



Università degli Studi di Ferrara
Facoltà di Ingegneria

Corso di Laurea in Ingegneria Elettronica e delle
Telecomunicazioni

**REALIZZAZIONE DI UN SOFTWARE PER
L'ACQUISIZIONE E L'EDITING AUDIO
E SVILUPPO DI EFFETTI ED ALGORITMI
DI SOUND PROCESSING**

Tesi di Laurea di:
Cristian Zambelli

Relatore:
Prof. Davide Bertozzi

Correlatore:
Ing. Simone Medardoni

Anno Accademico 2004/2005

... non penso che l'apparecchiatura prenderà il sopravvento. Certo non potremmo fare ciò che facciamo senza l'apparecchiatura. Ma potremmo comunque fare un ottimo spettacolo musicale senza di essa. Sta tutto nel modo in cui controlli l'apparecchiatura, non devi lasciare che sia lei a controllarti. Si tratta solo di usare gli strumenti disponibili, quando sono disponibili. Ci sono un sacco di strumenti elettronici che possiamo utilizzare. Sta a noi decidere se utilizzarli o meno. Sono estensioni del prodotto delle nostre menti. Devi avere in testa qualcosa per poterlo esprimere. L'apparecchiatura non è in grado di pensare, non può controllarsi. Sarebbe interessante vedere cosa possono fare con l'apparecchiatura, quattro persone che non se ne intendono...

David Gilmour & Roger Waters – Pink Floyd

Sommario

Capitolo 1 – Introduzione

1.1 “The Story so far...”	6
1.2 Effetti. Cosa sono e come funzionano.	7
1.3 Come vengono costruiti questi oggetti.	11
1.4 Propositi della tesi.	15
1.5 Cosa c’è di nuovo in tutto questo.	19

Capitolo 2 – Progettazione del sistema di Low Frequency Oscillation

2.1 Parliamo di LFO.	21
2.2 Back to the sixties... i primi LFO.	22
2.3 La nuova generazione di LFO.	25
2.3.1 LFO digitali pseudo – randomici con trigger a isteresi.	25
2.3.2 LFO digitali Regular Stair Stepped.	33
2.4 Sviluppo dell’ LFO – Maker con Matlab.	39

Capitolo 3 – Progetto del sistema di acquisizione del segnale audio e metodi per l’analisi

3.1 Prefazione del sistema.	57
3.2 Metodi di Acquisizione del segnale audio arbitrario.	58
3.2.1 By – File Acquisition (Wave Method).	58
3.2.2 By – File Acquisition (MP3 Method).	64
3.2.3 Real – Time Acquisition.	70
3.2.4 Speed Benchmark.	72
3.3 Metodi di elaborazione sul segnale audio acquisito.	74
3.3.1 Calcolo della PSD (Power Spectral Density).	74
3.3.2 Calcolo dello spettrogramma.	80
3.4 Sviluppo del Signal Acquisition System con Matlab.	82

Capitolo 4 – Sviluppo degli algoritmi di modulazione diretta del segnale

4.1 Prefazione del sistema.....	113
4.2 La modulazione di ampiezza sui segnali audio con LFW.....	114
4.2.1 Il Tremolo: cos'è e come fa	114
4.2.2 Ring Modulator.....	116
4.2.3 Speed Enhancement.....	117
4.3 LFOCoder.....	121
4.3.1 VoCoding e idea di fondo.....	121
4.3.2 Realizzazione del LFOCoder.....	123
4.3.3 Analisi dei metodi di filtraggio usati dagli effetti di modulazione diretta.....	123
4.4 Sviluppo del Direct Modulation Tools System con Matlab.....	125

Capitolo 5 – Algoritmi di Modulazione e SmartPlayer Technique

5.1 Prefazione del sistema.....	159
5.2 Chorus.....	160
5.2.1 Come funziona un effetto Chorus.....	160
5.2.2 Parametri di un Chorus.....	163
5.2.3 Speed Test.....	165
5.3 Flanger.....	166
5.3.1 Come funziona un effetto Flanger.....	166
5.3.2 Parametri di un Flanger.....	169
5.3.3 Speed Test.....	171
5.4 La Tecnica SmartPlayer e lo Human Brain Behaviour.....	172
5.4.1 Approccio e sviluppo dell'idea.....	172
5.4.2 Parametri del sistema di SmartPlayer.....	173
5.4.3 Implementazione in Matlab del sistema SmartPlayer.....	177
5.5 Sviluppo degli algoritmi di modulazione con Matlab.....	178

Capitolo 6 – Conclusioni del progetto

6.1 Conclusioni principali.....	260
6.2 Confronto con software di alta fascia Syntrillium Cool Edit Pro.....	261

6.2.1 Features supportate.....	261
6.2.2 Speed Test caricamento dei file PCM.....	265
6.2.3 Speed Test decodifica file Motion Picture Expert Group (MPEG-1) Layer 3.....	266
6.2.4 Speed Test computazione degli effetti.....	267
6.3 User – Friendliness del sistema.....	267
6.4 Tempi di applicazione di un processing completo.....	269

Appendice A – We need Distortion!

A.1 C'è distorsione e distorsione.....	272
A.2 Metodi di realizzazione degli effetti di distorsione “analog based”.....	275
A.3 Simulazione con Matlab.....	276

Appendice B – Expanding e la tecnica di Noise Reduction

B.1 Come funziona il sistema.....	284
B.2 Perché usare espansione e noise gating.....	286
B.3 Simulazione con Matlab.....	287

Appendice C – Equalizzazione di un segnale audio

C.1 Come funziona un equalizzatore.....	293
C.1.1 Controlli di Tono.....	293
C.1.2 Equalizzatori Grafici.....	295
C.1.3 Equalizzatori Parametrici.....	297
C.2 Sfere di applicazione.....	298
C.2.1 Presenza.....	298
C.2.2 Speaker Cross – Over.....	298
C.3 Simulazione con Matlab.....	299
Bibliografia.....	308
Ringraziamenti.....	309

Capitolo 1

Introduzione

1.1 “The Story so far...”

...the borders of sound has finally smashed up. With this amazing unit you can reproduce the typical sound of a vibrato execution or recreating the subtle variation of a creamie sound that's triggered by the pick and feed your amp...

Con queste parole, nel 1971 la Roland – Boss Corp. dava il via per la prima volta, alla creazione industriale in serie di un effetto di modifica del suono: il Chorus CE-1. In quegli anni, dove il suono di una chitarra o di un basso elettrico erano ancora legati al semplice concetto di un suono “pulito” in uscita da un amplificatore a valvole o al massimo aggiustato da un semplice Wah¹, questa scoperta risultò agli occhi dei musicisti di tutto il mondo, un semplice lancio pubblicitario di un prodotto destinato ai pochi sperimentatori del suono virtuoso o altamente futurista, non solo per il prezzo iniziale di vendita, ma anche per le doti tecniche necessariamente richieste per lavorare con oggetti di questa tipologia. D'altra parte, si poneva per la prima volta in 20 anni (la data a cui si fa risalire più o meno la costruzione del primo amplificatore per chitarra elettrica), una normale persona che avesse come interesse il semplice piacere di suonare uno strumento musicale, di fronte a un circuito elettronico pieno zeppo di transistori bipolari a giunzione, amplificatori operazionali, linee di ritardo analogiche, ecc. Oggi tutto questo ci fa sorridere essendo ormai abituati al continuo evolvere dei sistemi integrati digitali, in cui in unico chip di silicio, grande più o meno come una moneta da 50 centesimi di euro (ma dal costo ben superiore...), è possibile svolgere milioni di operazioni al secondo e salvarle in memoria per poi rielaborarle ancora. Negli anni '70 però, la rottura di questa barriera, imposta oltre che dalle scarse conoscenze elettroniche in uso, anche dalla cosiddetta “Società dei musicisti”, che guardava con un occhio dispregiativo chi cercava di porsi alla pari con le scoperte tecnologiche in auge, rappresentò un enorme salto di qualità nella composizione della musica e nello sviluppo delle tecnologie elettroniche applicate alla musica e in particolare si fa riferimento per la prima volta allo studio della teoria dei segnali e dei loro metodi di elaborazione. Ai giorni nostri anni però, si sta assistendo ad una progressiva scomparsa del mondo analogico e di tutto ciò che lo circonda (tranne per alcune applicazioni fondamentali come l'amplificazione ad esempio), per un dominio incontrastato della realtà digitale. Ebbene, il fatto che

