ;
```

```
// linear interpolation
linInterpolate(x0, x1, delta) = x0 + delta * (x1-x0);
siglinInterpol(order, x) = x : seq(r, order, interpolate)
with{
    interpolate(y) = y + .5 * (y' - y);
};
// bilinear interpolation
bilinInterpolate(x0, x1, x0b, x1b, dt1, dt2) =
    linInterpolate(
        linInterpolate(x0, x1, dt1),
        linInterpolate(x0b, x1b, dt1),
        dt2)
    with{
        linInterpolate(x0, x1, delta) = x0 + delta * (x1-x0);
    };
// lists interpolations
FrequenciesListinterpolate(index, dt1, dt2) =
    bilinInterpolate(FrequenciesListCH1(index), FrequenciesListCH2(index),
    FrequenciesListCH3(index), FrequenciesListCH4(index), dt1, dt2);
AmplitudesListinterpolate(index, dt1, dt2) =
    bilinInterpolate(AmplitudesListCH1(index), AmplitudesListCH2(index),
    AmplitudesListCH3(index), AmplitudesListCH4(index), dt1, dt2);
BandwidthsListinterpolate(index, dt1, dt2) =
    bilinInterpolate(BandwidthsListCH1(index), BandwidthsListCH2(index),
    BandwidthsListCH3(index), BandwidthsListCH4(index), dt1, dt2);

// optimized BP from the TPT version of the SVF Filter by Vadim Zavalishin
BPSVF(glin, bw, cf, x) = loop ~ si.bus(2) : (! , ! , _)
    with {
        g = tan(cf * ma.PI * (1.0/ma.SR));
        Q = cf / max(ma.EPSILON, bw);
        R = 1.0 / (Q + Q);
        G = 1.0 / (1.0 + 2.0 * R * g + g * g);
        loop(s1, s2) = u1 , u2 , bp * glin
```

```
            with {
                bp = (g * (x - s2) + s1) * G;
                bp2 = bp + bp;
                v2 = bp2 * g;
                u1 = bp2 - s1;
                u2 = v2 + s2;
            };
    };

// Spectre BP Filter Banks
BandpassFiltersBank(bypassFilter, filterPartials, filterOrder, globalFreq,
    globalAmps, globalBW, interpolation1, interpolation2, x) = x <:
    par(i, filterPartials,
        seq(r, filterOrder,
            BPSVF(
                AmplitudesListinterpolate( (i + 1), interpolation1, interpolation2)
                    * globalAmps,
                BandwidthsListinterpolate( (i + 1), interpolation1, interpolation2)
                    * globalBW,
                FrequenciesListinterpolate( (i + 1), interpolation1,
                    interpolation2) * globalFreq
            )
        )
    ):> (+ / filterPartials) * (1 - bypassFilter) + x * bypassFilter;

// Hyperbolic Tangent Saturator Parameter
THRESHOLD = 1000;
// Hyperbolic Tangent Saturator Function
saturator(lim, x) = lim * ma.tanh( x / (max(lim, ma.EPSILON)) );

// DC Blocker Parameters
ZERO = 1;
POLE = .995;
// DC Blocker Filter Function
dcblocker(zero, pole, x) = x : _ <: _, mem : _, * (zero) : - : + ~ * (pole);
```

```
// Costrained (Modified) Lorenz System
LorenzSystem(x0, y0, z0, dt, beta, rho, sigma, tanHrange) =
    ( LorenzSystemEquations : par(i, 3, dcblocker(1, .995)) :
        par(i, 3, saturator(tanHrange)) :
            par(i, 3, _ : BandpassFiltersBank(0, 32, 1, 1, 1, 2, 1, 1))
    ) ~ si.bus(3) :
        par(i, 3, _ / (tanHrange)) :> (_ / (3 * 2))
    with {
        x_init = x0-x0'; y_init = y0-y0'; z_init = z0-z0';

        LorenzSystemEquations(x, y, z) =
            (x + (sigma * (y - x)) * dt + x_init),
            (y + ((rho * x) - (x * z) - y) * dt + y_init),
            (z + ((x * y) - (beta * z)) * dt + z_init);
    };

process = LorenzSystem(1.2, 1.3, 1.6, .150, 2, 3.4, 1.9, THRESHOLD) <: _, _;
```

Listing 15: Algoritmo del Sistema di Lorenz Modificato con Filterbak

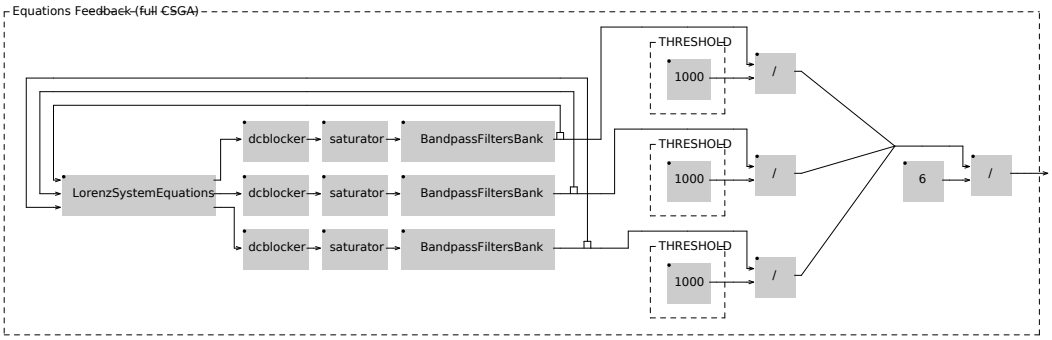

Figure 19: Topologia del feedback delle Equazioni del Sistema di Lorenz Modificato con Filterbank

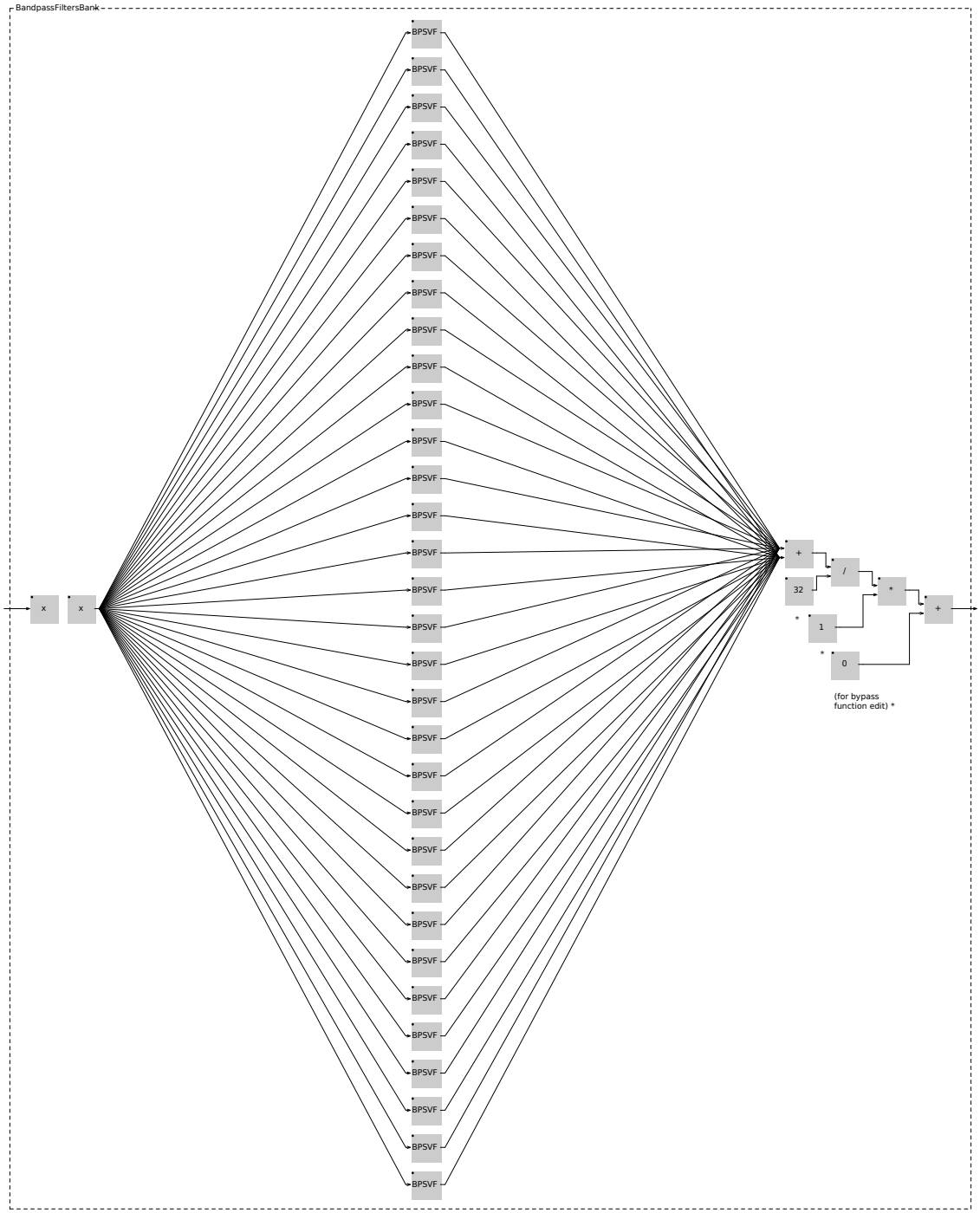

Figure 20: Topologia del Filterbank con i filtri Bandpass SVF TPT

Questo andrà a rappresentare un singolo Agente richiamato all'interno della FDN. Prima di passare alla descrizione globale della rete, voglio ringraziare qui Edoardo Staffa per avermi assistito nel realizzare un semplice metodo di analisi DFT per i file audio .wav tramite alcune librerie nel Linguaggio di programmazione

Python3. In un secondo momento ho poi modificato il codice per ottenere le liste con il metodo che mi occorreva per informare i filtri di Faust. L'algoritmo in Python è il seguente:

```
# import libraries
import matplotlib.pyplot as plt
from scipy.io import wavfile as wav
from scipy.fftpack import fft
import numpy as np
import argparse
import os



# Print the sample rate and length of the audio file in seconds
print('''
This code take as arguments the:
$audiofile.wav $Minimum frequency in the range and $Maximum frequency in the
    range $Number of Bins,
then do an FFT in the entire spectrum multiplying the desidered number of Bins in
    the section
that you want to analyze with this bins for the entire spectrum.
Then after the analysis plot in the result only the elements corresponding to the
    desired frequency range
with a boolean mask to select only the elements desidered (analyzed with the
    desidered number of Bins)
''')

# Parse the command line arguments
parser = argparse.ArgumentParser()
parser.add_argument("audiofilein", help="Name of the audio file with the
    extension")
parser.add_argument("min_frequency", type=int, help="Minimum frequency in the
    range")
parser.add_argument("max_frequency", type=int, help="Maximum frequency in the
    range")
```

```
parser.add_argument("num_bins", type=int, help="Number of Bins")
args = parser.parse_args()

# Read the audio file
rate, data = wav.read(args.audiofilein)
print(f"Sample Rate of the audio file: {rate}")

# calculate the number of bin for your Hz range
resolutionbin = ((20000 / (args.max_frequency - args.min_frequency)) *
    args.num_bins)
print(f"Number of Bins necessary in the full spectrum for your bin N in your
    range: {resolutionbin}")

# number of bins used for the fft
numberofbins = int( (resolutionbin / 20000) * rate)

# Perform the FFT with numberofbins bins
fft = np.fft.fft(data, n=(numberofbins))

# Get the frequencies and amplitudes
frequencies = np.abs(fft)

# Get the frequencies corresponding to each element in the FFT result
frequencies_axis = np.fft.fftfreq(len(frequencies), d=1/rate)

# Calculate the bin bandwidth
bin_bandwidth = rate / (numberofbins)

# Select only the elements corresponding to the desired frequency range
selected_indices = (frequencies_axis >= args.min_frequency) & (frequencies_axis
    <= args.max_frequency)

# set the tolerance value
tolerance = 1000
```

```
# create empty lists for the filtered frequencies and amplitudes
filtered_frequencies = []
filtered_amplitudes = []

# loop through the selected frequencies and amplitudes
for i, freq in enumerate(selected_frequencies_axis):
    # initialize a flag to indicate whether the frequency should be included
    include_frequency = True
    # loop through the filtered frequencies
    for f in filtered_frequencies:
        # check if the difference between the current frequency and any other
            frequency in the list is within the tolerance range
        if abs(freq - f) < tolerance:
            include_frequency = False # set the flag to False
            break # exit the inner loop
    # check if the flag is still True
    if include_frequency:
        # if the flag is still True, append the frequency and the corresponding
            amplitude to the filtered lists
        filtered_frequencies.append(freq)
        filtered_amplitudes.append(scaled_amplitudes[i])

# update the selected_frequencies_axis and scaled_amplitudes lists with the
    filtered values
selected_frequencies_axis = filtered_frequencies
scaled_amplitudes = filtered_amplitudes

'''
# Normalize the selected amplitudes
normalized_amplitudes = selected_amplitudes_axis /
    np.max(selected_amplitudes_axis)

# Scale the normalized amplitudes to the desired range (0 to 1 in this example)
scaled_amplitudes = normalized_amplitudes * 1
'''
```

```
# Plot the scaled amplitudes
plt.plot(selected_frequencies_axis, scaled_amplitudes)
plt.show()

# Use os.path.splitext to split the extension from the root
newfilename = (args.audiofilein)
root, ext = os.path.splitext(newfilename)

# Save the selected frequencies
with open((root)+".lib", "w") as f:
    f.write((root)+"_frequencies = (\n")
    for i in range(len(selected_frequencies_axis)):
        if i == len(selected_frequencies_axis) - 1:
            f.write(str(selected_frequencies_axis[i]) + ") ; \n \n")
        else:
            f.write(str(selected_frequencies_axis[i]) + ", ")

# Save the scaled amplitudes
    f.write((root)+"_amplitudes = (\n")
    for i in range(len(scaled_amplitudes)):
        if i == len(scaled_amplitudes) - 1:
            f.write(str(scaled_amplitudes[i]) + ") ; \n \n")
        else:
            f.write(str(scaled_amplitudes[i]) + ", ")