¹ Per Wah si intende un effetto di modifica del contenuto spettrale informativo di un segnale audio, costituito da un filtro passa – basso a frequenza variabile a seconda della posizione del pedale di azionamento e a Q molto elevato. Il concetto verrà ritrattato successivamente.

per trattare dei segnali audio puramente analogici e costituiti da una banda di frequenza relativamente ridotta, si debbano utilizzare degli oggetti minuscoli che considerano qualsiasi cosa come uno “0” o un “1” e arrivano a lavorare a volte anche a frequenze di qualche miliardo di Hertz, non risulta essere per il 60% del mercato musicale e dei suoi utenti una buona cosa. C’è infatti una diatriba cominciata circa 10 anni fa (e che continua tutt’ora), riguardante questo argomento, ed è nata quando una piccola azienda americana chiamata Alesis decise di introdurre per la prima volta sul mercato mondiale, un modulatore di segnali audio basato interamente su DSP². In questa situazione però, non avvenne la stessa cosa del 1971, anche perché si era già a conoscenza di che cosa doveva fare quel particolare oggetto. Accadde invece una profonda spaccatura sulle opinioni delle persone affiliate alla musica, che per la prima volta non riguardavano il suono, ma come questo suono veniva trattato in maniera elettronica. I musicisti si avvicinano agli ingegneri.

1.2 Effetti. Cosa sono e come funzionano.

Si è parlato però fino ad ora di “effetti”, senza chiarirne il concetto principale. Cos’è un “effetto”? Un effetto musicale è un oggetto costituito da dispositivi elettronici al suo interno, che preso in ingresso, un segnale audio nella sua banda base dai 20 Hz sino a circa 20 KHz, esegue una trasformazione del segnale stesso tramite opportuni procedimenti di modifica, per poi ridare in uscita ancora una volta un segnale audio nella sua banda base. In sostanza quindi, si può vedere un effetto, come un sistema costituito da una “Black Box” che esegue delle operazioni al suo interno su un segnale, lasciandone inalterata la sua caratteristica frequenziale fondamentale.

² È l’acronimo di Digital Signal Processor, un chip elettronico che svolge più o meno le stesse funzionalità di un microprocessore, utilizzando però una struttura interna che gli permette di essere particolarmente adatto per svolgere operazioni matematiche sui segnali digitali come moltiplicazioni, convoluzioni, trasformazioni funzionali, ecc.

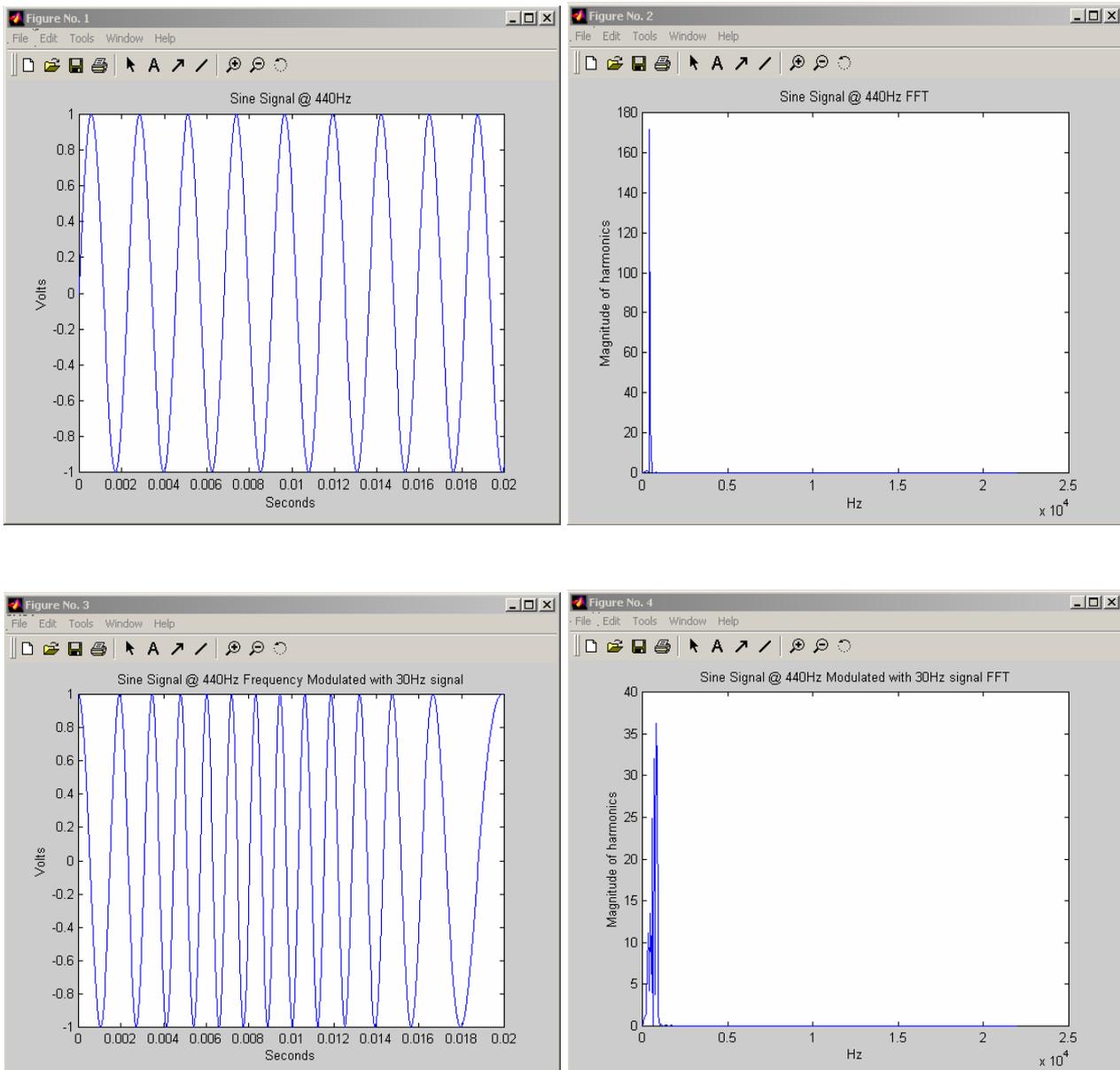


Figura 1: Immagini della modulazione diretta in FM di un segnale audio (codice disponibile su `../Codice/Matlab/FM.m` nel CD allegato alla tesi)

In questo esempio il segnale di ingresso (sinusoide) ha una frequenza fondamentale di 440 Hz (corrispondente alla nota musicale LA del pentagramma), la “Black Box” lo trasforma e in uscita abbiamo un segnale con un diverso spettro rispetto al segnale originario, ma che conserva ancora la frequenza fondamentale di 440 Hz. Si tratta della semplice modulazione di frequenza che è già nota dalla teoria sulle comunicazioni elettriche. Ovviamente questo era un semplice esempio per far capire cosa avviene più o meno all’interno di un effetto musicale, anche se le operazioni che possono essere svolte sono molte di più. A questo proposito infatti, esistono delle categorie

principali e delle sottocategorie di effetti musicali con cui possiamo classificare questi oggetti.

Partendo dalle macrocategorie, possiamo definire quindi, le seguenti tipologie di effetti:

- Effetti di modifica dell'involuppo del segnale o di modifica del contenuto spettrale: si tratta di particolari oggetti che, preso in ingresso un segnale audio, ne vanno a rivelare il suo involuppo e tramite alcune operazioni di filtraggio lo modificano
- Effetti di distorsione non lineare: si tratta di oggetti che fanno uso delle non linearità del sistema di modifica del segnale audio, per generare delle armoniche a frequenza superiore dello spettro originario
- Effetti di spazializzazione e di auralizzazione: si tratta di oggetti che utilizzano una tecnologia interna che gli permette di campionare il segnale audio e successivamente ritardarlo per generare all'orecchio umano una sensazione di lavorare in situazioni ambientali differenti (all'aperto, in una stanza, ecc.)
- Effetti di modulazione: si tratta di oggetti che, preso il segnale audio in ingresso, effettuano una biforcazione del percorso del segnale all'interno del sistema, per ottenere in uscita un segnale audio non trattato con un processo di modulazione, sommato al segnale trattato e opportunamente ritardato. Da non confondere con il processo di modulazione diretta, che corrisponde alla modulazione di ampiezza, frequenza o fase compiuta sul segnale audio in ingresso al sistema.

Dalla definizione di queste categorie, si passa ad una definizione più specifica per ogni effetto e si valutano quindi le sottocategorie citando anche i prodotti del mercato musicale, che svolgono queste funzioni:

- Effetti Wah: sono effetti che sfruttano un'operazione di filtraggio sul segnale tipicamente passa – basso a fattore Q molto elevato per enfatizzare la risonanza di una particolare frequenza del segnale audio. es. Jim Dunlop Cry Baby, Bad Horse Wah, ecc.

- Effetti Octave: sono effetti che tramite un'operazione di rettificazione o di conversione in frequenza verso il basso danno in uscita un suono di una o due ottave³ inferiore/superiore. es. Boss OC-2, MXR Blue Box, Ampeg Scrambler, ecc.
- Effetti Overdrive, Distorsione, Fuzz: sono effetti che utilizzano le non linearità del sistema introdotte dai dispositivi a semiconduttore che costituiscono il sistema (transistori, diodi, ecc.), per generare armoniche a frequenze superiori donando una caratteristica molto "sporca" al segnale di uscita, utilizzando un'operazione di clipping. I primi sono effetti di distorsione molto morbida introducendo poca non linearità, i secondi effettuano un clipping maggiore, i terzi snaturano completamente la forma d'onda del segnale audio portandolo in una forma d'onda quasi quadra. es. Boss OD-3, Boss DS-1 , Boss FZ-1, Maestro FuzzTone FS-1, Ibanez TS-9 e TS-808 Tube Screamer, ecc.
- Effetti Delay o Echo: sono effetti che campionano il segnale audio in ingresso e lo mandano in uscita ritardato di un opportuno tempo sommandolo allo stesso segnale audio eventualmente retroazionato all'ingresso del sistema (feedback). es. Boss DD-5, EHX Memory Man, ecc.
- Effetti Riverbero: sono effetti che eseguono un filtraggio massiccio sul segnale per far credere all'ascoltatore di essere all'interno di un determinato ambiente. es. Boss RV-3, Digitech DigiReverb
- Effetti Chorus: sono effetti che eseguono una biforcazione del segnale audio di ingresso in modo da avere in uscita, un segnale audio non trattato dall'effetto, sommato ad un segnale su cui è stata svolta un'operazione di ritardo modulata da un LFO⁴ e un'operazione di modulazione diretta di frequenza. es. Boss CH-5, EHX SmallClone, Digitech MultiChorus
- Effetti Flanger: sono effetti molto simili al Chorus, ma con l'aggiunta della retroazione del segnale in ingresso tramite Feedback e sfruttando l'utilizzo di un minore ritardo. es. Boss BF-3, Digitech Turbo Flange, ecc.

³ Per ottava si intende in ambito musicale, un intervallo di otto note che corrisponde ad un intervallo di frequenza valutato in potenze di 2. es. Il LA centrale è a 440Hz, mentre il LA un'ottava superiore è a 880Hz.

⁴ Acronimo di Low Frequency Oscillator. Rappresenta un oscillatore a frequenza subsonica (al di sotto della banda audio), che serve per effettuare l'operazione di modulazione del ritardo introdotto sul segnale audio in ingresso all'effetto.

- Effetti Tremolo: sono effetti che svolgono una modulazione di ampiezza diretta basata su LFO, sul segnale audio in ingresso, che corrisponde ad una variazione del livello del volume del segnale. es. Boss TR-2, Voodoo Labs Tremolo, ecc.
- Effetti Phaser: sono effetti che utilizzano delle reti di ritardo o phasing stage per avere in uscita un segnale sfasato rispetto al segnale di ingresso. es. Boss PH-3, Digitech Hyper Phase, ecc.

1.3 Come vengono costruiti questi oggetti

È stata data una definizione di effetto, che tipologie di effetto esistono sul mercato, le sottocategorie per ogni macrocategoria, ma come è fatto un effetto musicale? Questa infatti è la domanda più importante che si pone il neofita del mondo musicale, dopo aver capito qual è la base di funzionamento dello stesso. Anche qui, come per le tipologie, esistono vari metodi con cui un effetto può essere costruito:

- Effetti Floor – Mounted o Stomp boxes: sono costituiti da una scatola tipicamente di alluminio o materiale ABS di ridotte dimensioni, all'interno della quale viene posta la circuiteria elettronica dell'effetto. È dotato di due o tre prese (dette Jack 1/4") che servono da terminali di ingresso e di uscita del segnale audio da trattare (sia esso monoaurale o stereofonico). Richiedono un'alimentazione a batteria o tramite trasformatori Wall – Mart da 9 a 18 Volts in DC con un ridotto consumo di corrente. I metodi utilizzati per il processamento del segnale audio sono tipicamente analogici o al massimo viene utilizzata la tecnologia semi – digitale⁵. Questi effetti hanno la caratteristica di poter essere azionati durante l'esecuzione di un brano o comunque mentre si suona tramite un footswitch DPDT⁶ che svolge anche l'operazione di true bypass (il segnale di ingresso viene bufferizzato e instradato verso l'uscita quando l'effetto non è attivo). Essi possiedono come caratteristica elettrica, una elevata impedenza di ingresso (dell'ordine del Mohm) e una bassa impedenza di uscita (poche decine di ohm), il che li rende ideali per trattare segnali musicali che provengono direttamente da uno strumento musicale, il quale genera segnali audio arbitrari in banda base con ampiezza da 50 mV a 1,5V.

⁵ Si intende l'utilizzo di integrati come amplificatori operazionali o trigger insieme a componenti puramente passivi.

⁶ Acronimo di Double Pole Double Throw o Dual Pole Dual Throw. È un interruttore a doppio polo e doppio scambio usato per trattare linee bipolari.



Figura 2: Digitech X - Series MultiChorus

- **Effetti Rack – Mounted:** a differenza della prima tipologia, questi effetti sono costruiti per la maggior parte dei casi, con una circuiteria elettronica puramente digitale che impiega DSP ad elevate prestazioni e dispositivi di campionamento/quantizzazione del segnale audio in ingresso (ADC Sigma – Delta) e di riconversione in analogico (High – Speed DAC). Essa viene fatta risiedere all’interno di un rettangolo di larghezza pari a 19” e di altezza e profondità variabile a seconda della dimensione del dispositivo (in particolare l’altezza determina il numero di unità che l’oggetto occupa 1U = 44.1mm). Sono dotati di più prese di connessione ingresso/uscita del segnale con diverse tipologie, tra cui: 1/4” Jack, XLR – Type, S – PDIF, Toslink, ecc. Sono azionabili tramite MIDI Controller⁷ o con semplici footswitch e vengono alimentati direttamente con la tensione di rete (nella maggior parte dei casi c’è un trasformatore all’interno che porta la tensione di lavoro a 9VDC) a scapito però di richiedere molta corrente proprio per le capacità di calcolo che i dispositivi interni richiedono. Sono unità dal costo particolarmente elevato, ma che hanno una capacità di

⁷ Si tratta di una pedaliera che è composta da una serie di footswitch che a seconda della pressione o meno, inviano un messaggio MIDI (Musical Instrument Digital Interface) al sistema tramite un linguaggio chiamato SysEx (System Exclusive).

processazione superiore dei normali floor – mounted effects. Le loro caratteristiche I/O in termini di impedenza li rendono ideali a trattare qualsiasi segnale audio arbitrario in banda base che provenga o da uno strumento musicale o da un'altra sorgente esterna e questo è quasi sempre possibile grazie ad uno switch che permette di scegliere qual è il livello di segnale con cui si va a trattare.



Figura 3: Alesis Midiverb 4 Rack - Mounted

- Effetti Pedal – Board o MultiFX: sono effetti che si pongono esattamente a metà tra le due categorie citate precedentemente, perché includono al loro interno una circuiteria digitale formata da DSP e da ADC/DAC, ma includono tutte le caratteristiche meccaniche dei Floor – Mounted Effects. Spesso questi effetti sono la scelta primaria di un utente che vuole avere una grande quantità di opzioni di trattamento del segnale audio a scapito però di una minore qualità. In alcuni casi queste unità prevedono la possibilità di essere integrate con schede SmartMedia SD o Flash Card per poter essere utilizzate come sampler⁸.