# Save the bin bandwidth
    f.write((root)+"_bandwidths = (\n" + str(bin_bandwidth) + "\n) ; \n")
```

Listing 16: Algoritmo della DFT in Python

Questo codice prende come argomenti: un file audio .wav in ingresso, un range compreso fra una frequenza minima e massima, il numero di bin da utilizzare per l'analisi DFT, ed infine il numero di picchi ordinati per frequenze dalle più alte in ampiezza che si vogliono in output con i relativi valori di ampiezze, con il bandwidth

dell'analisi. E in queste liste in output: frequenze, ampiezze e bandwidth, i valori di frequenza in uscita dall'analisi simili a quelli già salvati in un intervallo di 40Hz, vengono eliminati, e i restanti picchi principali salvati nel file .lib

4.3 Feedback Networks

Per concludere questa osservazione sulla struttura sistemica di RITI, discuterò brevemente, ripetendo alcuni concetti fondamentali illustrati ad inzio capitolo. I tre stati di feedback principale su cui si basa questo sistema sono: Il feedback interno alle equazioni differenziali, o chiamato anche come fattore di magnificazione; ne possiamo vedere un implementazione in questo paper[45]. Questo ci permette di poter portare le equazioni di Lorenz da uno stato di autoscillazione al solo funzionamento come funzione di trasferimento (waveshaping) per il segnale in ingresso. Il feedback della FDN, permette ai segnali presenti nella rete di ricircolare con opportuni tempi di ritardo, così da avere delle decorrelazioni fra i CSGA, oltre che dei comportamenti del sistema che possano avere una loro storia nel tempo, anche nel tornare quando il fattore di magnificazione di Lorenz è pari a 0 così da poter apprezzare solo la storia del sistema all'interno della rete senza avere nuovi contributi. Ed infine, poiché ho dotato le equazioni di Lorenz di un ingresso esterno che viene sfruttato sia dalla reiniezione dei segnali provenienti dalla FDN, che dall'aggiunta di 4 microfoni esterni, il sistema viene aperto all'ambiente: perturbato dagli stessi segnali che produce all'interno della stanza, che ne variano il comportamento nel tempo, e da fenomeni di Larsen che emergono quando il fattore di magnificazione che produce l'autoscillazione in Lorenz ha un contributo energeticamente inferiore rispetto a quello del Larsen proveniente dalla stanza.

[45] J. Liang and W. Song. "DIFFERENCE EQUATION OF LORENZ SYSTEM". en. In: *International Journal of Pure and Apllied Mathematics* 83.1 (Feb. 2013). ISSN: 1311-8080, 1314-3395. DOI: 10.12732/ijpam.v83i1.9. URL: http://www.ijpam.eu/contents/2013-83-1/9/ (visited on 11/22/2022).

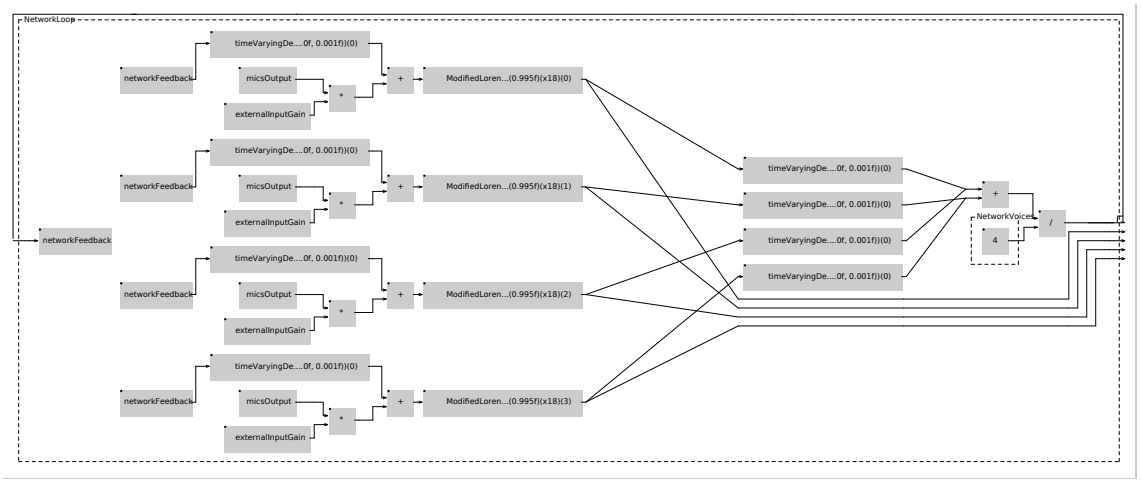

Figure 21: Esempio della Rete di RITI compilata con 4 Voci (4 CSGA)

La possibilità di regolare i tre stati di feedback e di portare il sistema "sulla soglia del caos" esplorando nella performance tutti i suoi spazi, rivolgendo gli altoparlanti verso le pareti trasformando così l'ambiente stesso nello spazio di sistema, rappresenta per me un estensione: del concetto di strumento musicale, dell'interazione con la macchina, ed infine dell'utopia di un nuovo ascolto, in un mondo dove è sempre più necessario prendere consapevolezza della complessità che ci circonda.

4.4 Motivazioni Personali

Prima di concludere questo elaborato, vorrei prima soffermarmi a spiegare alcune delle motivazioni personali che hanno suscitato in me l'interesse per i sistemi complessi adattivi in live electronics, e che mi hanno spinto in particolare alla creazione di questo brano.

In questi ultimi anni di studi musicali, sono emerse nel mio fare musica delle questioni per me di fondamentale importanza che hanno accompagnato il mio operato e che hanno mosso il mio interesse verso i le cibernetiche in musica.

In primo luogo indagare rispetto al rapporto che esiste fra l'interprete, la notazione, e lo strumento musicale; che può essere vista in qualche modo come una parte contenuta all'interno della relazione fra uomo-macchina-ambiente di cui parla Agostino

Di Scipio.

Le prime forme di semiografia musicale possono essere rintracciate in diverse civiltà che risalgono addirittura attorno al 2000 A.C., sin d'allora l'uomo ha posto al centro della sua indagine l'interrogativo sul come fissare sempre meglio successioni strutturate di suoni semplici o complessi tramite una serie di parametri nel dominio discreto (la notazione), e che interpretati in un secondo momento potessero restituire l'idea iniziale in una resa acustica nel dominio del continuum temporale. Questi parametri principalmente impiegati nella semiografia musicale Occidentale, sono stati fino all'incirca ad inizio del XX Secolo: le altezze dei suoni, le intensità, i ritmi, le durate, ed infine il timbro desiderato, quest'ultimo descritto richiedendo l'impiego di un certo strumento musicale o di specifici gruppi strumentali. Tuttavia il significato stesso del termine musica non è comunque univoco ed ha avuto (ed ha ancora ad oggi) diverse accezioni utilizzate nei vari periodi storici. Difatti con l'arrivo del XXI Secolo, sono state messe in crisi le nozioni di strumento musicale e di semiografia musicale, in quanto la pratica del comporre non si è più solamente limitata al comporre con i timbri dei diversi strumenti musicali, ma si è aperta anche alla possibilità del comporre stesso nel timbro, spingendo i compositori a cercare vie alternative ed alle volte individuali in cui scrivere musica differente dalla tradizione. Riguardo gli strumenti musicali invece, se la voce è difatti il primo strumento dell'uomo in quanto abita il corpo stesso che ne è il custode, la macchina che nel suo senso generico è qualsiasi strumento il cui moto relativo trasmetta o anche amplifichi la forza umana. Sin dall'antichità la macchina si è prestata all'impiego come strumento musicale sottoforma di qualcosa per assistere l'uomo nella sua intenzione del fare musica secondo l'organizzazione dei parametri esposti precedentemente; con la composizione nel timbro viene invece a mancare l'idea di una musica composta solamente da queste tipiche relazioni tramite gli strumenti musicali, rivalutando l'idea stessa che si ha di uno strumento musicale, aperta ora di fatto alla possibilità del poter comporre quanto il timbro stesso integrando anche la creazione dello strumento che lo produce.

Come dice Trevor Wishart nel suo libro On Sonic Art:

In this book i will suggest that the logic of this assertion is inverted. It is notability which determines the importances of pitch, rythm, and duration and not vice versa and that much can be learned by looking at musical cultures without a system of notation... In this book, I will suggest that we do not need to deal with a finite set of possibilities. The idea that music has to be built upon finite lattice and the related idea that permutational procedures are a valid way to proceed will be criticised here and a musical methodology developed for dealing with a continuum using the concept of transformation[46]

Come comportarsi quindi davanti ad una condizione in cui viene messa in crisi l'idea stessa che si ha di musica, di notazione musicale, e di strumento musicale? Questa indagine mi ha portato a rivolgere l'attenzione al rapporto con la notazione e gli strumenti musicali in quelle opere dei compositori contemporanei, che raccogliendo le ceneri di questa crisi del XXI Secolo, hanno deciso di approfondire l'idea dietro l'utilizzo di uno strumento musicale con cui fare musica, non più come qualcosa di funzionale al riprodurre dei parametri annotati in partitura a priori, ma come un sistema che offre una serie di possibilità; in cui lo strumento diviene il corpo acustico su cui indagare tutta una serie di relazioni e gesti possibili dell'interprete, alla ricerca di un'esplorazione sonora dello spazio del sistema, e in cui la partitura diviene traccia del come replicare l'esistenza di qualsiasi tipo d'interazione con il sistema. In tal senso Trevor Wishart suggerisce di guardare alla Teoria delle Catastrofi di René Thom, assumendo che le regole formali di una musica passono essere ricavate dall'osservazione di uno strumento stesso come se questo fosse un sistema. Nella mia indagine per portare dei validi esempi, ho dunque rintracciato questo tipo di atteggiamento nei confronti dello strumento e della notazione in alcune opere come: Pression di Helmut Lachenmann, dove la partitura viene ad indicare come agire sul corpo dello strumento schiudendone nella sua totalità il suo mondo sonoro

[46]Trevor Wishart. *On Sonic Art.* en. Edinburgh: harwood academic publishers, 1996, p. 355.

accessibile, o nell'opera Necessità d'interrogare il cielo di Giorgio Netti, dove a partire dagli elementi della notazione tradizionale, lo studio condotto sul corpo dello strumento e degli esiti acustici del gesto, scendono così tanto "ad un basso livello di articolazione" da schiuderne in questo tutta una serie di nuove possibilità acustiche, o anche nei lavori stessi di Wishart, come nel ciclo Vox dove la voce viene esplorata in tutte le sue ambiguità rendendo complesso il collegamento fra il suono e la sorgente che lo produce.

In senso più generale la storia della musica elettroacustica è fortunatamente costellata di pionieri che hanno messo in crisi il linguaggio musicale e l'impiego degli strumenti musicali, ma l'ulteriore scarto che propongo qui, consiste nel fatto che se è vero quindi come dice Trevor Wishart che le regole formali di una musica possono essere ricavate dall'osservazione di un sistema, la pratica del disegnare un sistema autonomo (la macchina) con cui fare musica, ne riflette di conseguenza la quantità di informazione che questo contiene al suo interno, e di conseguenza il tipo di relazioni e interazioni con il suo ambiente che ci interessano dal punto di vista compositivo.

In secondo luogo, ho trovato interessante indagare rispetto al rapporto conflittuale che l'uomo ha con la tecnologia nella società contemporanea. In effetti, l'uomo ha da sempre un rapporto conflittuale con la tecnologia, se sin dai tempi di Platone emergevano queste problematiche e preoccupazioni relative alla discretizzazione del pensiero con la scrittura (come possiamo leggere in Fedro), giungendo alle paure più attuali di Norbert Weiner di fronte alla scoperta della cibernetica, possiamo dire che la sua paura di una società contemporanea composta da interazioni sistemiche fra l'uomo e la macchina in cui il primo è succube della seconda, sembra in qualche modo essere stata profetica ed essersi realizzata. Come scrive Norbert Wiener nel suo libro Introduzione alla cibernetica. L'uso umano degli esseri umani, inizialmente pubblicato con il titolo The Human Use Of Human Beings: Cybernetics And Society[47] nel 1950:

La nostra concezione della società ideale differisce dalla società ideale prospettata

[47]Norbert Wiener. *The Human Use of Human Beings.* en. 1950, p. 224.

dai fascisti e da molti magnati del mondo degli affari e della politica. Essi preferiscono una organizzazione in cui tutti i comandi provengano dall'alto senza che sia possibile nessuna riverisbilità. Sotto di essi gli uomini sono stati ridotti al livello di esecutori degli ordini di un centro nervoso che pretende di essere superiore. Desidero che questo libro sia inteso come una protesta contro questa utilizzazione inumana degli esseri umani, poiché sono convinto che impiegare un uomo richiedendogli e attribuendogli meno di quanto comporta la sua condizione umana, significa abbruttire questa condizione e sperperare le sue energie. È una degradazione della condizione umana legare un uomo a un remo e impiegarlo come sorgente di energia; ma è altrettanto degradante segregarlo in una fabbrica e assegnarlo a un compito meramente meccanico che richieda meno di un milionesimo delle sue facoltà cerebrali. È assai più facile infatti organizzare una fabbrica o una galera che impieghi gli esseri umani per una insignificante frazione delle loro attitudini che costruire una società in cui essi possano elevarsi in tutta la loro statura. Coloro che soffrono di un complesso di potenza sanno che la meccanizzazione dell'uomo è il mezzo più semplice per realizzare le loro ambizioni. Sono convinto che questa facile via al potere comporta non soltato l'annullamento di quelli che io credo siano i valori morali dell'umanità, ma anche l'eliminazione delle attuali, labilissime possibilità di sopravvivenza della razza umana per un periodo considerevole.

Se in risposta a questa affermazione di Wiener passiamo ad una seconda citazione, proveniente invece da un libro recente che critica la nostra condizione nella società contemporanea, scritto e pubblicato da François J. Bonnet. nel 2020:

The increasingly frenetic accumulation of sounds and images is a part of this intensification, aiming to fill the void of a present that is forever sinking into the past. At every moment, the world must be supplied with novel sensory material generated to replace existing material that has already become obsolete. The exaltation of the present as the promise of pleasure Is therefore never really meant

to be actualised, since it will immediately be obsolesced by a continuously renewed influx of promises. Needs, cravings, and desires are endlessly rebooted. The present moment, as constructed by our society, is nothing but a procedure of forgetting, a catalyst for amnesia, and as such it enables the repetition which, in turn, through the presentation of novelty, allows us to distance ourselves 3 little further from the past. And this writing and rewriting of the present—as if it were a palimpsest—continues to accelerate, demanding that we forget at an ever higher fre- quency. What was is of no concern; all has beens are eliminated in favour of what is, with little regard for what will be. Where the modern era anticipated a present yet to come, transposing current momentum into future accomplishments, our postindustrial, postmodern era sees the future as nothing but a vague, indeterminate site hosting an indistinct cloud of promises and signs as desirable as they are deadly.[48]

Troveremo che non solo le paure profetiche di Wiener si sono realizzate già da diverso tempo, ma che da molto tempo ne stiamo già raccogliendo le inevitabili conseguenze, che si riflettono dal mio punto di vista in una società anestetizzata, schiava della tecnologia, e livellata in qualche modo ad una condizione di eterno presente. In risposta a questa condizione, disegnare i propri sistemi e di conseguenza riappropriarsi della tecnologia, rappresenta per me un modo costruttivista di rispondere ai problemi del mondo contemporaneo attraverso la pratica musicale.

Per concludere infine le motivazioni che mi hanno portato a questo tipo di scrittura musicale, l'ultimo aspetto che ho reputato importante affrontare è la forma, nei suoi aspetti Macro-temporali e Micro-temporali.

Se ammettiamo quindi come sostiene François J. Bonnet, ed altri filosofi come Mark Fisher e Slavoj Žižek, che la nostra società contemporanea ha in qualche modo normalizzato una velocità schizofrenica nelle cose, dove tutto deve essere estremamente funzionale per motivi pratici, e dove non abbiamo più modo di esercitare alcun controllo sullo scorrere del tempo, la riappropriazione del tempo nella forma,

[48] François J. Bonnet. *After Death.* en. Urbanomic, 2020, p. 80.

dell'ascolto, ed infine dello spazio, diviene per me un elemento fondamentale nella composizione, dove tutti gli elementi del sistema hanno un ruolo condiviso che contribuisce a formare la complessità nel suo insieme. A tal proposito ho sempre trovato interessante osservare nelle evoluzioni temporali come un piccolo cambiamento all'interno di un sistema possa conseguentemente portare a cambiamenti radicali nella Macroforma; proprio come nella sensibilità alle condizioni della teoria del caos, che ci illustra come in un sistema caotico, a variazioni infinitesime delle condizioni iniziali corrispondono variazioni significative del comportamento futuro. Dunque in tal senso è diventato per me un aspetto importante lavorare con sistemi emergenti che possano rendere udibili questi processi di continua trasformazione del suono, e che non rendano udibile una sola traiettoria possibile nell'evoluzione formale di un brano.

È a tal proposito che Stockhausen, nella sua teoria della "Forma unificata del tempo", notò già nel 1960 come in tal senso la differenza fra i Micro-elementi nella forma, e la Macroforma, non consista tanto in diversi ruoli e compiti del comporre, quanto più a diversi livelli di percezione del tempo, poiché altezza e ritmo sono fondamentalmente la stessa cosa se percepita a diverse scale temporali, e ce lo illustra nella sua composizione Kontakte realizzata fra il 1958 ed il 1960, mettendo in certi termini in luce, come nella forma di un brano musicale il ruolo delle parti sia sostanzialmente un problema di complessità che dipende dal punto di vista dell'osservatore.

4.5 Conclusioni

In conclusione, abbiamo avuto modo di introdurre l'argomento delle cibernetiche in Musica, parlando di come l'esigenza di un pensiero sistemico abbia pian piano fatto breccia nell'operato di alcuni dei più importanti compositori del XX Secolo. Arrivando fino ad osservare e studiare il realizzarsi di questa condizione attraverso l'operato di Dario Sanfilippo e da Agostino Di Scipio, di cui abbiamo approfondito alcuni dei principali meccanismi presenti nei loro lavori compositivi, nel desiderio di creare attraverso questi modelli una strada da seguire per la composizione dei

sistemi complessi adattivi per la performance musicale in live electronics. Ed Infine mettendo in luce l'importanza di una prospettiva ontologica ed epistemeologica nella composizione musicale sistemica.

A First appendix

In questo appendice sono riportati tutti i codici dei sistemi e le relative librerie di oggetti trattati nel corso della tesi.

`Audible_Ecosystemics_2.dsp` è il file principale di una implementazione completa del sistema di Audible Ecosystemics 2 di Agostino Di Scipio, sviluppato con il supporto e il prezioso aiuto di Dario Sanfilippo, nel Debug del sistema e sviluppo della libreria del codice; e conseguente al prezioso lavoro svolto con i miei compagni nella Classe di esecuzione ed interpretazione della musica elettroacustica di Giuseppe Silvi, che mi ha permesso di approfondire le tematiche discusse nel corso della tesi riguardo al brano e poter sviluppare strategie efficaci per questa implementazione.

`aelibrary.lib` è invece la libreria degli oggetti utilizzati nella implementazione del sistema di Audible Ecosystemics 2 di Agostino Di Scipio (oggetti discussi anche nella tesi).

`RITI_v1_CelloC2.dsp` è il file principale della implementazione del mio sistema di RITI v.1.0. `RITI.lib` è invece la libreria degli oggetti utilizzati nella implementazione del mio sistema di RITI v.1.0. `SpectreLists.lib` è infine una libreria con i risultati delle analisi condotte sulle note del violoncello, utilizzata per passare le informazioni ai filtri passabanda nel sistema di RITI v.1.0.

Audible_Ecosystemics_2.dsp