⁸ Un sampler è un dispositivo elettronico che registra un segnale audio e ne permette oltre che la normale riproduzione alcune piccole operazioni di elaborazione come ad esempio il cambiamento di velocità o di tono.



Figura 4: Pedal Board ZOOM BFX - 708 II

- Effetti Computer – based o plug – ins: è l’ultima generazione in termini tecnologici di effetti musicali. Non si tratta in realtà di entità concrete, ma bensì di programmi che risiedono all’interno di un calcolatore, sfruttando l’intera capacità di elaborazione del microprocessore. Il loro funzionamento è vincolato quindi, non solo dalla quantità di memoria presente all’interno del sistema e dal microprocessore che si sta utilizzando, ma anche dalla scheda audio che esegue l’operazione di “trasferimento” del segnale audio arbitrario, dalla sorgente al microprocessore. Per trasferimento, si intende che il segnale debba essere opportunamente trattato (mediante campionamento quantizzazione e codifica del segnale), prima di essere dato in pasto al microprocessore per la successiva elaborazione. Nella maggior parte dei casi, i programmi che eseguono le operazioni di modifica del segnale audio, accettano dei segnali comunemente modulati con la tecnica PCM⁹, oppure vanno a lavorare direttamente con segnali compressi che sfruttano la tecnologia Motion Picture Expert Group – Layer 3, Real Audio, Windows Media Audio, ecc. Nel corso degli anni si sono sviluppate varie metodologie di Computer – based Effects, dipendenti dalla tipologia di schede audio utilizzate, dalle piattaforme di processazione, ecc. In questo momento lo standard che v’è per la maggiore è l’ ASIO VST¹⁰ di Steinberger, il quale tramite le chiamate di sistema basate su Windows – API, processi Host – based native,

⁹ Acronimo di Pulse Coded Modulation. È una particolare tecnica di modulazione di un segnale analogico, per renderlo un segnale numerico interpretabile da un sistema digitale. Esso sfrutta la cascata di tre blocchi fondamentali: campionamento del dato, quantizzazione e codifica secondo l’alfabeto dei simboli e delle forme d’onda del sistema.

¹⁰ Acronimo di Audio Stream Input Output Virtual Studio Technology. È uno standard proprietario sviluppato da Steinberger. Solo in questi anni cominciano a essere disponibili i primi SDK per gli sviluppatori di software.

riesce ad integrare all'interno dei software utilizzati per la registrazione e la manipolazione dei segnali audio come ad esempio Steinberger Cubase o Syntrillium Cool Edit, gli oggetti necessari (detti appunto plug – ins) a svolgere l'operazione di modifica del segnale stesso. Questi oggetti, richiedono spesso una capacità di elaborazione elevata, dal momento che devono eseguire delle operazioni matematiche non banali sullo spettro del segnale, che involgono quindi trasformazioni funzionali, decomposizioni polifase, ecc. Gli effetti di questa categoria vengono spesso utilizzati, non tanto per il gusto di effettuare la modifica del segnale audio con l'operazione prescelta, ma per eseguire quella che viene definita operazione di audio modeling. A dire la verità, più che una semplice operazione, l'audio modeling sta diventando da un paio di anni a questa parte, una vera e propria scienza, che si occupa di ricreare, con l'utilizzo di software matematici o di sviluppo, delle unità effetti non più in produzione da molti anni oppure delle valutazioni comparative (benchmark) su unità presenti sul mercato.



Figura 5: VST Plug - in per Steinberger Cubase VST

1.4 Propositi della tesi

Finalmente abbiamo dato una definizione completa relativa agli effetti musicali. Ma in sostanza di cosa si occupa questa tesi? Questa trattazione avrà il compito di sviluppare delle metodologie innovative, basate su effetti esistenti e non, riguardanti in particolare, il campo degli effetti di modulazione con alcuni cenni agli effetti di modifica dell'involgimento o del contenuto spettrale

informativo del segnale, sfruttando la tecnologia degli effetti Computer – based, arrivando alla realizzazione vera e propria di un software stand – alone¹¹ scritto con il software Matlab di MathWorks Inc. e costituito come una GUI¹². Per entrare più nel dettaglio, ci si occuperà degli effetti Chorus, Flanger – Phaser, Tremolo, enfatizzando i concetti realizzativi ed effettuando dei confronti con i prodotti presenti sul mercato musicale. Si è scelta questa particolare categoria di effetti, perché spesso, benché rappresentino l'80% dell'utilizzo del musicista medio, vengono bistrattati e considerati inutili nella maggior parte dei casi. Questo avviene in sostanza per due motivazioni: la prima è che le case costruttrici di effetti concreti (Rack, Floor – Mounted, ecc.) seguono una linea di produzione standard, che vincola i musicisti a suonare più o meno tutti allo stesso modo (il che può essere perfetto per una tribute – band, ma non per un musicista emergente ...), imponendo al mercato di bypassare le innovazioni e di spargere notizie a volte prive di verità; la seconda motivazione è che i software e i plug – ins usati nella manipolazione dei segnali audio, sono spesso troppo complessi per un utente che non mangi pane e musica tutti i giorni della settimana e nonostante questa elevata complessità, non sono comunque garantite tutte le funzionalità che invece un esperto vorrebbe avere a disposizione. Questo sembrerebbe un discorso campato in aria e completamente privo di senso. Ecco perché verranno citati alcuni esempi che dimostreranno che non è così. A tal proposito si fa riferimento ora alla pubblicità dell'unità effetti X – Series MultiChorus prodotta dalla Harman International Industries Corp. – Digitech:

Our new Multi Chorus is the first true multi-voice stompbox ever! With up to 16 voices that you can morph through by rotating the Voice knob! Now you can get chorusing so thick it flows like butter out of your speakers, in stereo or mono. It even has randomization that thickens the chorusing further as you add more voices. The Multi Chorus also has switchable cabinet emulation and stereo outputs.

So take this astounding features:

Multi-voice chorus

Blend from 1 to 16

voices

Voice randomization

¹¹ Si tratta di software che può essere eseguito direttamente come applicazione a se stante e non richiede l'integrazione in altri programmi di trattamento del segnale audio (plug – ins).

¹² Acronimo di Graphical User Interface. Si tratta di applicazioni basate su finestre grafiche, che permettono una iterazione facilitata con l'utente finale.

*Stereo and switchable
CIT™ Cabinet Modeling
outputs*

Quando nel 2002, uscì sui principali giornali del settore musicale questa pubblicità, la prima reazione fu di stupore. Per la prima volta sul mercato, è disponibile una unità stompbox, in grado di generare più voci (cioè più di una biforcazione del segnale audio di ingresso) per eseguire il Chorus. Incredibile, ma non vero. Non vero perché, chi utilizza questi oggetti tutti i giorni e in particolare chi li smonta per riparazioni o semplicemente vuole capire come sono costruiti nel nucleo, si è accorto che il circuito che esegue l'operazione di chorusing (per adesso diremo così poi spiegheremo meglio come funziona in dettaglio un chorus), in realtà non prevede la generazione di 16 voci, ma bensì di una soltanto, sfruttando però un filtro particolare che fa credere all'ignaro ascoltatore di essere in possesso di un costosissimo chorus polifonico che addirittura esegue la randomizzazione delle voci. Inoltre si parla di Cabinet Emulation Technology che l'utente può attivare/disattivare a scelta. Anche qui le informazioni sono non proprio corrette. È vero che esiste un Cabinet Simulator ed è vero che ci sono due uscite stereofoniche, ma non è vero che questa funzionalità sia switchabile, anzi, è sempre attiva ed è perfettamente identica a quella presente nella cugina unità CH-5 della rivale Roland – Boss Inc. Ecco perché si è parlato di “standardizzazione” del suono e informazioni non veritiere dei produttori. Una persona potrebbe dire però che trattandosi di un prodotto a bassa fascia (cioè riservato alla maggior parte degli utenti e anche ai non esperti), è logico che si possa incorrere in queste “trappole”. Il problema però, è che queste cose succedono anche per i prodotti “migliori” e molto più costosi. Prendiamo ad esempio queste affermazioni tratte dal manuale d'uso, dell'unità Rack – Mounted MIDIVerb prodotta dalla Alesis:

The Chorus effect is achieved by splitting the signal into three parts with a dry signal and a separate Detuning section for both left and right channels. When the left channel is detuned sharp, the right is detuned flat, and vice versa. The detuning is further effected by being modulated by an LFO (low frequency oscillator) which causes the detuning to vary. Many variables are available in this scheme: the Predelay can be varied, the LFO depth can be varied, the LFO speed can be varied, and a portion of the detuned signal can be fed back to the input to increase the effect. Finally, the waveform shape of the LFO can be changed from a smooth sinewave, to a more abrupt squarewave to make the pitch detuning more pronounced...

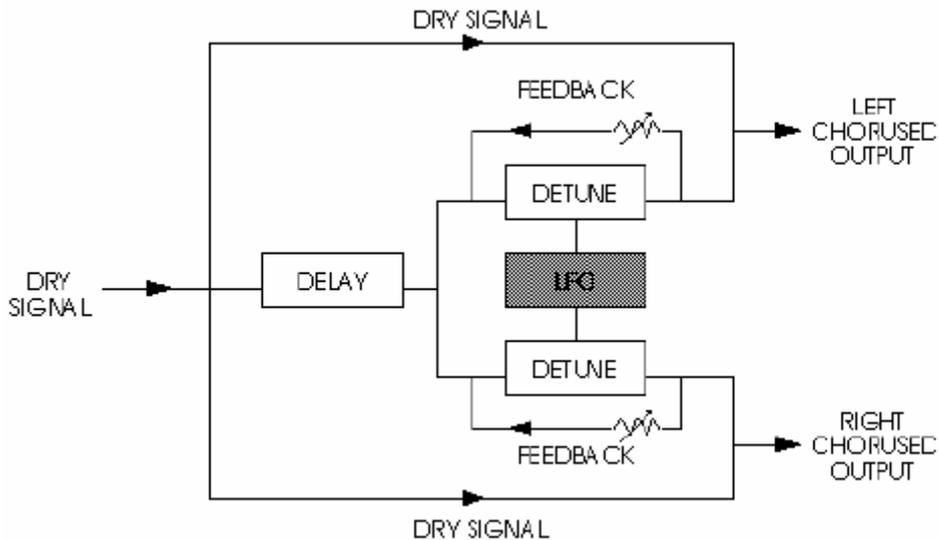
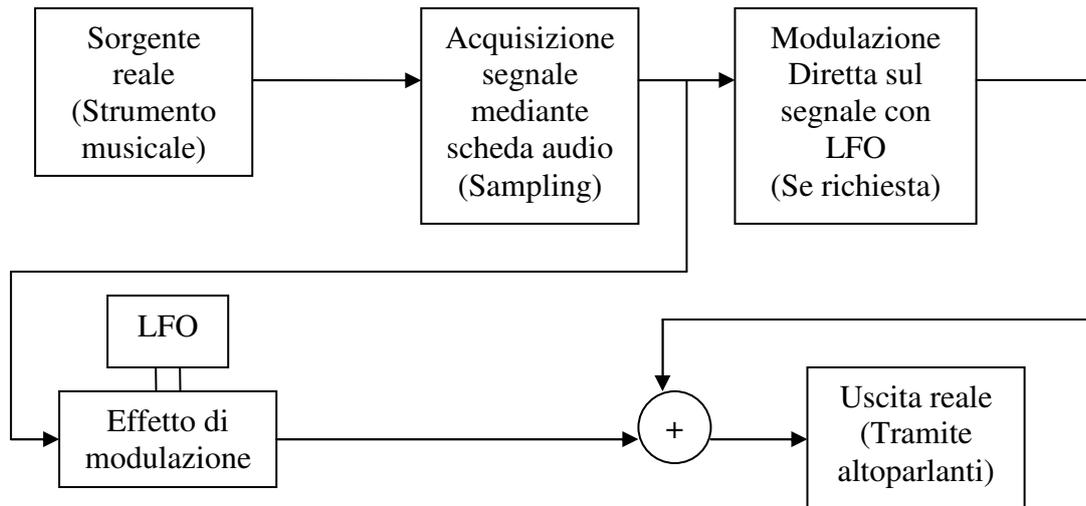


Figura 6: Schema del Chorus di un Alesis MidiVerb 4

Si tratta di un ottimo schema e di un ottimo metodo per effettuare il chorus su un segnale audio. Ci sono solo alcuni piccoli problemi. Dov'è il controllo di tono¹³? In questo schema infatti, il chorus viene applicato a tutto il segnale audio causando non pochi problemi se si pensa ad esempio ad un segnale che proviene da uno strumento musicale come un ERB elettrico¹⁴, il quale ha una gamma di frequenze che copre praticamente tutto lo spettro in banda base audio, rischiando di camuffare il suono dello strumento in un “guazzabuglio” di suoni distorti e poco pregevoli all’ascolto. E soprattutto, perché dobbiamo essere sempre per forza obbligati a scegliere tra forme d’onda standardizzate per l’LFO e non possiamo crearne una noi, secondo i nostri gusti musicali? Lo stesso discorso è valido per tutta la categoria di plug – ins creati per computer, i quali hanno dei suoni molto standardizzati e poco versatili in termini di applicabilità. Per tutte queste motivazioni ed altre che verranno spiegate durante il corso di questa tesi si cercherà di introdurre delle migliorie a queste problematiche. In sostanza quindi, quello che si vuole realizzare è un software così strutturato:

¹³ Per controllo di tono si intende un particolare filtraggio eseguito sul segnale audio, in modo da poterne controllare la timbrica finale.

¹⁴ Acronimo di Extended Range Bass. Si tratta di un basso elettrico che ha più di 4 corde e che segue un’ accordatura in intervalli di quarta partendo almeno dal SI (B). Per ulteriori informazioni si visiti il sito www.conklinguitars.com.



Quello di cui si dovrà occupare Matlab, sarà quindi, l’acquisizione del segnale audio da uno strumento musicale, la registrazione su hard – disk e l’operazione di modulazione per poi eseguire il playing del segnale risultante tramite la scheda audio stessa e renderlo disponibile all’ascolto, ovviamente con la possibilità di salvare il risultato del processo su un file.

1.5 Cosa c’è di nuovo in tutto questo?

Il titolo di questo paragrafo corrisponde perfettamente alla domanda di chi fino ad ora ha letto queste pagine introduttive. Dopo che si è capito come funziona un effetto e come è fatto dentro, cosa si propone di nuovo? Le innovazioni di questa trattazione le possiamo riassumere in questo elenco, che verrà più e più volte ripreso durante il corso dell’esposizione:

- Uso di un software matematico (Matlab) anziché di un semplice “manipolatore” su segnali audio, che permetterà al prodotto finale di avere una maggiore qualità in termini di precisione e di velocità di calcolo.
- Nuovi metodi di realizzazione dei LFO, interamente controllabili dall’utente del software (niente più onde standardizzate alla semplice senoide) sfruttando la tecnologia messa a disposizione da Matlab per il campionamento dei dati.
- Completo controllo sul segnale audio, che ammetterà ogni tipo di modulazione diretta sullo stesso, oltre al normale processo di modulazione introdotto dall’effetto desiderato.

- Reale applicazione della polifonia con la tecnica SmartPlayer¹⁵ (qui inventata e realizzata). Ciò significa che si può scegliere il numero di voci di un determinato effetto e controllare il parametro di ogni voce.
- Interfaccia GUI che permette il controllo rapido del segnale audio in ingresso mediante rappresentazione nel dominio del tempo, delle frequenze e rappresentazione spettrale CFT.
- Nuove tipologie di effetti e nuovi metodi di manipolazione del segnale audio tra cui: LFOCoder, Tube Simulator, ecc., oltre ai preset salvati in memoria per il modelling di unità effetti già esistenti.

¹⁵ La tecnologia SmartPlayer si basa sui concetti principali della composizione musicale. Per ulteriori informazioni si suggerisce di documentarsi su alcuni testi di teoria musicale.