```
declare name "Agostino Di Scipio - AUDIBLE ECOSYSTEMICS n.2";
declare author "Luca Spanedda";
declare author "Dario Sanfilippo";
declare version "alpha";
declare description " 2022 version - Realised on composer's instructions
    of the year 2017 edited in LAquila, Italy";

// import faust standard library
import("stdfaust.lib");
// import audible ecosystemics objects library
import("aelibrary.lib");



//------- ------------- ----- -----------
//-- AE2 ---------------------------------------------------------------------
//------- --------



// MAIN SYSTEM FUNCTION
outputrouting(grainOut1, grainOut2, out1, out2, out3, out4, out5, out6, mic1,
    mic2, mic3, mic4, diffHL, memWriteDel1, memWriteDel2, memWriteLev, cntrlLev1,
    cntrlLev2, cntrlFeed, cntrlMain, cntrlMic1, cntrlMic2, directLevel,
    timeIndex1, timeIndex2, triangle1, triangle2, triangle3, sampWOut, sig1,
    sig2, sig3, sig4, sig5, sig6, sig7) =
out1, out2, out3, out4, out5, out6; // choose here the signals in output

process =
    si.bus(8) :> si.bus(4) :
    (signalflow1a : signalflow1b : signalflow2a : signalflow2b : signalflow3) ~
        si.bus(2) :
       outputrouting;

signalflow1a( grainOut1, grainOut2, mic1, mic2, mic3, mic4 ) =
```

```
grainOut1, grainOut2,
mic1, mic2, mic3, mic4,
((diffHL, memWriteDel1, memWriteDel2, memWriteLev, cntrlLev1, cntrlLev2,
    cntrlFeed, cntrlMain) : SF1Ainspect)
with {
    Mic_1A_1 = mic3 : gainMic_1A1;
    Mic_1A_2 = mic4 : gainMic_1A2;
    map6sumx6 = (Mic_1A_1 : integrator(.01) : delayfb(.01, .95)) +
              (Mic_1A_2 : integrator(.01) : delayfb(.01, .95)) :
                  \(x).(6 + x * 6);

    localMaxDiff = ((map6sumx6, Mic_1A_1) : localmax) ,
                  ((map6sumx6, Mic_1A_2) : localmax) :
                      \(x, y).(x - y);

    SenstoExt = (map6sumx6, localMaxDiff) :
        localmax <: _ , (_ : delayfb(12, 0)) : + : * (.5) :
            LP1(.5) : tgroup("Control", vgroup("System Inspectors",
                      hgroup("Signal Flow 1a [1]",
                      hgroup("Sens. to Ext. Cond.",
                      inspect(101, -1, 1)))));

    diffHL = ((Mic_1A_1 + Mic_1A_2) : HP3(var2) : integrator(.05)) ,
              ((Mic_1A_1 + Mic_1A_2) : LP3(var2) : integrator(.10)) :
                  \(x, y).((x - y) : tgroup("Control", vgroup("System Inspectors",
                                   hgroup("Signal Flow 1a [1]",
                                   hgroup("diffHL Centroid",
                                   inspect(100, -1, 1))))) *
                      (1 - SenstoExt) : delayfb(.01, .995) :
                          LP5(25) : \(x).(.5 + x * .5) :
                              // LIMIT - max - min
                              limit(1, 0);

    memWriteLev = (Mic_1A_1 + Mic_1A_2) : integrator(.1) : delayfb(.01, .9) :
        LP5(25) : \(x).(1 - (x * x)) :
```

```
            // LIMIT - max - min
            limit(1, 0);

    memWriteDel1 = memWriteLev : delayfb((var1 / 2), 0) :
        // LIMIT - max - min
        limit(1, 0);

    memWriteDel2 = memWriteLev : delayfb((var1 / 3), 0) :
        // LIMIT - max - min
        limit(1, 0);

    cntrlMain = (Mic_1A_1 + Mic_1A_2) * SenstoExt : integrator(.01) :
        delayfb(.01, .995) : LP5(25) :
            // LIMIT - max - min
            limit(1, 0);

    cntrlLev1 = cntrlMain : delayfb((var1 / 3), 0) :
        // LIMIT - max - min
        limit(1, 0);

    cntrlLev2 = cntrlMain : delayfb((var1 / 2), 0) :
        // LIMIT - max - min
        limit(1, 0);

    cntrlFeed = cntrlMain : \(x).(ba.if(x <= .5, 1.0, (1.0 - x) * 2.0)) :
            // LIMIT - max - min
            limit(1, 0);
};

signalflow1b( grainOut1, grainOut2, mic1, mic2, mic3, mic4, diffHL, memWriteDel1,
    memWriteDel2, memWriteLev, cntrlLev1, cntrlLev2, cntrlFeed, cntrlMain ) =
mic1, mic2, mic3, mic4,
diffHL, memWriteDel1, memWriteDel2, memWriteLev, cntrlLev1, cntrlLev2, cntrlFeed,
    cntrlMain,
((cntrlMic1, cntrlMic2, directLevel, timeIndex1, timeIndex2, triangle1,
```

```
    triangle2, triangle3) : SF1Binspect)
with {
    Mic_1B_1 = mic1 : gainMic_1B1;
    Mic_1B_2 = mic2 : gainMic_1B2;
    // cntrlMic - original version
    cntrlMic(x) = x : HP1(50) : LP1(6000) :
        integrator(.01) : delayfb(.01, .995) : LP5(.5);

    // cntrlMic - alternative version
    // cntrlMic(x) = x : HP2(50) : LP1(6000) :
    // integrator(.01) : delayfb(.01, .995) : LP5(.04);
    cntrlMic1 = Mic_1B_1 : cntrlMic :
        // LIMIT - max - min
        limit(1, 0);

    cntrlMic2 = Mic_1B_2 : cntrlMic :
        // LIMIT - max - min
        limit(1, 0);

    directLevel =
        (grainOut1 + grainOut2) : integrator(.01) : delayfb(.01, .97) :
           LP5(.5) <:
                _ ,
               (_ : delayfb(var1 * 2, (1 - var3) * 0.5)) : + :
                   \(x).(1 - x * .5) :
                       // LIMIT - max - min
                       limit(1, 0);

    timeIndex1 = triangleWave(1 / (var1 * 2)) : \(x).((x - 2) * 0.5);

    timeIndex2 = triangleWave(1 / (var1 * 2)) : \(x).((x + 1) * 0.5);

    triangle1 = triangleWave(1 / (var1 * 6)) * memWriteLev;

    triangle2 = triangleWave(var1 * (1 - cntrlMain));
```

```
    triangle3 = triangleWave(1 / var1);
};

signalflow2a( mic1, mic2, mic3, mic4, diffHL, memWriteDel1, memWriteDel2,
    memWriteLev, cntrlLev1, cntrlLev2, cntrlFeed, cntrlMain, cntrlMic1,
    cntrlMic2, directLevel, timeIndex1, timeIndex2, triangle1, triangle2,
    triangle3 ) =
mic1, mic2, mic3, mic4,
diffHL, memWriteDel1, memWriteDel2, memWriteLev, cntrlLev1, cntrlLev2, cntrlFeed,
    cntrlMain,
cntrlMic1, cntrlMic2, directLevel, timeIndex1, timeIndex2, triangle1, triangle2,
    triangle3,
((sampWOut, sig1, sig2, sig3, sig4, sig5, sig6, sig7) : SF2Ainspect)
with {
    Mic_2A_1 = mic1 : gainMic_2A1;
    Mic_2A_2 = mic2 : gainMic_2A2;
    micIN1 = Mic_2A_1 : HP1(50) : LP1(6000) *
        (1 - cntrlMic1);

    micIN2 = Mic_2A_2 : HP1(50) : LP1(6000) *
        (1 - cntrlMic2);

    SRSect1(x) = x : sampler(var1, (1 - memWriteDel2), (var2 + (diffHL * 1000)) /
        261) :
        SRinspect(1) : HP4(50) : delayfb(var1 / 2, 0);

    SRSect2(x) = x : sampler(var1, (memWriteLev + memWriteDel1) / 2, (290 -
        (diffHL * 90)) / 261) :
        SRinspect(2) : HP4(50) : delayfb(var1, 0);

    SRSect3(x) = x : sampler(var1, (1 - memWriteDel1), ((var2 * 2) - (diffHL *
        1000)) / 261) :
        SRinspect(3) : HP4(50);
```

```
SRSectBP1(x) = x : SRSect3 : BPsvftpt(diffHL * 400, (var2 / 2) * memWriteDel2);

SRSectBP2(x) = x : SRSect3 : BPsvftpt((1 - diffHL) * 800, var2 * (1 -
    memWriteDel1));

SRSect4(x) = x : sampler(var1, 1, (250 + (diffHL * 20)) / 261) : SRinspect(4);

SRSect5(x) = x : sampler(var1, memWriteLev, .766283) : SRinspect(5);

SampleWriteLoop = loop ~ _
with {
    loop(fb) =
        (
            ( SRSect1(fb) ,
                SRSect2(fb) ,
                SRSectBP1(fb) ,
                SRSectBP2(fb) :> +
            ) * (cntrlFeed * memWriteLev)
        ) <:
        (
            _ + (micIN1 + micIN2) : _ * triangle1 ),
            _ ,
            SRSect4(fb) ,
            SRSect5(fb) ,
            SRSect3(fb) ;
};

sig1 = micIN1 * directLevel;

sig2 = micIN2 * directLevel;

sampWOut = SampleWriteLoop : \(A,B,C,D,E).( A );

variabledelaysig3(x) = x : de.delay(max(0, ba.sec2samp(.05)), max(0,
    int(ba.sec2samp(.05 * cntrlMain))) );
```

```
    sig3 = SampleWriteLoop : \(A,B,C,D,E).( B ) : _ *
        memWriteLev : variabledelaysig3 * triangle2 * directLevel;

    sig4 = SampleWriteLoop : \(A,B,C,D,E).( B ) : _ *
        memWriteLev * (1-triangle2) * directLevel;

    sig5 = SampleWriteLoop : \(A,B,C,D,E).( C ) :
        HP4(50) : delayfb(var1 / 3, 0);

    sig6 = SampleWriteLoop : \(A,B,C,D,E).( D ) :
        HP4(50) : delayfb(var1 / 2.5, 0);

    sig7 = SampleWriteLoop : \(A,B,C,D,E).( E ) : delayfb(var1 / 1.5, 0) *
        directLevel;
};

signalflow2b( mic1, mic2, mic3, mic4, diffHL, memWriteDel1, memWriteDel2,
    memWriteLev, cntrlLev1, cntrlLev2, cntrlFeed, cntrlMain, cntrlMic1,
    cntrlMic2, directLevel, timeIndex1, timeIndex2, triangle1, triangle2,
    triangle3, sampWOut, sig1, sig2, sig3, sig4, sig5, sig6, sig7 ) =
mic1, mic2, mic3, mic4,
diffHL, memWriteDel1, memWriteDel2, memWriteLev, cntrlLev1, cntrlLev2, cntrlFeed,
    cntrlMain,
cntrlMic1, cntrlMic2, directLevel, timeIndex1, timeIndex2, triangle1, triangle2,
    triangle3,
sampWOut, sig1, sig2, sig3, sig4, sig5, sig6, sig7,
grainOut1, grainOut2,
out1, out2
with {
    grainOut1 =
        granular_sampling(var1, timeIndex1, memWriteDel1, cntrlLev1, 21, sampWOut)
            :
           GSinspect(1);

    grainOut2 =
```

```
        granular_sampling(var1, timeIndex2, memWriteDel2, cntrlLev2, 20, sampWOut)
            :
           GSinspect(2);


    out1 =
        (
           ((sig5 : delayfb(.040, 0)) * (1 - triangle3)),
            (sig5 * triangle3),
           ((sig6 : delayfb(.036, 0)) * (1 - triangle3)),
           ((sig6 : delayfb(.036, 0)) * triangle3 ),
             sig1,
             0,
             sig4,
             grainOut1 * (1 - memWriteLev) + grainOut2 * memWriteLev
        ) :> _ : gainMic_3O1;


    out2 =
        (
            (sig5 * (1 - triangle3)),
           ((sig5 : delayfb(.040, 0)) * triangle3),
            (sig6 * (1 - triangle3)),
            (sig6 * triangle3),
             sig2,
             sig3,
             sig7,
             grainOut1 * memWriteLev + grainOut2 * (1 - memWriteLev)
        ) :> _ : gainMic_3O2;
};


signalflow3( mic1, mic2, mic3, mic4, diffHL, memWriteDel1, memWriteDel2,
    memWriteLev, cntrlLev1, cntrlLev2, cntrlFeed, cntrlMain, cntrlMic1,
    cntrlMic2, directLevel, timeIndex1, timeIndex2, triangle1, triangle2,
    triangle3, sampWOut, sig1, sig2, sig3, sig4, sig5, sig6, sig7, grainOut1,
    grainOut2, out1, out2 ) =
grainOut1, grainOut2,
```

```
(
( out1, out2,
( out2 : delayfb((var4 / 2) / 344, 0)),
( out1 : delayfb((var4 / 2) / 344, 0)),
( out1 : delayfb(var4 / 344, 0)),
( out2 : delayfb(var4 / 344, 0)) )
: SF3Oinspect
),
mic1, mic2, mic3, mic4,
diffHL, memWriteDel1, memWriteDel2, memWriteLev, cntrlLev1, cntrlLev2, cntrlFeed,
    cntrlMain,
cntrlMic1, cntrlMic2, directLevel, timeIndex1, timeIndex2, triangle1, triangle2,
    triangle3,
sampWOut, sig1, sig2, sig3, sig4, sig5, sig6, sig7;
```

aelibrary.lib

```
declare name "Agostino Di Scipio - AUDIBLE ECOSYSTEMICS n.2";
declare author "Luca Spanedda";
declare author "Dario Sanfilippo";
declare version "alpha";
declare description " 2022 version - Realised on composer's instructions
    of the year 2017 edited in LAquila, Italy";

// import faust standard library
import("stdfaust.lib");

// PERFORMANCE SYSTEM VARIABLES
SampleRate = 44100;
var1 = nentry("t:Control/h:System Variables/Var 1", 8.0, 1, 20, 1);
var2 = nentry("t:Control/h:System Variables/Var 2", 50, 1, 10000, 1);
var3 = nentry("t:Control/h:System Variables/Var 3", .25, 0, 1, .001);
var4 = nentry("t:Control/h:System Variables/Var 4", 8.2, 1, 20, 1);



//------- ------------- ----- -----------
//-- LIBRARY ------------------------------------------------------------------
//------- --------



//-------------------------------------------------------------- UTILITIES --
// limit function for library and system
limit(maxl,minl,x) = x : max(minl, min(maxl));

//------------------------------------------------------------------ DELAYS --
delayfb(delSec,fb,x) = loop ~ _ : mem
with{
    loop(z) = ( (z * fb + x) @(max(0, ba.sec2samp(delSec)-1)) );
};
```