Capitolo 2

Progettazione del sistema di Low Frequency Oscillation

2.1 Parliamo di LFO

Entriamo ora nel vivo della realizzazione della tesi, andando a valutare l'argomento che sta alla base di tutto quello che riguarda gli effetti di modulazione di un segnale audio arbitrario: gli oscillatori a frequenza subsonica, meglio conosciuti come LFO¹. La conoscenza della natura di questi oggetti è infatti, fondamentale per realizzare tutte le applicazioni principali che servono per trattare un segnale con una banda relativamente ridotta come il nostro e serviranno anche delle particolari caratteristiche che devono assolutamente essere rispettate per poter implementare questi oggetti e inserirli in un effetto, senza incorrere nelle problematiche più comuni quali: il rumore, il consumo di potenza, ecc. Ma partiamo dal principio. Perché serve un oscillatore? E soprattutto, perché la frequenza imposta da questo oscillatore deve essere al di sotto della banda audio e quindi considerata sub – sonica? La risposta alla prima domanda, può essere data, semplicemente guardando la figura 6 del capitolo 1 o il diagramma immediatamente sottostante a pag. 10. Si nota che l'elemento principale che fa eseguire l'operazione di modulazione è basato su un oscillatore, anche perché (ricordando la teoria delle comunicazioni elettriche) è necessario un segnale modulante per eseguire il processo di modulazione sia esso a portante analogica o impulsiva. Inoltre, trattandosi di oggetti (stiamo parlando degli effetti) che basano il loro funzionamento su un movimento ondulatorio tipico, che andrebbe a simulare un particolare movimento del musicista o dello strumento, è necessario come nucleo di base, un oscillatore. La risposta alla seconda domanda è invece un po' più complicata da esaudire, perché richiede la conoscenza di alcuni concetti che riguardano la teoria musicale e elettronica, anche se con alcuni accorgimenti la possiamo rendere comprensibile a tutti. Abbiamo, fin dal principio, definito cos'è un segnale audio arbitrario: si tratta di un segnale la cui grandezza elettrica che lo costituisce, è variabile nel tempo ed è caratterizzato da una banda di frequenze che va da 20 Hz a 20 KHz². Ciò significa che l'orecchio umano è in grado di percepire questi segnali nella loro interezza ed è anche in grado di percepire una variazione sulla frequenza e l'ampiezza di essi. Quanto deve essere grande questa variazione? A priori, uno potrebbe dire che non c'è nessun vincolo sulla massima deviazione relativa alle due grandezze costituenti il segnale. Non è così. Non lo è perché, se noi andiamo ad esempio ad effettuare una modulazione di ampiezza diretta sul segnale audio ci accorgiamo che più di tanto è inutile andare a

¹ In alcuni testi, questi oscillatori prendono il nome di SSFO, ovvero Sub Sonic Frequency Oscillator.

² In questo caso si considera un essere umano, con un udito perfetto.

variare la frequenza del segnale modulante, perché ci troveremmo in una situazione in cui il segnale audio di partenza diventerebbe affetto da rumore di modulazione, a causa dell'inserimento di componenti spettrali all'interno delle banda audio che si andrebbero a sovrapporre allo spettro originario del segnale causando delle modifiche sullo spettro stesso. Tale fenomeno prende il nome, nel mondo musicale, di Audio Cross – Modulation. Allo stesso modo, per quanto riguarda una modulazione di frequenza diretta sul segnale audio, non è possibile utilizzare un segnale modulante a frequenza all'interno della banda audio, in questo caso per due motivazioni: la prima è che si ha sicuramente il fenomeno dell' Audio Cross – Modulation; la seconda è che non si può avere una variazione di frequenza troppo grande sul segnale di ingresso perché in questo caso non si parlerebbe più di modulazione, ma di pitch – shifting³ e l'alterazione dei neumi⁴ sarebbe troppo elevata, causando uno stravolgimento totale del suono finale. Per tutte queste motivazioni, si è quindi scelto di utilizzare degli oscillatori la cui frequenza massima di oscillazione fosse al più di 16 – 18 Hz e la minima frequenza di oscillazione, intorno agli 0.3 – 0.6 Hz che risulta prossima alla DC, ma è in grado ancora di far percepire all'ascoltatore, che un processo di modulazione sta avvenendo. Con l'avvento degli effetti di modulazione, in cui in sostanza il tutto è basato su un ritardo variabile nel tempo, dove questo ritardo deve essere sufficientemente piccolo (dell'ordine dei ms), per poter eseguire il processo sul segnale, si richiede che questa operazione di variazione, sia eseguita da un oscillatore: un LFO appunto.

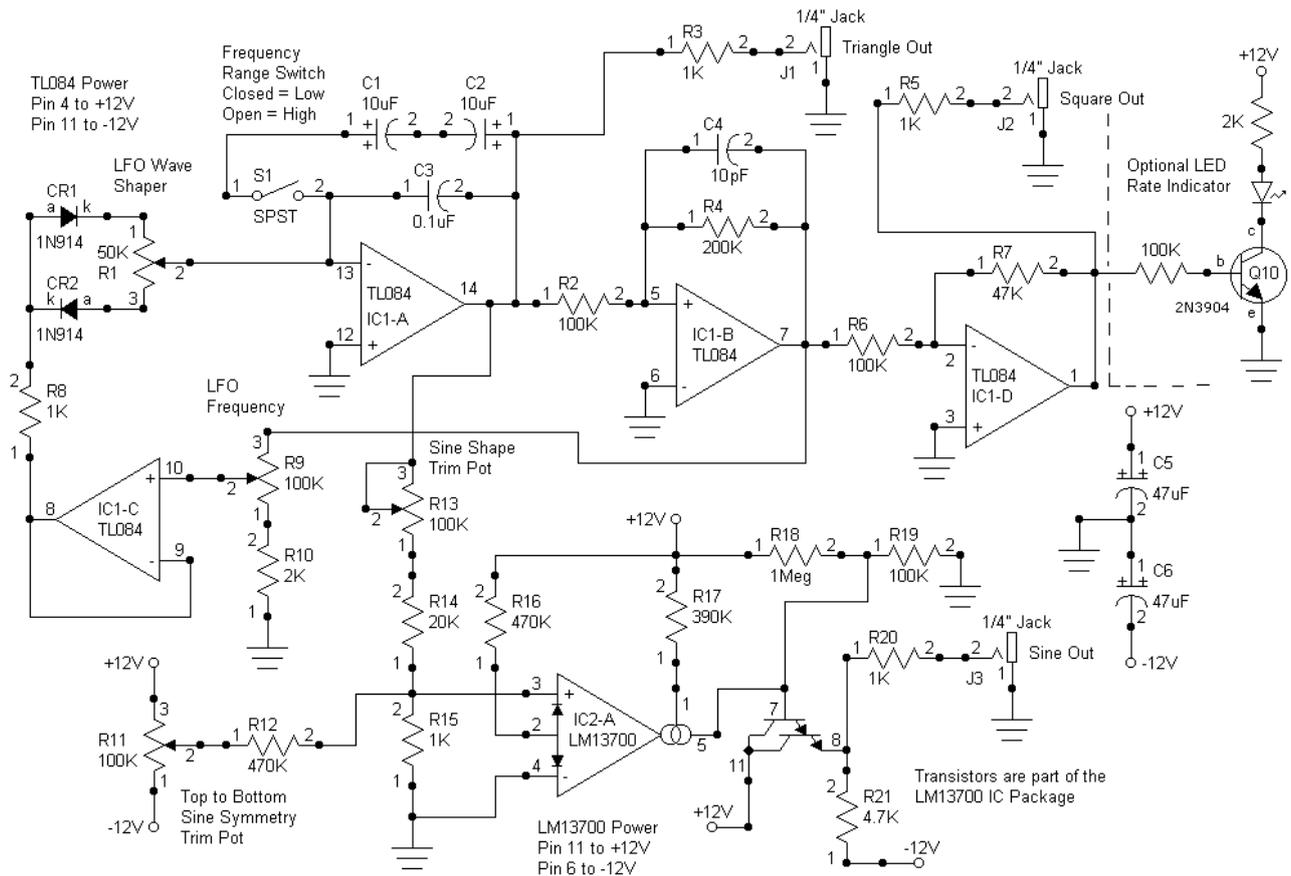
2.2 Back to the sixties... i primi LFO

I primi prototipi di LFO all'interno dei sintetizzatori degli anni '60, utilizzavano delle soluzioni circuitali puramente analogiche, che comprendevano l'uso dei tubi termoionici. Una decade dopo però, con lo sviluppo imponente della tecnologia integrata, si è potuto assistere, oltre ad una riduzione delle dimensioni di questi dispositivi, un aumento sensibile delle prestazioni dei dispositivi elettronici utilizzati per la costruzione di questi oscillatori. Un esempio principale di LFO analogico è l'oscillatore che sta all'interno dei sintetizzatori Moog e Oberheim e in questa tesi ne si presenta una reissue⁵ del 2002:

³ È un'operazione comunemente usata nell'audio processing, ogniqualvolta sia necessario traslare la frequenza di un segnale audio ad un'altra che corrisponde ad un cambiamento di tonalità.