```
//-------------------------------------------------------------- SAMPLEREAD --
sampler(lengthSec, memChunk, ratio, x) =
    it.frwtable(3, 20 * SampleRate, .0, writePtr, x, readPtr) * window
    with {
        memChunkLimited = max(0.010, min(1, memChunk));
        bufferLen = lengthSec * SampleRate;
        writePtr = ba.period(bufferLen);
        grainLen = max(1, ba.if(writePtr > memChunkLimited * bufferLen,
            memChunkLimited * bufferLen, 1));
        readPtr = y
            letrec {
                'y = (ratio + y) % grainLen;
            };
        window = min(1, abs(((readPtr + grainLen / 2) % grainLen) -
            grainLen / 2) / 200);
    };

//-------------------------------------------------------------- INTEGRATOR ---
integrator(seconds, x) = an.abs_envelope_tau(limit(1000,.001,seconds), x);

//-------------------------------------------------------------- LOCALMAX ---
localMax(seconds, x) = loop ~ si.bus(4) : _ , ! , ! , !
with {
    loop(yState, timerState, peakState, timeInSamplesState) =
        y , timer , peak , timeInSamples
    with {
        timeInSamples = ba.if(reset + 1 - 1', seconds *
            ma.SR, timeInSamplesState);
        reset = timerState >= (timeInSamplesState - 1);
        timer = ba.if(reset, 1, timerState + 1);
        peak = max(abs(x), peakState * (1.0 - reset));
        y = ba.if(reset, peak', yState);
    };
};
```

```
localmax(resetPeriod, x) = localMax(limit(1000,0,resetPeriod), x);

//-------------------------------------------------------------- TRIANGLE ---
triangularFunc(x) = abs(ma.frac((x - .5)) * 2.0 - 1.0);
triangleWave(f) = triangularFunc(os.phasor(1,f));

//-------------------------------------------------------------- FILTERS ---
onePoleTPT(cf, x) = loop ~ _ : ! , si.bus(3)
with {
    g = tan(cf * ma.PI * (1/ma.SR));
    G = g / (1.0 + g);
    loop(s) = u , lp , hp , ap
    with {
        v = (x - s) * G;
        u = v + lp;
        lp = v + s;
        hp = x - lp;
        ap = lp - hp;
    };
};

SVFTPT(Q, cf, x) = loop ~ si.bus(2) : (! , ! , _ , _ , _ , _ , _)
    with {
        g = tan(cf * ma.PI * (1.0/ma.SR));
        R = 1.0 / (2.0 * Q);
        G1 = 1.0 / (1.0 + 2.0 * R * g + g * g);
        G2 = 2.0 * R + g;
        loop(s1, s2) = u1 , u2 , lp , hp , bp , bp * 2.0 * R , x - bp * 4.0 * R
            with {
                hp = (x - s1 * G2 - s2) * G1;
                v1 = hp * g;
                bp = s1 + v1;
                v2 = bp * g;
                lp = s2 + v2;
                u1 = v1 + bp;
```

```
            u2 = v2 + lp;
        };
    };

SVFTPT2(K, Q, CF, x) = circuitout : ! , ! , _ , _ , _ , _ , _ , _ , _ , _
with{
    g = tan(CF * ma.PI / ma.SR);
    R = 1.0 / (2.0 * Q);
    G1 = 1.0 / (1.0 + 2.0 * R * g + g * g);
    G2 = 2.0 * R + g;
    circuit(s1, s2) = u1 , u2 , lp , hp , bp, notch, apf, ubp, peak, bshelf
        with{
            hp = (x - s1 * G2 - s2) * G1;
            v1 = hp * g;
            bp = s1 + v1;
            v2 = bp * g;
            lp = s2 + v2;
            u1 = v1 + bp;
            u2 = v2 + lp;
            notch = x - ((2*R)*bp);
            apf = x - ((4*R)*bp);
            ubp = ((2*R)*bp);
            peak = lp -hp;
            bshelf = x + (((2*K)*R)*bp);
        };
    // choose the output from the SVF Filter (ex. bshelf)
    circuitout = circuit ~ si.bus(2);
};

LPTPT(CF, x) = onePoleTPT(max(ma.EPSILON, min(20480, CF)), x) : (_ , ! , !);
HPTPT(CF, x) = onePoleTPT(max(ma.EPSILON, min(20480, CF)), x) : (! , _ , !);

LPSVF(Q, CF, x) = SVFTPT2(0, Q, max(ma.EPSILON, min(20480, CF)), x) : _ , ! , ! ,
    ! , ! , ! , ! , ! ;
HPSVF(Q, CF, x) = SVFTPT2(0, Q, max(ma.EPSILON, min(20480, CF)), x) : ! , _ , ! ,
```

```
    ! , ! , ! , ! , !;

BPsvftpt(bw, cf, x) = Q , CF , x : SVFTPT : (! , ! , ! , _ , !)
    with {
        CF = max(20, min(20480, LPTPT(1, cf)));
        BW = max(1, min(20480, LPTPT(1, bw)));
        Q = max(.01, min(100, BW / CF));
    };

// Butterworth
butterworthQ(order, stage) = qFactor(order % 2)
with {
    qFactor(0) = 1.0 / (2.0 * cos(((2.0 * stage + 1) *
    (ma.PI / (order * 2.0)))));
    qFactor(1) = 1.0 / (2.0 * cos(((stage + 1) * (ma.PI / order))));
};

LPButterworthN(1, cf, x) = LPTPT(cf, x);
LPButterworthN(N, cf, x) = cascade(N % 2)
with {
    cascade(0) = x : seq(i, N / 2, LPSVF(butterworthQ(N, i), cf));
    cascade(1) = x : LPTPT(cf) : seq(i, (N - 1) / 2,
    LPSVF(butterworthQ(N, i), cf));
};

HPButterworthN(1, cf, x) = HPTPT(cf, x);
HPButterworthN(N, cf, x) = cascade(N % 2)
with {
    cascade(0) = x : seq(i, N / 2, HPSVF(butterworthQ(N, i), cf));
    cascade(1) = x : HPTPT(cf) : seq(i, (N - 1) /
    2, HPSVF(butterworthQ(N, i), cf));
};

// Filters Order Butterworth
LP1(CF, x) = x : LPButterworthN(1, CF);
```

```
HP1(CF, x) = x : HPButterworthN(1, CF);
LP2(CF, x) = x : LPButterworthN(2, CF);
HP2(CF, x) = x : HPButterworthN(2, CF);
LP3(CF, x) = x : LPButterworthN(3, CF);
HP3(CF, x) = x : HPButterworthN(3, CF);
LP4(CF, x) = x : LPButterworthN(4, CF);
HP4(CF, x) = x : HPButterworthN(4, CF);
LP5(CF, x) = x : LPButterworthN(5, CF);
HP5(CF, x) = x : HPButterworthN(5, CF);

// Filters Order in series
// LP1(CF, x) = x : LPTPT(CF);
// HP1(CF, x) = x : HPTPT(CF);
// LP2(CF, x) = x : LPTPT(CF) : LPTPT(CF);
// HP2(CF, x) = x : HPTPT(CF) : HPTPT(CF);
// LP3(CF, x) = x : LPTPT(CF) : LPTPT(CF) : LPTPT(CF);
// HP3(CF, x) = x : HPTPT(CF) : HPTPT(CF) : HPTPT(CF);
// LP4(CF, x) = x : LPTPT(CF) : LPTPT(CF) : LPTPT(CF) : LPTPT(CF);
// HP4(CF, x) = x : HPTPT(CF) : HPTPT(CF) : HPTPT(CF) : HPTPT(CF);
// LP5(CF, x) = x : LPTPT(CF) : LPTPT(CF) : LPTPT(CF) : LPTPT(CF) : LPTPT(CF);
// HP5(CF, x) = x : HPTPT(CF) : HPTPT(CF) : HPTPT(CF) : HPTPT(CF) : HPTPT(CF);

//-------------------------------------------------------- GRANULAR SAMPLING ---
grain(L, position, duration, x, trigger) = hann(phase) * buffer(readPtr, x)
with {
    maxLength = 20 * SampleRate;
    length = L * SampleRate;
    hann(ph) = sin(ma.PI * ph) ^ 2.0;
    lineSegment = loop ~ si.bus(2) : _ , ! , _
    with {
        loop(yState, incrementState) = y , increment , ready
        with {
            ready = ((yState == 0.0) | (yState == 1.0)) & trigger;
            y = ba.if(ready, increment, min(1.0, yState + increment));
            increment = ba.if(ready, (1.0/ma.SR) / max((1.0/ma.SR), duration),
```

```
                incrementState);
        };
    };
    phase = lineSegment : _ , !;
    unlocking = lineSegment : ! , _;
    lock(param) = ba.sAndH(unlocking, param);
    grainPosition = lock(position);
    grainDuration = lock(duration);
    readPtr = grainPosition * length + phase * grainDuration * ma.SR;
    buffer(readPtr, x) =
        it.frwtable(3, maxLength, .0, writePtr, x, readPtrWrapped)
    with {
        writePtr = ba.period(length);
        readPtrWrapped = ma.modulo(readPtr, length);
    };
};

// works for N >= 2
triggerArray(N, rate) = loop ~ si.bus(3) : (! , ! , _) <:
    par(i, N, == (i)) : par(i, N, \(x).(x > x'))
with {
    loop(incrState, phState, counterState) = incr , ph , counter
    with {
        init = 1 - 1';
        trigger = (phState < phState') + init;
        incr = ba.if(trigger, rate * (1.0/ma.SR), incrState);
        ph = ma.frac(incr + phState);
        counter = (trigger + counterState) % N;
    };
};

grainN(voices, L, position, rate, duration, x) = triggerArray(voices, rate) :
    par(i, voices, grain(L, position, duration, x));

granular_sampling(var1, timeIndex, memWriteDel, cntrlLev, divDur, x) =
```

```
    grainN(10, var1, position, rate, duration, x) :> _
with {
    rnd = no.noise;
    memPointerJitter = rnd * (1.0 - memWriteDel) * .01;
    position = timeIndex * (1.0 - ((1.0 - memWriteDel) * .01)) +
        memPointerJitter;
    density = 1.0 - cntrlLev;
    rate = 50 ^ (density * 2.0 - 1.0);
    grainDuration = .023 + (1.0 - memWriteDel) / divDur;
    duration = grainDuration + grainDuration * .1 * rnd;
};

//-------------------------------------------------------------------------- GUI ---
VHmetersEnvelope = abs : max ~ -(1.0/ma.SR) :
    max(ba.db2linear(-84)) : ba.linear2db;

inspect(i, lower, upper) = _ * 1000 <: _ ,
    vbargraph("sig_%2i [style:numerical]", lower * 1000, upper * 1000) :
        attach : _ / 1000;

diffDebug(x) = an.abs_envelope_tau(1, (x-x')) * (SampleRate/2);

SF2Ainspect = tgroup("Control", vgroup("System Inspectors", par(i, 8,
    hgroup("Signal Flow 2a [23]", _ : inspect(i+1, -1, 1)))));
SF1Binspect = tgroup("Control", vgroup("System Inspectors", par(i, 8,
    hgroup("Signal Flow 1b [22]", _ : inspect(i+1, -1, 1)))));
SF1Ainspect = tgroup("Control", vgroup("System Inspectors", par(i, 8,
    hgroup("Signal Flow 1a [21]", _ : inspect(i+1, -1, 1)))));
SF3Oinspect = tgroup("Control", vgroup("System Inspectors", par(i, 6,
    hgroup("Signal Flow 3 [24]", _ : inspect(i+1, -1, 1)))));

GSinspect(i, x) = x *
    si.smoo(ba.db2linear(vslider("t:Control/h:Internal/h:Granular Sampling/GS %i
    Gain [unit:db]", 0, -84, 24, .001))) <:
    attach(_, VHmetersEnvelope(_) : vbargraph("t:Control/h:Internal/h:Granular
```

```
        Sampling/GS %i [unit:dB]", -80, 20));

SRinspect(i, x) = x * si.smoo(ba.db2linear(vslider("t:Control/h:Internal/h:Sample
    Read/SR %i Gain [unit:db]", 0, -84, 24, .001))) <:
    attach(_, VHmetersEnvelope(_) : vbargraph("t:Control/h:Internal/h:Sample
        Read/SR %i [unit:dB]", -80, 20));

gainMic_1A1(x) = x * si.smoo(ba.db2linear(vslider("t:Control/h:Mixer/h:Signal
    Flow 1A/Mic 3 [10][unit:db]", 0, -84, 24, .001))) <:
    attach(_, VHmetersEnvelope(_) : vbargraph("t:Control/h:Mixer/h:Signal Flow
        1A/Mic 3 VH [unit:dB]", -80, 20));
gainMic_1A2(x) = x * si.smoo(ba.db2linear(vslider("t:Control/h:Mixer/h:Signal
    Flow 1A/Mic 4 [11][unit:db]", 0, -84, 24, .001))) <:
    attach(_, VHmetersEnvelope(_) : vbargraph("t:Control/h:Mixer/h:Signal Flow
        1A/Mic 4 VH [unit:dB]", -80, 20));
gainMic_1B1(x) = x * si.smoo(ba.db2linear(vslider("t:Control/h:Mixer/h:Signal
    Flow 1B/Mic 1 [12][unit:db]", 0, -84, 24, .001))) <:
    attach(_, VHmetersEnvelope(_) : vbargraph("t:Control/h:Mixer/h:Signal Flow
        1B/Mic 1 VH [unit:dB]", -80, 20));
gainMic_1B2(x) = x * si.smoo(ba.db2linear(vslider("t:Control/h:Mixer/h:Signal
    Flow 1B/Mic 2 [13][unit:db]", 0, -84, 24, .001))) <:
    attach(_, VHmetersEnvelope(_) : vbargraph("t:Control/h:Mixer/h:Signal Flow
        1B/Mic 2 VH [unit:dB]", -80, 20));
gainMic_2A1(x) = x * si.smoo(ba.db2linear(vslider("t:Control/h:Mixer/h:Signal
    Flow 2A/Mic 1 [14][unit:db]", 0, -84, 24, .001)))<:
    attach(_, VHmetersEnvelope(_) : vbargraph("t:Control/h:Mixer/h:Signal Flow
        2A/Mic 1 VH [unit:dB]", -80, 20));
gainMic_2A2(x) = x * si.smoo(ba.db2linear(vslider("t:Control/h:Mixer/h:Signal
    Flow 2A/Mic 2 [15][unit:db]", 0, -84, 24, .001)))<:
    attach(_, VHmetersEnvelope(_) : vbargraph("t:Control/h:Mixer/h:Signal Flow
        2A/Mic 2 VH [unit:dB]", -80, 20));
gainMic_3O1(x) = x * si.smoo(ba.db2linear(vslider("t:Control/h:Mixer/h:Signal
    Flow 3/Out1 [16][unit:db]", 0, -84, 24, .001)))<:
    attach(_, VHmetersEnvelope(_) : vbargraph("t:Control/h:Mixer/h:Signal Flow
        3/Out1 VH [unit:dB]", -80, 20));
```

```
gainMic_3O2(x) = x * si.smoo(ba.db2linear(vslider("t:Control/h:Mixer/h:Signal
    Flow 3/Out2 [17][unit:db]", 0, -84, 24, .001)))<:
    attach(_, VHmetersEnvelope(_) : vbargraph("t:Control/h:Mixer/h:Signal Flow
        3/Out2 VH [unit:dB]", -80, 20));


//---------------------------------- TEST SIGNALS (Like Decorrelated Larsen) ---
noise(initSeed) = LCG ~ _ : (_ / m)
with{
    // variables
    // initSeed = an initial seed value
    a = 18446744073709551557; // a large prime number
    c = 12345; // a small prime number, such as 12345
    m = 2 ^ 31; // 2.1 billion
    // linear_congruential_generator
    LCG(seed) = ((a * seed + c) + (initSeed-initSeed') % m);
};


Test(N) = par(i, N, fi.bandpass(4, 1000, 1020, noise((i+1) * 469762049)) * 400);
```

RITI_v1_CelloC2.dsp

```
// import faust standard library
import("stdfaust.lib");
// import RITI objects library
import("RITI.lib");



// SYSTEM INTERFACE ----------------------------------------
TGroup(x) = tgroup("Main", x);
    MixerGroup(x) = hgroup("Mixer", x);
        FiltersGroup(x) = hgroup("Bandpass Filters Bank", x);
            TFreqsGroup(x) = tgroup("Bank Voices", x);
            FreqsGroup(i, x) = hgroup("Voice_%i", x);
        FDNGroup(x) = hgroup("Feedback Delay Network", x);
            INDelayGroup(x) = hgroup("Input Delays", x);
            OUTDelayGroup(x) = hgroup("Output Delays", x);
            GainDelayGroup(x) = hgroup("Gains Network", x);
            TDelayGroup(x) = tgroup("Delay Times", x);
        InsOutsGroup(x) = hgroup("Inputs and Outputs", x);
        LorenzFuncGroup(x) = hgroup("Lorenz Equation Parameters", x);
    InspectorsGroup(x) = vgroup("Inspectors", x);

// SYSTEM VARIABLES ----------------------------------------
DelSecondsMax = 12;
SystemSpaceVarOUT = 2.8713;
SystemSpaceVarIN = 2.3132;
BPFOrder = 1;
BPFilters = 32;
NetworkVoices = 4;
InitDCBlockzero = 1;
InitDCBlockpole = 0.995;
InitX0 = 1.2;
InitY0 = 1.3;
InitZ0 = 1.6;
```

```
Dtf = TGroup((ba.db2linear(MixerGroup(LorenzFuncGroup(vslider("Dt [unit:dB]", 0,
    -60, 60, .001)))))) : onepoletau(2);
Sigmaf = TGroup((MixerGroup(LorenzFuncGroup(vslider("Sigma", 10, 1, 19, .001)))))
    : onepoletau(2);
Rhof = TGroup((MixerGroup(LorenzFuncGroup(vslider("Rho", 3.518, 2.8, 53.2,
    .001))))) : onepoletau(2);
Betaf = TGroup((MixerGroup(LorenzFuncGroup(vslider("Beta", 1.073, 0.2666, 5.066,
    .001))))) : onepoletau(2);
BPFilterBypassf = TGroup((MixerGroup(FiltersGroup(vslider("BP Bypass", 0, 0, 1,
    .001))))) : onepoletau(2);
BPFilterDirectf = TGroup((MixerGroup(FiltersGroup(vslider("BP Signal", 1, 0, 1,
    .001))))) : onepoletau(2);
GlobalBPFrequenciesf(i) = TGroup((16 ^
    MixerGroup(FiltersGroup(TFreqsGroup(FreqsGroup(i+1, vslider("Frequency
    [unit:Hz]", 0, -1, 1, .001)))))) : onepoletau(2);
GlobalBPBWf = TGroup((MixerGroup(FiltersGroup(vslider("Bandwidth [unit:Hz]", 1,
    1, 100, .001))))) : onepoletau(2);
ChngList1f(i) = TGroup((MixerGroup(FiltersGroup(TFreqsGroup(FreqsGroup(i+1,
    nentry("Change List 1", 1, 1, 4, 1)))))));
ChngList2f(i) = TGroup((MixerGroup(FiltersGroup(TFreqsGroup(FreqsGroup(i+1,
    nentry("Change List 2", 2, 1, 4, 1)))))));
ChngList3f(i) = TGroup((MixerGroup(FiltersGroup(TFreqsGroup(FreqsGroup(i+1,
    nentry("Change List 3", 3, 1, 4, 1)))))));
ChngList4f(i) = TGroup((MixerGroup(FiltersGroup(TFreqsGroup(FreqsGroup(i+1,
    nentry("Change List 4", 4, 1, 4, 1)))))));
Interpolations1f(i) = TGroup((MixerGroup(FiltersGroup(TFreqsGroup(FreqsGroup(i+1,
    vslider("Interpolations A", 0, 0, 1, .001)))))) : onepoletau(2);
Interpolations2f(i) = TGroup((MixerGroup(FiltersGroup(TFreqsGroup(FreqsGroup(i+1,
    vslider("Interpolations B", 0, 0, 1, .001)))))) : onepoletau(2);
Saturationf = TGroup((MixerGroup(LorenzFuncGroup(vslider("TanH [unit:TanH]", 50,
    1, 100, .001))))) : onepoletau(2);
LorenzFeedbackf = TGroup(MixerGroup(LorenzFuncGroup(vslider("Lorenz FB", 1, 0, 1,
    .001)))) : onepoletau(2);
OutputGainf = TGroup(((ba.db2linear(MixerGroup(InsOutsGroup(vslider("Master
```

```
    [unit:dB]", -80, -80, 0, .001))))) : \(x).( (x > ba.db2linear(-80)) * x ))) :
    onepoletau(2);
ExternalInputGainf =
    TGroup(((ba.db2linear(MixerGroup(InsOutsGroup(vslider("Externals [unit:dB]",
    -80, -80, 80, .001))))) :
    \(x).( (x > ba.db2linear(-80)) * x ))) : onepoletau(2);



// MODIFIED LORENZ SYSTEM ----------------------------------------


ModifiedLorenzSystem(x0, y0, z0, dt, beta, rho, sigma, bypassFilter,
    directFilter, filterPartials, filterOrder, globalFreq, globalAmps, globalBW,
    interpolation1, interpolation2, changeList1, changeList2, changeList3,
    changeList4, saturation, dcBlockzero, dcBlockpole, internalFeedback, i,
    externalInput) =
    (
        (
        par(i, 3, _ * internalFeedback) :
        TGroup(InspectorsGroup(hgroup("Lorenz Feedback", XYZinspect(i)))) :
            LorenzSystemEquations
        ) :
            par(i, 3, _ : dcblocker(dcBlockzero, dcBlockpole)) :
            TGroup(InspectorsGroup(hgroup("DC Blocker", XYZinspect(i)))) :
                par(i, 3, _ : saturator(saturation)) :
                TGroup(InspectorsGroup(hgroup("Hyperbolic Tangent",
                    XYZinspect(i)))) :
                    par(i, 3, _ : BandpassFiltersBank(bypassFilter, directFilter,
                        filterPartials, filterOrder, globalFreq, globalAmps,
                        globalBW, interpolation1, interpolation2, changeList1,
                        changeList2, changeList3, changeList4)) :
                    TGroup(InspectorsGroup(hgroup("Bandpass Filters",
                        XYZinspect(i))))
    ) ~ si.bus(3) :
        par(i, 3, / (max(saturation, ma.EPSILON))) :> _ / 3
    with {
```

```
        x_init = x0-x0'; y_init = y0-y0'; z_init = z0-z0';
        LorenzSystemEquations(x, y, z) =
            ((((x + externalInput) + sigma * (y - x) * dt + x_init))),
            ((((y + externalInput) + (rho * x - x * z - y) * dt + y_init))),
            ((((z + externalInput) + (x * y - beta * z) * dt + z_init)));
    };


// GLOBAL SYSTEM NETWORK ----------------------------------------

GlobalSystemNetwork(Mic1, Mic2, Mic3, Mic4) =
    (NetworkLoop ~ _ : (si.block(1), si.bus(NetworkVoices))) :
           par(i, NetworkVoices, _ : dcblocker(InitDCBlockzero, InitDCBlockpole) :
               normalization(1)) :
              par(i, NetworkVoices, _ * OutputGainf )
    with{
        NetworkLoop(networkFeedback) = par(i, NetworkVoices,
            (networkFeedback * (TGroup(MixerGroup(FDNGroup
            (TDelayGroup(GainDelayGroup(
            vslider("Voice_%i+1[2]", 1, 0, 1, .001)))))))) :
                (timeVaryingDelay(TGroup(MixerGroup(FDNGroup
                (TDelayGroup(INDelayGroup(vslider("Voice_%i+1[2] [unit:Sec]",
                (SystemSpaceVarIN * (NetworkVoices - i)), .001, DelSecondsMax,
                    .001)))))), 0) + (((Mic1,Mic2,Mic3,Mic4) :> _ / 4) *
                    ExternalInputGainf)) :
                   ModifiedLorenzSystem(InitX0, InitY0, InitZ0, Dtf, Betaf, Rhof,
                       Sigmaf, BPFilterBypassf, BPFilterDirectf, BPFilters,
                       BPFOrder,
                   GlobalBPFrequenciesf(i), 1, GlobalBPBWf, Interpolations1f(i),
                       Interpolations2f(i), ChngList1f(i), ChngList2f(i),
                       ChngList3f(i), ChngList4f(i), Saturationf, InitDCBlockzero,
                       InitDCBlockpole, LorenzFeedbackf, i)
                          )
                       ) <:
                       (par( i, NetworkVoices, _ :
```

```
                        timeVaryingDelay(TGroup(MixerGroup(FDNGroup
                    (TDelayGroup(OUTDelayGroup(vslider("Voice_%i+1[2]
                        [unit:Sec]",
                    SystemSpaceVarOUT * (i + 1), .001, DelSecondsMax, .001))))))
                        , 0)) :>
                       +/NetworkVoices), (si.bus(NetworkVoices));
    };


process = si.bus(8) :> si.bus(4) : GlobalSystemNetwork;
```

RITI.lib

```
// import faust standard library
import("stdfaust.lib");
// Import lists: Frequencies, Amps, Bandwidth
import("SpectreLists.lib");



//-------------------------------------------------------------- FUNCTIONS ---
// Give a distance in meters - Delay time in samples in output
meterstoSamps(Meters) = ((1000 / 343.1) * Meters) * (ma.SR / 1000);
ms2samp(t) = (t/1000) * ma.SR;
sec2samp(t) = t * ma.SR;

limit(maxl,minl,x) = x : max(minl, min(maxl));
XYZinspect(i, x, y, z) = hgroup("NetworkVoice_%i[2]", (Xvmeter((i+1), 0, 1000,
    x), Yvmeter((i+1), 0, 1000, y), Zvmeter((i+1), 0, 1000, z)));
Xvmeter(i, minx, maxx, x) = attach(x, x : peakenvelope(10) : vbargraph("X_%i[2]
    [style:numerical]", minx, maxx));
Yvmeter(i, minx, maxx, y) = attach(y, y : peakenvelope(10) : vbargraph("Y_%i[2]
    [style:numerical]", minx, maxx));
Zvmeter(i, minx, maxx, z) = attach(z, z : peakenvelope(10) : vbargraph("Z_%i[2]
    [style:numerical]", minx, maxx));
Vmeter(i, minx, maxx, x) = attach(x, x : peakenvelope(10) : vbargraph("%i[2]
    [style:numerical]", minx, maxx));
Hmeter(i, minx, maxx, x) = attach(x, x : peakenvelope(10) : hbargraph("%i[2]
    [style:numerical]", minx, maxx));

//---------------------------------------- CHAOTIC EQUATIONS CONSTRICTIONS ---
saturator(lim, x) = lim * ma.tanh(x / (max(lim, ma.EPSILON)));

dcblocker(zero, pole, x) = x : dcblockerout
    with{
        onezero = _ <: _,mem : _,*(zero) : -;
        onepole = + ~ * (pole);
```

```
        dcblockerout = _ : onezero : onepole;
    };

timeVaryingDelay(delTime, fbGain, x) = loop ~ _ : mem
with {
    loop(fb) = (fb * fbGain + x) :
        de.sdelay( ba.sec2samp(DelSecondsMax),
        1024,
        ba.sec2samp(delTime)-1 );
    };

//-------------------------------------------------------------- FILTERS ---
// Zavalishin's SVF BP FILTER
// optimized BP from the TPT version of the SVF Filter by Vadim Zavalishin
// reference : (by Will Pirkle)
// http://www.willpirkle.com/Downloads/AN-4VirtualAnalogFilters.2.0.pdf
BPSVF(glin, bw, cf, x) = loop ~ si.bus(2) : (! , ! , _)
    with {
        g = tan(cf * ma.PI * (1.0/ma.SR));
        Q = cf / max(ma.EPSILON, bw);
        R = 1.0 / (Q + Q);
        G = 1.0 / (1.0 + 2.0 * R * g + g * g);
        loop(s1, s2) = u1 , u2 , bp * glin
            with {
                bp = (g * (x - s2) + s1) * G;
                bp2 = bp + bp;
                v2 = bp2 * g;
                u1 = bp2 - s1;
                u2 = v2 + s2;
            };
    };
// Zavalishin's Onepole TPT Filter
// reference : same of BPSVF
onePoleTPT(cf, x) = loop ~ _ : ! , si.bus(3) // Outs: lp , hp , ap
with {
```

```
    g = tan(cf * ma.PI * (1.0/ma.SR));
    G = g / (1.0 + g);
    loop(s) = u , lp , hp , ap
    with {
        v = (x - s) * G; u = v + lp; lp = v + s; hp = x - lp; ap = lp - hp;
    };
};
// Lowpass TPT
LPTPT(cf, x) = onePoleTPT(cf, x) : (_ , ! , !);
// Highpass TPT
HPTPT(cf, x) = onePoleTPT(cf, x) : (! , _ , !);

// Spectre BP Filter Banks
BandpassFiltersBank(bypassFilter, directFilter, filterPartials, filterOrder,
    globalFreq, globalAmps, globalBW, interpolation1, interpolation2, chngList1,
    chngList2, chngList3, chngList4, x) = x <:
    par(i, filterPartials,
        seq(r, filterOrder,
            BPSVF(
                AmplitudesListinterpolate( (i + 1), chngList1, chngList2,
                    chngList3, chngList4, interpolation1, interpolation2) *
                    globalAmps,
                BandwidthsListinterpolate( (i + 1), chngList1, chngList2,
                    chngList3, chngList4, interpolation1, interpolation2) *
                    globalBW,
                FrequenciesListinterpolate( (i + 1), chngList1, chngList2,
                    chngList3, chngList4, interpolation1, interpolation2) *
                    globalFreq
            )
        )
    ):> (+ / filterPartials) * directFilter + (x * bypassFilter);

onepoletau(tau, x) = fb ~ _
with {
    fb(y) = (1.0 - s) * x + s * y;
```

```
    s = exp(-1.0/(tau * ma.SR));
    // tau = desired smoothing time constant in seconds
    };

//-------------------------------------------------------------- ANALIZERS ---
peakHolder(holdTime, x) = loop ~ si.bus(2) : ! , _
with {
    loop(timerState, outState) = timer , output
    with {
        isNewPeak = abs(x) >= outState;
        isTimeOut = timerState >= (holdTime * ma.SR - 1);
        bypass = isNewPeak | isTimeOut;
        timer = ba.if(bypass, 0, timerState + 1);
        output = ba.if(bypass, abs(x), outState);
    };
};

// RMS with indipendent attack and release time:
// reference:
// Udo Zlzer - Digital Audio Signal Processing Second Edition
// reference :
// https://fmipa.umri.ac.id/wp-content/uploads/2016/03/
// Udo-Zolzer-digital-audio-signal-processing.9780470997857.40435.pdf
RMS(att,rel,x) = loop ~ _ : sqrt
with {
    loop(y) = (1.0 - coeff) * x * x + coeff * y
    with {
        attCoeff = exp(-2.0 * ma.PI * (1.0/ma.SR) / att);
        relCoeff = exp(-2.0 * ma.PI * (1.0/ma.SR) / rel);
        coeff = ba.if(abs(x) > y, attCoeff, relCoeff);
    };
};

// Moving Average RMS
movingAverage(seconds, x) = x - (x @ N) : fi.pole(1.0) / N
```

```
with {
N = seconds * ma.SR;
};
RMSRectangular(seconds, x) = sqrt(max(0, movingAverage(seconds, x * x)));

// Peak Envelope (envelope follower)
// reference :
// https://www.dariosanfilippo.com/blog/2017/
// lookahead-limiting-in-pure-data/
// reference :
// https://www.cs.princeton.edu/courses/archive/spr05/cos579/DSP/DSP.html
peakenvelope(t,x) = abs(x) <: loop ~ _ * rt60(t)
with{
    loop(y,z) = ( (y,z) : max);
    rt60(t) = 0.001^((1/ma.SR)/t);
};

// peak-hold module with an exponential decay curve
peakHoldwDecay(holdSeconds, frequencyCut, decayT60, x) = x :
    peakHolder(holdSeconds) : LPTPT(frequencyCut) : peakenvelope(decayT60);

//-------------------------------------------------------- LOOKAHEAD LIMITERS ---
// reference :
// https://www.dariosanfilippo.com/blog/2017/
// lookahead-limiting-in-pure-data/
// reference :
// https://users.iem.at/zmoelnig/publications/limiter/
// Peak normalization
// reference :
//
    https://www.hackaudio.com/digital-signal-processing/amplitude/peak-normalization/

normalization(treshold, x) = treshold / ( x : peakHoldwDecay(.1, 500, 10) ) *
    x @ (( 2/1000 ) * ma.SR);
```