⁴ È l'altezza di una nota sul pentagramma.

⁵ È una riedizione di un prodotto già uscito negli anni precedenti.



For unused LM13700 connect a 1Meg resistor from +12V to pins 14, 15 and 16 tied together and connect pins 10 and 13 to -12V via 1Meg.

Figura 2: Internal Oberheim LFO

Si tratta di un LFO in grado di produrre LFW sinusoidali, triangolari, quadre, rampe esponenziali e a dente di sega con frequenze che vanno da 0.1 Hz sino a circa 500 Hz (i sintetizzatori di questo tipo usavano frequenze di oscillazione a volte superiori alla banda audio, per ottenere volutamente effetti di rumore e di modulazione spuria). Il funzionamento del circuito è piuttosto semplice: la chiave di tutto è l'integrato IC1-A (amplificatore operazionale a FET TL084 a basso rumore di ingresso) in configurazione a integratore che rappresenta il cuore del LFO. Esso è seguito da un comparatore (IC1-B FET TL084) con molta isteresi nella caratteristica ingresso – uscita che effettua il sensing dell'integratore. All'accensione del circuito si supponga che il circuito IC1-B sia a livello di tensione -12V e che il potenziometro R1 (LFO Wave Shaper) sia al centro. L'uscita di IC1-B è connessa al pin 3 di R9 (potenziometro 100 KΩ per il controllo della frequenza). R9 agisce come un partitore di tensione regolabile e R10 assicura che il pin 2 di R9 non sia mai completamente a massa. IC1-C (sempre un op.amp. FET TL084) è un inseguitore a guadagno unitario e fornisce una sorgente di tensione a bassa impedenza che permette di erogare o assorbire corrente attraverso R8, CR1, CR2 e R1. Se il pin 2 di R9 fosse a massa non ci sarebbe flusso di corrente attraverso R8 e l'oscillazione del LFO si fermerebbe. Con questa connessione però, anche con il potenziometro R9

completamente verso R10 l'oscillatore è alla sua frequenza più bassa di oscillazione. Questo avviene perché la tensione che appare sul pin 8 di IC1-C determina la corrente che scorre su R8, CR2 e R1. Siccome IC1-A è un integratore, tutto quello di cui ha bisogno è un po' di corrente che scorra sul suo ingresso invertente per causare all'uscita una salita o una discesa lineare. Più corrente si utilizza, maggiore sarà la frequenza di oscillazione. Quando l'uscita di IC1-A raggiunge circa 6V (determinati dal rapporto R2 e R4), l'uscita di IC1-B va alta. La funzione di C4 è quella di regolarizzare la salita o la discesa di IC1-B, erogando o assorbendo un veloce spike di corrente all'ingresso non invertente (pin 5) dell'integrato quando l'uscita comincia ad andare alta o bassa. Una volta che l'uscita di IC1-B è alta (+12V) essa eroga 60uA attraverso R4, mentre per portare l'uscita al di sotto del livello di massa bisogna consumare un po' più di 60uA attraverso R2 all'ingresso di IC1-A. Ciò accade quando l'uscita di IC1-A cala leggermente sotto i -6V in corrispondenza della commutazione a livello alto di IC1-B. A questo punto l'uscita di IC1-B andrà basso e il ciclo ricomincerà da capo. R1 determina il tasso di corrente di carica o scarica per l'integratore IC1-A e dei suoi componenti associati per il rate della salita o discesa dell'uscita dell'integrato. La corrente in uscita dal pin 8 di IC1-C è sempre erogata su CR1 e assorbita su CR2. Se R1 è settato verso il pin 1 si ottiene una LFW a rampa, mentre se è settato verso il pin 3 si ottiene una LFW a dente di sega. Con R1 centrato si produce al pin 1 di IC1-A una LFW triangolare. L'integrato IC1-D (TL084 a FET) è usato per attenuare la LFW quadra o a impulso a $\pm 6V$. La circuiteria per lo shaping della sinusoide usa le caratteristiche di distorsione non – lineare che risultano in un overdriving dell'ingresso dell'amplificatore a transconduttanza (LM13700). R13 effettua lo shaping per determinare quanta distorsione si usa aggiustando la LFW triangolare ai capi di R15. Il transistor Darlington è costruito all'interno del chip LM13700 e serve per bufferizzare la LFW sinusoidale. C5 e C6 servono per la stabilizzazione dell'alimentazione. Questo sostanzialmente è il metodo più semplice per ottenere un LFO analogico, anche se esistono alcune varianti circuitali più complesse che permettono di ottenere un migliore controllo e una migliore generazione delle LFW, basati su un controllo in tensione dell'oscillatore. Tali circuiti prendono il nome di VCLFO (Voltage Controlled Low Frequency Oscillator) e sono una rivisitazione dei VCO (Voltage Controlled Oscillator) all'interno dei ben più noti circuiti PLL (Phase Locked Loop). Si tratta di una tipologia circuitale basata su diodi a capacità variabile⁶ e accordati su una banda di frequenze che dipende dalla tensione DC ai capi del diodo. Un esempio di VCLFO utilizzati nell'audio processing è il circuito del VCLFO Roland:

⁶ Meglio conosciuti come diodi VARICAP.