```
limitation(threshold, x) = ( x : peakHoldwDecay(.1, 500, 10) ) :
    ( threshold / max(ma.EPSILON, _) : min(1.0) ) *
        ( x @ (ms2samp(1)));

//-------------------------------------------------------------- NONLINEARITY ---



nonLinearity(exponent, refPeriod, y) = y : lowFreqNoise <:
    (nonlinearsig(exponent, ma.tanh(_ * normRMS)))
with{
    lowFreqNoise(x) = x : seq(i, 4, LPTPT(1.0 / refPeriod));
    noiseRMS(x) = x * x : LPTPT(1.0 / (10.0 * refPeriod)) : sqrt;
    normRMS(x) = 1.0 / max(ma.EPSILON, noiseRMS(x));
    nonlinearsig(exponent, x) = ma.signum(x) * pow(abs(x), exponent);
};
```

SpectreLists.lib

```
// import faust standard library
import("stdfaust.lib");



// INSTRUMENT SPECTRES ------------------------------------
// FFT analysis

// ------------------------------------------------- CELLO C
C_SUL_TASTO_frequencies = (
65.0, 131.0, 196.0, 261.0, 326.0, 392.0, 457.0, 522.0, 588.0, 653.0, 718.0,
    784.0, 849.0, 914.0, 979.0, 1045.0, 1110.0, 1175.0, 1241.0, 1306.0, 1371.0,
    1502.0, 1567.0, 1632.0, 1698.0, 1763.0, 1828.0, 1894.0, 1959.0, 2024.0,
    2089.0, 2351.0
 ) ;

C_SUL_TASTO_amplitudes = (
0.04196492, 0.27707616, 1.0, 0.08153769, 0.05522969, 0.01922137, 0.03436227,
    0.03067041, 0.01797695, 0.03381146, 0.0550058, 0.02913938, 0.05834666,
    0.03480883, 0.01985851, 0.02222397, 0.02735344, 0.00482074, 0.00496045,
    0.00848029, 0.01668042, 0.0121161, 0.04640077, 0.02313585, 0.00729633,
    0.00722812, 0.01041008, 0.00722225, 0.00592071, 0.01158114, 0.00443968,
    0.00282064
 ) ;

C_SUL_TASTO_bandwidths = (
1.00000000
 ) ;

C_SUL_PONTE_frequencies = (
65.0, 130.0, 195.0, 261.0, 326.0, 391.0, 456.0, 521.0, 586.0, 651.0, 717.0,
    782.0, 847.0, 912.0, 977.0, 1042.0, 1107.0, 1170.0, 1238.0, 1300.0, 1367.0,
    1434.0, 1499.0, 1564.0, 1629.0, 1694.0, 1760.0, 1825.0, 1890.0, 1954.0,
    2021.0, 2085.0
```

```
 ) ;

C_SUL_PONTE_amplitudes = (
0.02789289, 0.40853124, 1.0, 0.11091643, 0.0647856, 0.03734871, 0.02493605,
    0.02314501, 0.02637558, 0.03331669, 0.08188277, 0.04299356, 0.06739248,
    0.03783816, 0.0274411, 0.04182531, 0.0386263, 0.01525164, 0.00687666,
    0.01221123, 0.01760329, 0.00686469, 0.01441863, 0.03875951, 0.04289226,
    0.02230978, 0.03048675, 0.01094138, 0.00915901, 0.00736049, 0.00956486,
    0.00532164
 ) ;

C_SUL_PONTE_bandwidths = (
1.00000000
 ) ;

C_MOLTO_SUL_PONTE_frequencies = (
65.0, 130.0, 195.0, 261.0, 324.0, 391.0, 456.0, 522.0, 568.0, 652.0, 715.0,
    782.0, 849.0, 913.0, 978.0, 1043.0, 1108.0, 1173.0, 1304.0, 1369.0, 1434.0,
    1501.0, 1565.0, 1632.0, 1695.0, 1762.0, 1825.0, 1877.0, 1955.0, 2021.0,
    2086.0, 2136.0, 2216.0
 ) ;

C_MOLTO_SUL_PONTE_amplitudes = (
0.01789289, 1.0, 0.1119946, 0.37794605, 0.01853937, 0.09701652, 0.01508738,
    0.08147287, 0.01527196, 0.13719799, 0.04488286, 0.10778057, 0.03938983,
    0.1086438, 0.02285654, 0.18177087, 0.04472455, 0.03171519, 0.04385811,
    0.02564654, 0.04654733, 0.02098973, 0.14135472, 0.04765163, 0.05790227,
    0.04065074, 0.03339859, 0.02326387, 0.0358107, 0.02291944, 0.03277207,
    0.01508144, 0.01629959
 ) ;

C_MOLTO_SUL_PONTE_bandwidths = (
1.00000000
 ) ;
```

```
C_ARMONICI_frequencies = (
62.0, 128.0, 192.0, 253.0, 327.0, 383.0, 465.0, 575.0, 653.0, 731.0, 808.0,
    879.0, 980.0, 1051.0, 1106.0, 1253.0, 1307.0, 1406.0, 1472.0, 1568.0, 1633.0,
    1740.0, 1860.0, 1960.0, 2027.0, 2096.0, 2287.0, 2613.0, 2940.0, 3267.0,
    3593.0, 3922.0
 ) ;

C_ARMONICI_amplitudes = (
0.02379753, 0.01588414, 0.0630001, 0.01690107, 1.0, 0.01977984, 0.0071704,
    0.02076793, 0.76311832, 0.01959406, 0.00693621, 0.01205804, 0.43136463,
    0.00940068, 0.01014689, 0.00820981, 0.18989729, 0.01012334, 0.00511506,
    0.02526374, 0.55446251, 0.00962092, 0.00790504, 0.06238205, 0.0096802,
    0.00466146, 0.04442261, 0.01876171, 0.01171935, 0.00492013, 0.00646246,
    0.00434414
 ) ;

C_ARMONICI_bandwidths = (
1.00000000
 ) ;



// ------------------------------------------------- CELLO G
G_SUL_TASTO_frequencies = (
98.0, 195.0, 293.0, 391.0, 489.0, 537.0, 587.0, 685.0, 783.0, 880.0, 979.0,
    1076.0, 1174.0, 1272.0, 1371.0, 1468.0, 1566.0, 1664.0, 1762.0, 1860.0,
    1958.0, 2056.0, 2153.0, 2252.0, 2349.0, 2643.0, 2741.0, 2839.0, 2937.0,
    3230.0, 3328.0, 4013.0
 ) ;

G_SUL_TASTO_amplitudes = (
0.75217775, 1.0, 0.09843679, 0.02426847, 0.06051622, 0.00787687, 0.04838698,
    0.05785498, 0.0640251, 0.05363475, 0.02121633, 0.03638926, 0.01772408,
    0.02102163, 0.00274825, 0.00886369, 0.02193515, 0.01863539, 0.02143791,
    0.01670722, 0.01592137, 0.00709897, 0.01305707, 0.00348521, 0.00190274,
    0.00161424, 0.00333966, 0.00320751, 0.00126865, 0.00132724, 0.00152525,
```

```
    0.00097757
 ) ;

G_SUL_TASTO_bandwidths = (
1.00000000
 ) ;

G_SUL_PONTE_frequencies = (
98.0, 196.0, 294.0, 371.0, 491.0, 537.0, 588.0, 685.0, 785.0, 880.0, 981.0,
    1079.0, 1176.0, 1275.0, 1368.0, 1416.0, 1470.0, 1569.0, 1662.0, 1765.0,
    1863.0, 1959.0, 2025.0, 2069.0, 2135.0, 2249.0, 2353.0, 2452.0, 2549.0,
    2640.0, 2746.0, 2844.0
 ) ;

G_SUL_PONTE_amplitudes = (
0.93944667, 1.0, 0.1397667, 0.04960039, 0.07790099, 0.02185911, 0.0619144,
    0.09455926, 0.11242908, 0.10650139, 0.03686391, 0.08468119, 0.04027777,
    0.04988492, 0.00878743, 0.00832572, 0.02584302, 0.03396986, 0.03909495,
    0.02370068, 0.01383755, 0.00943089, 0.00821213, 0.01066961, 0.00879838,
    0.00467986, 0.00689834, 0.00597029, 0.00579739, 0.00418297, 0.00827453,
    0.00705785
 ) ;

G_SUL_PONTE_bandwidths = (
1.00000000
 ) ;

G_MOLTO_SUL_PONTE_frequencies = (
97.0, 190.0, 294.0, 370.0, 490.0, 545.0, 588.0, 686.0, 785.0, 882.0, 982.0,
    1079.0, 1178.0, 1275.0, 1373.0, 1471.0, 1568.0, 1671.0, 1766.0, 1863.0,
    1961.0, 2059.0, 2135.0, 2255.0, 2353.0, 2549.0, 2651.0, 2747.0, 2843.0,
    3235.0, 3334.0, 6664.0
 ) ;

G_MOLTO_SUL_PONTE_amplitudes = (
```

```
0.41314195, 0.33304011, 0.09897786, 0.09786711, 0.30920571, 0.03242771,
    0.24827038, 1.0, 0.24879086, 0.19577422, 0.11964251, 0.29149713, 0.07896685,
    0.2476789, 0.0553103, 0.06057399, 0.13858297, 0.01611911, 0.04639251,
    0.07277674, 0.18327809, 0.01780529, 0.00905065, 0.02117437, 0.0095725,
    0.04107566, 0.01187122, 0.02229674, 0.0160313, 0.01228535, 0.0104387,
    0.0054387
 ) ;

G_MOLTO_SUL_PONTE_bandwidths = (
1.00000000
 ) ;

G_ARMONICI_frequencies = (
50.0, 102.0, 191.0, 391.0, 495.0, 538.0, 656.0, 783.0, 889.0, 961.0, 1080.0,
    1175.0, 1290.0, 1408.0, 1510.0, 1569.0, 1676.0, 1760.0, 1853.0, 1962.0,
    2068.0, 2126.0, 2227.0, 2295.0, 2354.0, 2493.0, 2602.0, 2745.0, 2855.0,
    3137.0, 3922.0, 7999.0
 ) ;

G_ARMONICI_amplitudes = (
0.29967682, 0.52334332, 0.559064, 1.0, 0.3884109, 0.15656609, 0.14363522,
    0.84879013, 0.17350533, 0.1347252, 0.08320332, 0.68126899, 0.09573852,
    0.04995808, 0.0790263, 0.23214791, 0.07300799, 0.02958386, 0.03261651,
    0.27091008, 0.05083893, 0.06691769, 0.03115996, 0.01623343, 0.04412083,
    0.0099207, 0.0097774, 0.05262677, 0.01634304, 0.0090678, 0.00720456,
    0.00639569
 ) ;

G_ARMONICI_bandwidths = (
1.00000000
 ) ;



// ------------------------------------------------- CELLO D
D_SUL_TASTO_frequencies = (
```

```
53.0, 147.0, 203.0, 294.0, 367.0, 441.0, 534.0, 588.0, 735.0, 785.0, 882.0,
    961.0, 1028.0, 1078.0, 1175.0, 1269.0, 1323.0, 1470.0, 1617.0, 1764.0,
    1855.0, 1910.0, 2054.0, 2204.0, 2273.0, 2351.0, 2498.0, 2645.0, 2792.0,
    3233.0, 3380.0, 3527.0
 ) ;

D_SUL_TASTO_amplitudes = (
0.07116083, 1.0, 0.0405712, 0.49828005, 0.02470587, 0.25042786, 0.0250452,
    0.09954785, 0.54357606, 0.01573797, 0.2722688, 0.01228042, 0.14442907,
    0.01626654, 0.28175268, 0.02115358, 0.07170723, 0.06191843, 0.07354127,
    0.05126037, 0.01093969, 0.0343155, 0.06985491, 0.09585942, 0.00656447,
    0.01233555, 0.01134687, 0.01055565, 0.03127152, 0.01587526, 0.00962998,
    0.0068168
 ) ;

D_SUL_TASTO_bandwidths = (
1.00000000
 ) ;

D_SUL_PONTE_frequencies = (
47.0, 147.0, 295.0, 376.0, 442.0, 589.0, 737.0, 884.0, 1032.0, 1179.0, 1326.0,
    1474.0, 1621.0, 1768.0, 1916.0, 2063.0, 2211.0, 2358.0, 2506.0, 2653.0,
    2800.0, 2947.0, 3095.0, 3242.0, 3390.0, 3537.0, 3832.0, 4126.0, 4716.0,
    8695.0, 8842.0, 8990.0
 ) ;

D_SUL_PONTE_amplitudes = (
0.0115988, 0.79435758, 1.0, 0.00761094, 0.31147711, 0.27645337, 0.70041476,
    0.37004342, 0.22757333, 0.26280688, 0.0140129, 0.26240869, 0.22235847,
    0.09641164, 0.06606946, 0.17807532, 0.08549717, 0.0395032, 0.00685839,
    0.03022879, 0.03570775, 0.00762574, 0.00934675, 0.00905542, 0.01589669,
    0.00390371, 0.01159427, 0.00429153, 0.00508935, 0.00393937, 0.00423055,
    0.00489773
 ) ;
```

```
D_SUL_PONTE_bandwidths = (
1.00000000
 ) ;

D_MOLTO_SUL_PONTE_frequencies = (
50.0, 120.0, 193.0, 308.0, 367.0, 453.0, 537.0, 589.0, 736.0, 828.0, 880.0,
    964.0, 1031.0, 1101.0, 1179.0, 1275.0, 1330.0, 1408.0, 1480.0, 1630.0,
    1727.0, 1782.0, 1933.0, 2085.0, 2233.0, 2402.0, 2567.0, 2658.0, 2725.0,
    3385.0, 3564.0, 3716.0
 ) ;

D_MOLTO_SUL_PONTE_amplitudes = (
0.12016983, 0.20445953, 0.36800338, 0.15692499, 0.30131985, 0.14972588,
    0.18533416, 0.33731543, 0.72214925, 0.05300139, 0.42079979, 0.12813918,
    0.30809514, 0.09815371, 0.57583112, 0.04054637, 0.23351324, 0.0829156,
    0.97109771, 1.0, 0.05439881, 0.61768199, 0.46537996, 0.65365303, 0.2683161,
    0.195627, 0.09141798, 0.06337808, 0.21253215, 0.04839137, 0.07402531,
    0.14828266
 ) ;

D_MOLTO_SUL_PONTE_bandwidths = (
1.00000000
 ) ;

D_ARMONICI_frequencies = (
50.0, 120.0, 189.0, 301.0, 363.0, 588.0, 711.0, 775.0, 876.0, 937.0, 1009.0,
    1096.0, 1177.0, 1308.0, 1414.0, 1506.0, 1572.0, 1684.0, 1768.0, 1870.0,
    1929.0, 2068.0, 2162.0, 2277.0, 2349.0, 2458.0, 2580.0, 2780.0, 2940.0,
    3199.0, 3330.0, 4116.0
 ) ;

D_ARMONICI_amplitudes = (
0.08798947, 0.06955063, 0.38684497, 0.12097697, 0.0666715, 1.0, 0.05412838,
    0.07071769, 0.0406236, 0.0585358, 0.05028837, 0.17782794, 0.58121312,
    0.01898218, 0.04289775, 0.0657497, 0.07530507, 0.04046669, 0.07199584,
```

```
    0.01676172, 0.02490302, 0.02432096, 0.02480079, 0.02332054, 0.05601889,
    0.01049962, 0.00898637, 0.03078073, 0.02698139, 0.00997946, 0.01003087,
    0.01249989
 ) ;

D_ARMONICI_bandwidths = (
1.00000000
 ) ;



// ------------------------------------------------- CELLO A
A_SUL_TASTO_frequencies = (
50.0, 130.0, 220.0, 309.0, 368.0, 439.0, 538.0, 659.0, 879.0, 1099.0, 1319.0,
    1539.0, 1759.0, 1978.0, 2198.0, 2418.0, 2638.0, 2858.0, 3078.0, 3298.0,
    3517.0, 3737.0, 3957.0, 4177.0, 4617.0, 5056.0, 5496.0, 5717.0, 7035.0,
    7255.0, 8574.0, 9672.0
 ) ;

A_SUL_TASTO_amplitudes = (
0.01835863, 0.01216278, 1.0, 0.00749257, 0.01035828, 0.6674602, 0.00707754,
    0.62411215, 0.4053105, 0.64427134, 0.27821024, 0.08982961, 0.07338235,
    0.03442368, 0.22375855, 0.02409027, 0.04759307, 0.03888326, 0.01222045,
    0.03283333, 0.02061391, 0.02601317, 0.01631748, 0.01776269, 0.00854136,
    0.01528356, 0.01369543, 0.00841917, 0.02237106, 0.00969712, 0.00986048,
    0.01255551
 ) ;

A_SUL_TASTO_bandwidths = (
1.00000000
 ) ;

A_SUL_PONTE_frequencies = (
50.0, 130.0, 220.0, 309.0, 370.0, 440.0, 571.0, 660.0, 880.0, 962.0, 1099.0,
    1319.0, 1472.0, 1539.0, 1610.0, 1760.0, 1979.0, 2199.0, 2289.0, 2419.0,
    2639.0, 2859.0, 3079.0, 3298.0, 3518.0, 3738.0, 3958.0, 4178.0, 4398.0,
```

```
    4618.0, 5937.0, 6157.0, 7037.0, 7696.0
 ) ;

A_SUL_PONTE_amplitudes = (
0.01835863, 0.01595528, 1.0, 0.00749257, 0.01473354, 0.87218321, 0.00859515,
    0.58950176, 0.38136923, 0.0123053, 0.58324971, 0.37238753, 0.00848152,
    0.18288659, 0.01062739, 0.0110874, 0.05359905, 0.31640501, 0.01081347,
    0.02695659, 0.06988175, 0.0487292, 0.02884024, 0.05809226, 0.03553339,
    0.04916007, 0.02046966, 0.02435397, 0.00808617, 0.0085891, 0.00905874,
    0.01194446, 0.02025366, 0.00864089
 ) ;

A_SUL_PONTE_bandwidths = (
1.00000000
 ) ;

A_MOLTO_SUL_PONTE_frequencies = (
50.0, 97.0, 191.0, 309.0, 365.0, 440.0, 541.0, 658.0, 764.0, 878.0, 960.0,
    1010.0, 1098.0, 1236.0, 1316.0, 1431.0, 1471.0, 1534.0, 1759.0, 1977.0,
    2199.0, 2417.0, 2639.0, 2857.0, 3079.0, 3297.0, 3517.0, 3736.0, 4179.0,
    5718.0, 7260.0, 7699.0, 8358.0
 ) ;

A_MOLTO_SUL_PONTE_amplitudes = (
0.09118761, 0.12160486, 0.37748318, 0.00749257, 0.13470854, 0.22464179,
    0.07338048, 0.53291657, 0.03839275, 0.35695539, 0.05320811, 0.05098461, 1.0,
    0.0278835, 0.1524945, 0.02130052, 0.03015198, 0.0465949, 0.08738714,
    0.15106893, 0.56470169, 0.14170238, 0.1250138, 0.08395657, 0.0551964,
    0.06252154, 0.11631001, 0.03080954, 0.02444248, 0.02256155, 0.02680212,
    0.04103985, 0.02178977
 ) ;

A_MOLTO_SUL_PONTE_bandwidths = (
1.00000000
 ) ;
```

```
A_ARMONICI_frequencies = (
62.0, 130.0, 191.0, 261.0, 311.0, 370.0, 457.0, 537.0, 705.0, 878.0, 972.0,
    1117.0, 1169.0, 1247.0, 1472.0, 1697.0, 1755.0, 1983.0, 2050.0, 2287.0,
    2381.0, 2546.0, 2632.0, 2813.0, 3185.0, 3247.0, 3517.0, 4395.0, 6150.0,
    7892.0, 8787.0, 9665.0
 ) ;

A_ARMONICI_amplitudes = (
0.0518624, 0.01107689, 0.04390014, 0.01010253, 0.0061377, 0.02792264, 0.00731042,
    0.00808291, 0.0146077, 1.0, 0.0069374, 0.00814141, 0.00537457, 0.00460423,
    0.0111435, 0.02712111, 0.178429, 0.00467003, 0.0076334, 0.00658081,
    0.00390276, 0.00570306, 0.04632687, 0.00751121, 0.00556629, 0.00511733,
    0.01623212, 0.01861312, 0.01242462, 0.0044044, 0.00456787, 0.00629657
 ) ;

A_ARMONICI_bandwidths = (
1.00000000
 ) ;



// INSTRUMENT SPECTRES ------------------------------------
// index of the lists

FreqsListCH01(index) = ba.take(index, C_SUL_TASTO_frequencies ) ;
AmplsListCH01(index) = ba.take(index, C_SUL_TASTO_amplitudes ) ;
BandsListCH01(index) = ba.take(1, C_SUL_TASTO_bandwidths ) ;

FreqsListCH02(index) = ba.take(index, C_SUL_PONTE_frequencies ) ;
AmplsListCH02(index) = ba.take(index, C_SUL_PONTE_amplitudes ) ;
BandsListCH02(index) = ba.take(1, C_SUL_PONTE_bandwidths ) ;

FreqsListCH03(index) = ba.take(index, C_MOLTO_SUL_PONTE_frequencies ) ;
AmplsListCH03(index) = ba.take(index, C_MOLTO_SUL_PONTE_amplitudes ) ;
BandsListCH03(index) = ba.take(1, C_MOLTO_SUL_PONTE_bandwidths ) ;
```

```
FreqsListCH04(index) = ba.take(index, C_ARMONICI_frequencies ) ;
AmplsListCH04(index) = ba.take(index, C_ARMONICI_amplitudes ) ;
BandsListCH04(index) = ba.take(1, C_ARMONICI_bandwidths ) ;

FreqsListCH05(index) = ba.take(index, G_SUL_TASTO_frequencies ) ;
AmplsListCH05(index) = ba.take(index, G_SUL_TASTO_amplitudes ) ;
BandsListCH05(index) = ba.take(1, G_SUL_TASTO_bandwidths ) ;

FreqsListCH06(index) = ba.take(index, G_SUL_PONTE_frequencies ) ;
AmplsListCH06(index) = ba.take(index, G_SUL_PONTE_amplitudes ) ;
BandsListCH06(index) = ba.take(1, G_SUL_PONTE_bandwidths ) ;

FreqsListCH07(index) = ba.take(index, G_MOLTO_SUL_PONTE_frequencies ) ;
AmplsListCH07(index) = ba.take(index, G_MOLTO_SUL_PONTE_amplitudes ) ;
BandsListCH07(index) = ba.take(1, G_MOLTO_SUL_PONTE_bandwidths ) ;

FreqsListCH08(index) = ba.take(index, G_ARMONICI_frequencies ) ;
AmplsListCH08(index) = ba.take(index, G_ARMONICI_amplitudes ) ;
BandsListCH08(index) = ba.take(1, G_ARMONICI_bandwidths ) ;

FreqsListCH09(index) = ba.take(index, D_SUL_TASTO_frequencies ) ;
AmplsListCH09(index) = ba.take(index, D_SUL_TASTO_amplitudes ) ;
BandsListCH09(index) = ba.take(1, D_SUL_TASTO_bandwidths ) ;

FreqsListCH10(index) = ba.take(index, D_SUL_PONTE_frequencies ) ;
AmplsListCH10(index) = ba.take(index, D_SUL_PONTE_amplitudes ) ;
BandsListCH10(index) = ba.take(1, D_SUL_PONTE_bandwidths ) ;

FreqsListCH11(index) = ba.take(index, D_MOLTO_SUL_PONTE_frequencies ) ;
AmplsListCH11(index) = ba.take(index, D_MOLTO_SUL_PONTE_amplitudes ) ;
BandsListCH11(index) = ba.take(1, D_MOLTO_SUL_PONTE_bandwidths ) ;

FreqsListCH12(index) = ba.take(index, D_ARMONICI_frequencies ) ;
AmplsListCH12(index) = ba.take(index, D_ARMONICI_amplitudes ) ;
```

```
BandsListCH12(index) = ba.take(1, D_ARMONICI_bandwidths ) ;

FreqsListCH13(index) = ba.take(index, A_SUL_TASTO_frequencies ) ;
AmplsListCH13(index) = ba.take(index, A_SUL_TASTO_amplitudes ) ;
BandsListCH13(index) = ba.take(1, A_SUL_TASTO_bandwidths ) ;

FreqsListCH14(index) = ba.take(index, A_SUL_PONTE_frequencies ) ;
AmplsListCH14(index) = ba.take(index, A_SUL_PONTE_amplitudes ) ;
BandsListCH14(index) = ba.take(1, A_SUL_PONTE_bandwidths ) ;

FreqsListCH15(index) = ba.take(index, A_MOLTO_SUL_PONTE_frequencies ) ;
AmplsListCH15(index) = ba.take(index, A_MOLTO_SUL_PONTE_amplitudes ) ;
BandsListCH15(index) = ba.take(1, A_MOLTO_SUL_PONTE_bandwidths ) ;

FreqsListCH16(index) = ba.take(index, A_ARMONICI_frequencies ) ;
AmplsListCH16(index) = ba.take(index, A_ARMONICI_amplitudes ) ;
BandsListCH16(index) = ba.take(1, A_ARMONICI_bandwidths ) ;

// lists interpolations
FrequenciesListSum(index, Sel) =
                        (FreqsListCH01(index) * (Sel == 1)) +
                        (FreqsListCH02(index) * (Sel == 2)) +
                        (FreqsListCH03(index) * (Sel == 3)) +
                        (FreqsListCH04(index) * (Sel == 4)) ;
AmplitudesListSum(index, Sel) =
                        (AmplsListCH01(index) * (Sel == 1)) +
                        (AmplsListCH02(index) * (Sel == 2)) +
                        (AmplsListCH03(index) * (Sel == 3)) +
                        (AmplsListCH04(index) * (Sel == 4)) ;
BandwidthsListSum(index, Sel) =
                        (BandsListCH01(index) * (Sel == 1)) +
                        (BandsListCH02(index) * (Sel == 2)) +
                        (BandsListCH03(index) * (Sel == 3)) +
                        (BandsListCH04(index) * (Sel == 4)) ;
```

```
// ChngList = nentry("Change List", 1, 1, 4, 1);
// process = par(i, 10, FrequenciesListG((i + 1), ChngList), AmplitudesListG((i +
    1), ChngList), BandwidthsListG((i + 1), ChngList));

// linear interpolation
linInterpolate(x0, x1, delta) = x0 + delta * (x1-x0);
siglinInterpol(order, x) = x : seq(r, order, interpolate)
with{
    interpolate(y) = y + .5 * (y' - y);
};

// bilinear interpolation
bilinInterpolate(x0, x1, x0b, x1b, dt1, dt2) =
    linInterpolate(
        linInterpolate(x0, x1, dt1),
        linInterpolate(x0b, x1b, dt1),
        dt2)
    with{
        linInterpolate(x0, x1, delta) = x0 + delta * (x1-x0);
    };

// lists interpolations
FrequenciesListinterpolate(index, Sel1, Sel2, Sel3, Sel4, dt1, dt2) =
    bilinInterpolate(
        FreqsListCH01(index),
        FreqsListCH02(index),
        FreqsListCH03(index),
        FreqsListCH04(index),
        dt1, dt2);
AmplitudesListinterpolate(index, Sel1, Sel2, Sel3, Sel4, dt1, dt2) =
    bilinInterpolate(
        AmplsListCH01(index),
        AmplsListCH02(index),
        AmplsListCH03(index),
```

```
        AmplsListCH04(index),
        dt1, dt2);
BandwidthsListinterpolate(index, Sel1, Sel2, Sel3, Sel4, dt1, dt2) =
    bilinInterpolate(
        BandsListCH01(index),
        BandsListCH02(index),
        BandsListCH03(index),
        BandsListCH04(index),
        dt1, dt2);
```

B Second appendix

In questo appendice sono riportati i codici utilizzati per le analisi spettrali del violoncello effettuati in Python per il sistema RITI v.1.0.

`FFT_range.py` il primo prototipo di questo codice è stato sviluppato in collaborazione con il mio collega e amico Edoardo Staffa. L'analisi DFT del file audio può essere effettuata scegliendo il numero di Bins e il Range in frequenza da analizzare (il Bandwidth è ricavato sulla base di questi dati), l'output dell'analisi è sia un plot di uno spettrogramma in formato .html navigabile, che un file di testo con tutti i dati: frequenze, ampiezze e Bandwidth. `sort_amplitudes.py` prende la lista generata ed effettua un sort decrescente delle liste a partire dalle frequenze con i picchi di ampiezza più elevati a quelli meno elevati. `filter_frequencies.py` elimina le frequenze dalle liste troppo vicine ai picchi più elevati (impostato di default a 40Hz). `sort_frequencies.py` effettua il sort per frequenze in ordine crescente. `makelib.py` infine converte le liste .txt processate in un file .lib leggibile da Faust.

FFT_range.py

```
# import libraries
import matplotlib.pyplot as plt
from scipy.io import wavfile as wav
from scipy.fftpack import fft
import numpy as np
import argparse
import os
from bokeh.io import output_file
from bokeh.plotting import figure, output_file, save



# Parse the command line arguments
parser = argparse.ArgumentParser()
parser.add_argument("audiofilein", help="Name of the audio file with the
    extension")
parser.add_argument("min_frequency", type=int, help="Minimum frequency in the
    range")
parser.add_argument("max_frequency", type=int, help="Maximum frequency in the
    range")
parser.add_argument("num_bins", type=int, help="Number of Bins")
args = parser.parse_args()

# Read the audio file
rate, data = wav.read(args.audiofilein)
print(f"Sample Rate of the audio file: {rate}")

# calculate the number of bin for your Hz range
resolutionbin = ((20000 / (args.max_frequency - args.min_frequency)) *
    args.num_bins)
print(f"Number of Bins necessary in the full spectrum for your bin N in your
    range: {resolutionbin}")

# number of bins used for the fft
```

```
numberofbins = int( (resolutionbin / 20000) * rate)

# Perform the FFT with numberofbins bins
fft = np.fft.fft(data, n=(numberofbins))

# Get the frequencies and amplitudes
frequencies = np.abs(fft)

# Get the frequencies corresponding to each element in the FFT result
frequencies_axis = np.fft.fftfreq(len(frequencies), d=1/rate)

# Calculate the bin bandwidth
bin_bandwidth = rate / (numberofbins)

# Select only the elements corresponding to the desired frequency range
selected_indices = (frequencies_axis >= args.min_frequency) & (frequencies_axis
    <= args.max_frequency)
selected_amplitudes_axis = frequencies[selected_indices]
selected_frequencies_axis = frequencies_axis[selected_indices]

# Normalize the selected amplitudes
normalized_amplitudes = selected_amplitudes_axis /
    np.max(selected_amplitudes_axis)

# Scale the normalized amplitudes to the desired range (0 to 1 in this example)
scaled_amplitudes = normalized_amplitudes * 1

# Classic plot
# plt.plot(selected_frequencies_axis, scaled_amplitudes)
# plt.show()

# Create a Bokeh plot object
p = figure(title="FFT Plot", x_axis_label='Frequencies (Hz)',
    y_axis_label='Amplitudes (Linear)')
p.line(selected_frequencies_axis, scaled_amplitudes)
```

```
# Export the plot as an HTML file
output_file(args.audiofilein+"_DFT.html")
save(p)

# Use os.path.splitext to split the extension from the root
newfilename = (args.audiofilein)
root, ext = os.path.splitext(newfilename)

# Save the selected frequencies
with open((root)+".txt", "w") as f:
    f.write((root)+"_frequencies = \n")
    for i in range(len(selected_frequencies_axis)):
        if i == len(selected_frequencies_axis) - 1:
            f.write(format(selected_frequencies_axis[i], '.8f') + " \n")
        else:
            f.write(format(selected_frequencies_axis[i], '.8f') + ", ")

# Save the scaled amplitudes
    f.write((root)+"_amplitudes = \n")
    for i in range(len(scaled_amplitudes)):
        if i == len(scaled_amplitudes) - 1:
            f.write(format(scaled_amplitudes[i], '.8f') + " \n")
        else:
            f.write(format(scaled_amplitudes[i], '.8f') + ", ")

# Save the bin bandwidth
    f.write((root)+"_bandwidths = " + format(bin_bandwidth, '.8f') + " \n")
    f.close()



# print the generated .txt file
print("Contents of your data stored in the file: ")
f = open((root)+".txt", 'r')
file_contents = f.read()
print(file_contents)
```

```
f.close()
```

sort_amplitudes.py

```
# import libraries
import argparse
import os



parser = argparse.ArgumentParser()
parser.add_argument("listsfile", help="Name of the lists file with the extension")
args = parser.parse_args()

listsfile_no_ext = os.path.splitext(args.listsfile)[0]

with open(args.listsfile) as f:
    frequencies = []
    amplitudes = []
    bandwidths = []
    for line in f:
        if listsfile_no_ext+"_frequencies" in line:
            frequencies = [float(x) for x in f.readline().split(", ")]
        elif listsfile_no_ext+"_amplitudes" in line:
            amplitudes = [float(x) for x in f.readline().split(", ")]
        elif listsfile_no_ext+"_bandwidths" in line:
            bandwidths = float(line.split("=")[1])

# sort the amplitudes in descending order
sorted_amplitudes = sorted(amplitudes, reverse=True)

# update the frequencies and bandwidths accordingly
sorted_frequencies = [frequencies[amplitudes.index(amp)] for amp in
    sorted_amplitudes]
sorted_bandwidths = bandwidths

# Use os.path.splitext to split the extension from the root
newfilename = (args.listsfile)
```

```
root, ext = os.path.splitext(newfilename)


# Save the selected frequencies
with open((root)+".txt", "w") as f:
    f.write((root)+"_frequencies = \n")
    for i in range(len(sorted_frequencies)):
        if i == len(sorted_frequencies) - 1:
            f.write(format(sorted_frequencies[i], '.8f') + " \n")
        else:
            f.write(format(sorted_frequencies[i], '.8f') + ", ")


# Save the scaled amplitudes
    f.write((root)+"_amplitudes = \n")
    for i in range(len(sorted_amplitudes)):
        if i == len(sorted_amplitudes) - 1:
            f.write(format(sorted_amplitudes[i], '.8f') + " \n")
        else:
            f.write(format(sorted_amplitudes[i], '.8f') + ", ")


# Save the bin bandwidth
    f.write((listsfile_no_ext) + "_bandwidths = " + format(bandwidths, '.8f') +
        "\n")
    f.close()



# print the generated .txt file
print("Contents of your data stored in the file: ")
f = open((root)+".txt", 'r')
file_contents = f.read()
print(file_contents)
f.close()
```

filter_frequencies.py

```
# import libraries
import argparse
import os
import numpy as np



parser = argparse.ArgumentParser()
parser.add_argument("listsfile", help="Name of the lists file with the extension")
args = parser.parse_args()

listsfile_no_ext = os.path.splitext(args.listsfile)[0]

with open(args.listsfile) as f:
    frequencies = []
    amplitudes = []
    bandwidths = []
    for line in f:
        if listsfile_no_ext+"_frequencies" in line:
            frequencies = [float(x) for x in f.readline().split(", ")]
        elif listsfile_no_ext+"_amplitudes" in line:
            amplitudes = [float(x) for x in f.readline().split(", ")]
        elif listsfile_no_ext+"_bandwidths" in line:
            bandwidths = float(line.split("=")[1])



# create a mask to remove frequencies within 40Hz of each other
to_delete = []

for i in range(len(frequencies)):
    for j in range(i+1, len(frequencies)):
        if abs(frequencies[i] - frequencies[j]) < 40:
            to_delete.append(j)
```

```
mask = np.ones(len(frequencies), dtype=bool)
mask[to_delete] = False

# use the mask to filter the frequencies and amplitudes lists
filtered_frequencies = np.array(frequencies)[mask]
filtered_amplitudes = np.array(amplitudes)[mask]

# sort the amplitudes in descending order
sorted_amplitudes = sorted(filtered_amplitudes, reverse=True)

# update the frequencies and bandwidths accordingly
sorted_frequencies = [filtered_frequencies[np.where(filtered_amplitudes ==
    amp)[0][0]] for amp in sorted_amplitudes]

# Use os.path.splitext to split the extension from the root
newfilename = (args.listsfile)
root, ext = os.path.splitext(newfilename)

# Save the selected frequencies
with open((listsfile_no_ext)+".txt", "w") as f:
    f.write((listsfile_no_ext)+"_frequencies = \n")
    for i in range(len(sorted_frequencies)):
        if i == len(sorted_frequencies) - 1:
            f.write(str(sorted_frequencies[i]) + " \n")
        else:
            f.write(str(sorted_frequencies[i]) + ", ")

    # Save the scaled amplitudes
    f.write((listsfile_no_ext)+"_amplitudes = \n")
    for i in range(len(sorted_amplitudes)):
        if i == len(sorted_amplitudes) - 1:
            f.write(str(sorted_amplitudes[i]) + " \n")
        else:
            f.write(str(sorted_amplitudes[i]) + ", ")
```

```
# Save the bin bandwidth
    f.write((listsfile_no_ext) + "_bandwidths = " + format(bandwidths, '.8f') +
        "\n")
    f.close()


print("you have in the list", len(filtered_frequencies), "frequencies")



# print the generated .txt file
print("Contents of your data stored in the file: ")
f = open((listsfile_no_ext)+".txt", 'r')
file_contents = f.read()
print(file_contents)
f.close()
```

sort_frequencies.py

```
# import libraries
import argparse
import os

parser = argparse.ArgumentParser()
parser.add_argument("listsfile", help="Name of the lists file with the extension")
args = parser.parse_args()

listsfile_no_ext = os.path.splitext(args.listsfile)[0]

with open(args.listsfile) as f:
    frequencies = []
    amplitudes = []
    bandwidths = []
    for line in f:
        if listsfile_no_ext+"_frequencies" in line:
            frequencies = [float(x) for x in f.readline().split(", ")]
        elif listsfile_no_ext+"_amplitudes" in line:
            amplitudes = [float(x) for x in f.readline().split(", ")]
        elif listsfile_no_ext+"_bandwidths" in line:
            bandwidths = float(line.split("=")[1])

def sort_frequencies(frequencies, amplitudes, bandwidths):
    zipped = zip(frequencies, amplitudes)
    sorted_zipped = sorted(zipped, key=lambda x: x[0])
    sorted_frequencies, sorted_amplitudes = zip(*sorted_zipped)
    return list(sorted_frequencies), list(sorted_amplitudes), bandwidths
result = sort_frequencies(frequencies, amplitudes, bandwidths)
sorted_frequencies, sorted_amplitudes, sorted_bandwidths = result

# Save the selected frequencies
with open((listsfile_no_ext)+".txt", "w") as f:
    f.write((listsfile_no_ext)+"_frequencies = \n")
```

```
    for i in range(len(sorted_frequencies)):
        if i == len(sorted_frequencies) - 1:
            f.write(str(sorted_frequencies[i]) + " \n")
        else:
            f.write(str(sorted_frequencies[i]) + ", ")

# Save the scaled amplitudes
    f.write((listsfile_no_ext)+"_amplitudes = \n")
    for i in range(len(sorted_amplitudes)):
        if i == len(sorted_amplitudes) - 1:
            f.write(str(sorted_amplitudes[i]) + " \n")
        else:
            f.write(str(sorted_amplitudes[i]) + ", ")

# Save the bin bandwidth
    f.write((listsfile_no_ext) + "_bandwidths = " + format(bandwidths, '.8f') +
        "\n")
    f.close()



# print the generated .txt file
print("Contents of your data stored in the file: ")
f = open((listsfile_no_ext)+".txt", 'r')
file_contents = f.read()
print(file_contents)
f.close()
```

makelib.py

```
# import libraries
import argparse
import os

parser = argparse.ArgumentParser()
parser.add_argument("listsfile", help="Name of the lists file with the extension")
args = parser.parse_args()

listsfile_no_ext = os.path.splitext(args.listsfile)[0]

with open(args.listsfile) as f:
    frequencies = []
    amplitudes = []
    bandwidths = []
    for line in f:
        if listsfile_no_ext+"_frequencies" in line:
            frequencies = [float(x) for x in f.readline().split(", ")]
        elif listsfile_no_ext+"_amplitudes" in line:
            amplitudes = [float(x) for x in f.readline().split(", ")]
        elif listsfile_no_ext+"_bandwidths" in line:
            bandwidths = float(line.split("=")[1])

# Save the selected frequencies
with open((listsfile_no_ext)+".lib", "w") as f:
    f.write((listsfile_no_ext)+"_frequencies = (\n")
    for i in range(len(frequencies)):
        if i == len(frequencies) - 1:
            f.write(str(frequencies[i]) + "\n ) ; \n \n")
        else:
            f.write(str(frequencies[i]) + ", ")

# Save the scaled amplitudes
    f.write((listsfile_no_ext)+"_amplitudes = (\n")
```

```
    for i in range(len(amplitudes)):
        if i == len(amplitudes) - 1:
            f.write(str(amplitudes[i]) + "\n ) ; \n \n")
        else:
            f.write(str(amplitudes[i]) + ", ")


# Save the bin bandwidth
    f.write((listsfile_no_ext) + "_bandwidths = (\n" + format(bandwidths, '.8f') +
        "\n ) ; \n")
    f.close()
```

References

Alberto Giustiniano, Luca Fabbris e. *CFP18 cibernetica, sistemi, teorie, modelli*. URL: https://philosophykitchen.com/2022/03/cfp18-cibernetica-sistemi-teorie-modelli/. (accessed: 03.11.2022).

Barrow-Green, June. *L'Ottocento: astronomia. Il problema dei tre corpi e la stabilità del Sistema solare*. URL: https://www.treccani.it/enciclopedia/l-ottocento-astronomia-il-problema-dei-tre-corpi-e-la-stabilita-del-sistema-solare_%28Storia-della-Scienza%29/#:~:text=La%20formulazione%20del%20problema%20dei,il%20moto%20negli%20istanti%20successivi.. (accessed: 03.11.2022).

Black, H.S. "Stabilized Feed-Back Amplifiers". In: *ALEE. committee on communication* (1933).

Bonnet, François J. *After Death*. en. Urbanomic, 2020, p. 80.

Collins, Nicolas. "'Pea Soup' – A History". In: (September, 2011), pp. 1–23. URL: http://www.nicolascollins.com/texts/peasouphistoryOLD.pdf.

Di Scipio, Agostino. "'Sound is the interface': from *interactive* to *ecosystemic* signal processing". en. In: *Organised Sound* 8.3 (Dec. 2003), pp. 269–277. ISSN: 1355-7718, 1469-8153. DOI: 10.1017/S1355771803000244. URL: https://www.cambridge.org/core/product/identifier/S1355771803000244/type/journal_article (visited on 11/22/2022).

— *A Relational Ontology of Feedback*. en. Jan. 2022. DOI: 10.47041/TKUL7223. URL: https://echo.orpheusinstituut.be/article/a-relational-ontology-of-feedback (visited on 11/22/2022).

— *Circuiti del Tempo Un percorso storico-critico nella creatività musicale elettroacustica e informatica*. it. N. 38. Lucca: Libreria Musicale Italiana, 2021. ISBN: 9788855430685.

— "Composition by exploration of non-linear dynamic systems". In: *ICMC Glasgow 1990 Proceedings*. C.S.C. University of Padova, 1990, pp. 324–327.

Di Scipio, Agostino. “Polveri Sonore - Una prospettiva ecosistemica della composizione”. it. In: *La Camera Verde* (2014). Publisher: La Camera Verde, 17–42 Pages. (Visited on 11/22/2022).

Dunbar-Hester, Christina. “Listening to Cybernetics: Music, Machines, and Nervous Systems, 1950-1980”. en. In: *Science Technology e Human Values* 35.1 (2010), pp. 113–139. URL: http://www.jstor.org/stable/27786196.

Gallehr, Theo. *Gruppo Nuova Consonanza Azioni. Documentario.* URL: https://www.youtube.com/watch?v=dqvAhBJ99wA. (accessed: 06.02.2023).

Liang, J. and W. Song. “DIFFERENCE EQUATION OF LORENZ SYSTEM”. en. In: *International Journal of Pure and Apllied Mathematics* 83.1 (Feb. 2013). ISSN: 1311-8080, 1314-3395. DOI: 10.12732/ijpam.v83i1.9. URL: http://www.ijpam.eu/contents/2013-83-1/9/ (visited on 11/22/2022).

Lorenz, Edward. “Deterministic Nonperiodic Flow”. In: *Journal of Atmospheric Sciences* 20.2 (1963), pp. 130–148.

Maxwell, Mr. J. C. “on Governors”. In: (1868).

Mudd, Tom. “Between chaotic synthesis and physical modelling: Intrumentalizing with Gutter Synthesis”. en. In: *Proceedings of the Seventh Conference on Computation, Communication, Aesthetics X* (), p. 12.

Nitschke, Mathis. *Analog Neural Synthesis.* URL: https://mlure.art/analog-neural-synthesis/. (accessed: 04.11.2022).

Patteson, Thomas W. “The Time of Roland Kayn’s Cybernetic Music”. In: (), pp. 1–11. URL: https://kayn.nl/wp-content/uploads/2016/12/The-Time-of-Roland-Kayns-Cybernetic-Music-Thomas-Patteson.pdf.

Pirkle, Will. “Virtual Analog (VA) Filter Implementation and Comparisons v2.0”. en. In: *www.willpirkle.com* (), p. 24.

Ranzato, Sofia. “L’origine del mondo divino Una lettura di Esiodo”. ita. Thesis. Facoltà di Lettere e Filosofia C. d. L. S. in Scienze dell’Antichità, 2006/2007, p. 10.

Sanfilippo, Dario. "CONSTRAINED DIFFERENTIAL EQUATIONS AS COMPLEX SOUND GENERATORS". en. In: (2021), p. 8.

— "Time-Domain Adaptive Algorithms for Low- and High-Level Audio Information Processing". en. In: *Computer Music Journal* 45.1 (Mar. 2021), pp. 24–38. ISSN: 0148-9267, 1531-5169. DOI: 10.1162/comj_a_00592. URL: https://direct.mit.edu/comj/article/45/1/24/110389/Time-Domain-Adaptive-Algorithms-for-Low-and-High (visited on 11/22/2022).

— "Time-variant infrastructures and dynamical adaptivity for higher degrees of complexity in autonomous music feedback systems: the Order from noise (2017) project". en. In: *Musica/Tecnologia* (Aug. 2018). Artwork Size: 119-129 Pages Publisher: Musica/Tecnologia, 119–129 Pages. DOI: 10.13128/MUSIC_TEC-23804. URL: https://oajournals.fupress.net/index.php/mt/article/view/7482 (visited on 11/22/2022).

Sanfilippo, Dario and Andrea Valle. "Feedback Systems: An Analytical Framework". In: *Computer Music Journal* 37.2 (2013), pp. 12–27. DOI: doi:10.1162/COMJa00176. URL: https://direct.mit.edu/comj/article-abstract/37/2/12/94420/Feedback-Systems-An-Analytical-Framework?redirectedFrom=PDF.

Scott, Bernard. "Second-order cybernetics: an historical introduction". In: *Kybernetes* 33.9.10 (2003), pp. 1365–1378. DOI: doi:10.1108/03684920410556007. URL: https://sites.ufpe.br/moinhojuridico/wp-content/uploads/sites/49/2021/10/Ciber-2b-22-out.-second-order-cybernetcs.pdf.

Technology, Massachusetts Institute of. *Alvin Lucier on "I am sitting in a room"*. URL: https://www.youtube.com/watch?v=v9XJWBZBzq4. (accessed: 04.11.2022).

Von Foerster, Heinz. *Understanding Understanding: Essays on Cybernetics and Cognition*. en. Vol. 2. 1. 2005. DOI: 10.29173/cmplct8737. URL: https://journals.library.ualberta.ca/complicity/index.php/complicity/article/view/8737 (visited on 11/22/2022).

Wiener, Norbert. *The Human Use of Human Beings*. en. 1950, p. 224.

Wishart, Trevor. *On Sonic Art*. en. Edinburgh: harwood academic publishers, 1996, p. 355.

Zavalishin, Vadim. *THE ART OF VA FILTER DESIGN*. en. rev. 2.1.0 (October 28, 2018), p. 520.