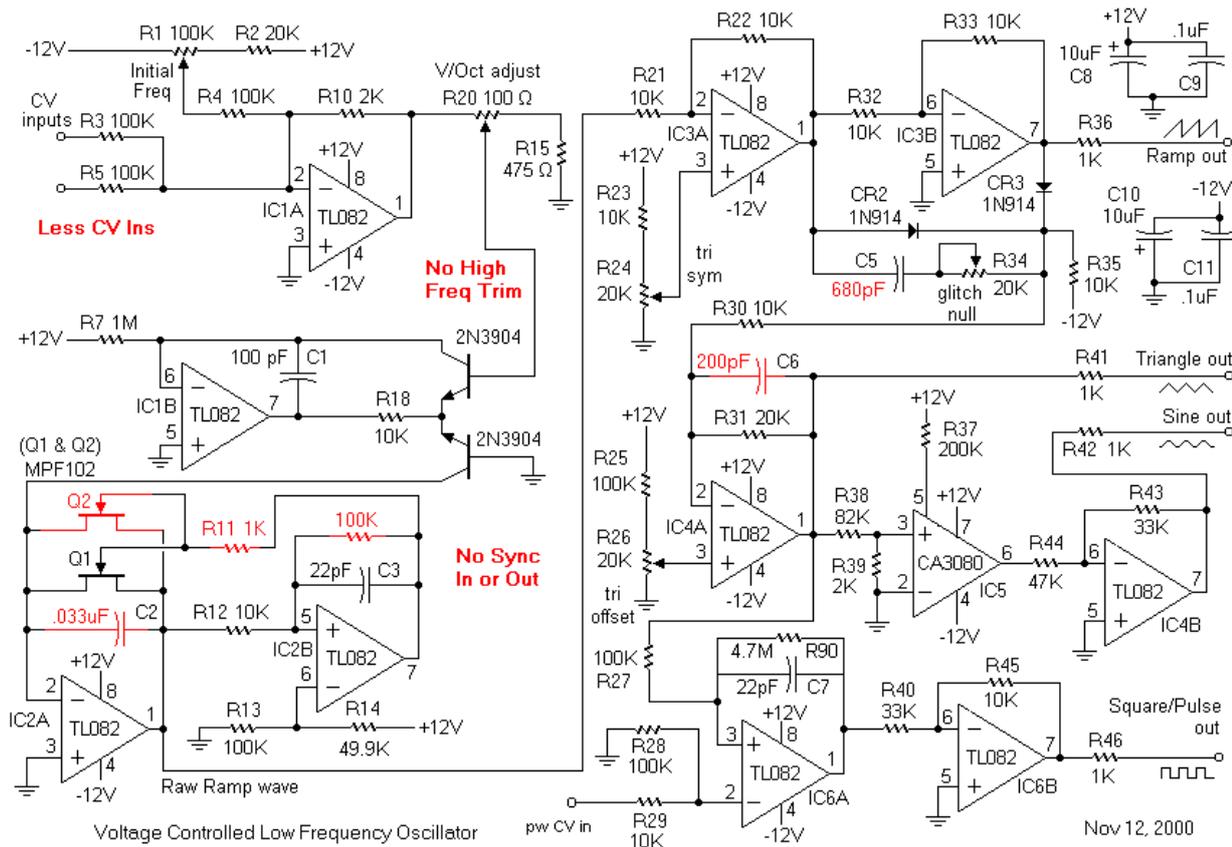


Figura 3: Roland Juno - X VCLFO

2.3 La nuova generazione di LFO

2.3.1 LFO digitali pseudo – randomici con trigger a isteresi

Ogni musicista che abbia provato almeno una volta gli effetti di modulazione come Flanger o Tremolo sa che questi oggetti sono piuttosto ripetitivi nella loro operazione di modifica delle caratteristiche audio del suono in ingresso. Infatti, proprio perché si utilizzano delle forme d'onda di modulazione periodiche (LFO sinusoidali, ecc.), a ogni periodo ricompare quella caratteristica sonorità che va a modificare sempre e costantemente il segnale audio portante su cui si vuole effettuare la modulazione. Tutta questa prevedibilità può risultare a volte un po' "stucchevole" e per aggiungere quindi imprevedibilità al processo di modulazione si può utilizzare un metodo definito pseudo – randomico⁷. Il termine "Pseudo" significa che si tratta di "qualcosa" che assomiglia alla realtà ma che non lo è, mentre "Randomico" indica che il carattere con cui questo "qualcosa" si presenta è del tutto aleatorio, cioè completamente al di fuori di ogni determinazione a priori. Quindi un segnale pseudo – randomico è una tipologia di segnale che assomiglia, ma che non è veramente

⁷ Questo concetto è alla base della tecnologia SmartPlayer e verrà riproposto più avanti, data la sua importanza.

un segnale casuale. È noto che i segnali randomici sono molto difficili da riprodurre, ma fortunatamente, quando si tratta di realizzare questa metodologia particolare di LFO, non dobbiamo fare le cose perfettamente identiche al modello valutato, ci si può accontentare a volte di alcune regolarità temporali, ma che alla lunga risulteranno indistinguibili dall'andamento complessivo della forma d'onda.

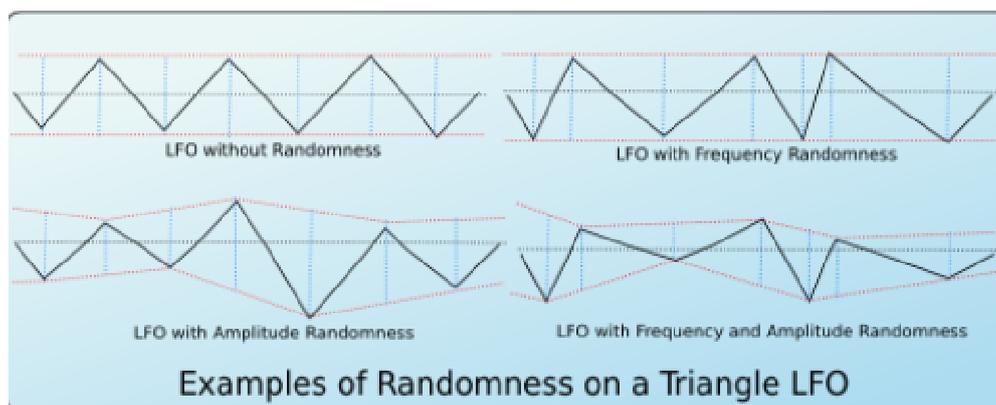


Figura 4: Tipologie di randomicità in un LFO (courtesy ZynAddSubFX)

Ci sono alcune tecniche di progettazione particolari che possono comunque produrre delle simulazioni di segnali randomici piuttosto convincenti. Una su tutte è la tecnica chiamata “Sequenza di conteggio a lunghezza massima”⁸, che permette di ottenere ottimi risultati. Tuttavia, per le applicazioni digitali, queste tecniche risultano piuttosto complicate da mettere in pratica e spesso sono anche troppo fedeli al concetto di randomicità, per poter essere utilizzate in un ambito musicale. Il più semplice tipo di LFO pseudo – randomico che possiamo realizzare è dato dalla semplice sovrapposizione di generatori di forme d’onda quadra. Un singolo oscillatore di forma d’onda quadra non è però molto casuale, si muove infatti solamente dal livello alto al livello basso periodicamente. Se però prendiamo un secondo oscillatore e lo facciamo lavorare ad una frequenza leggermente diversa dal primo, otteniamo un “beating” fra i due (effetto di battimento d’onda ben noto ai musicisti durante la fase di accordatura dello strumento). Ma il tutto è ancora troppo prevedibile. Se aggiungiamo un terzo oscillatore ecco che però le cose cominciano a complicarsi un po’. Questa tipologia di circuiti con LFO a tre oscillatori è usata a tutt’oggi nella maggior parte dei sintetizzatori PAIA per ottenere quello che viene chiamato “surf – sound”, proprio per la rassomiglianza della randomicità del segnale con le onde dell’oceano. L’aggiunta di un quarto oscillatore aumenta l’imprevedibilità a basso termine, ma alla lunga la LFW creata risulta essere

⁸ Tecnica usata nell’elaborazione numerica dei segnali, in cui si va a sfruttare la massima risoluzione di un contatore per migliorare le prestazioni del sistema.

riconoscibile. Abbiamo quindi descritto come è possibile ottenere un LFO di questo tipo in maniera piuttosto vaga, infatti non si è parlato dei dispositivi elettronici che serviranno per la costruzione di questa tipologia di circuiti. Il componente che sta alla base di questa famiglia di oscillatori è il Trigger di Schmitt (da questo il nome di LFO ... con trigger a isteresi). Un Trigger di Schmitt è un circuito digitale che utilizza la seguente notazione:

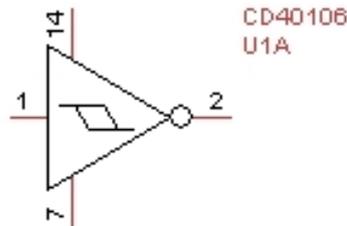


Figura 5: Simbolo del CMOS CD40106 Hex Inverter Triggered

Si tratta di un chip della famiglia CMOS (Complementary Metal Oxide Semiconductor field effect transistor), che servirà tipicamente al raddrizzamento di una forma d'onda utilizzando la seguente caratteristica ingresso – uscita:

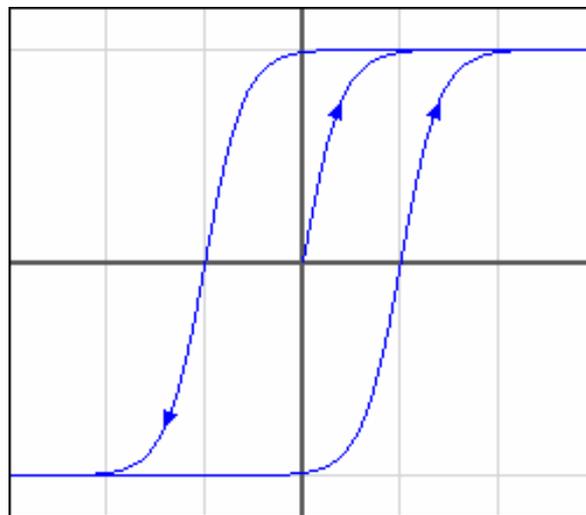


Figura 6: Isteresi tipica di un trigger

Le due caratteristiche vengono inseguite a seconda dell'andamento della tensione V_I . Se definiamo la soglia logica del circuito V_T come la tensione alla quale $V_O = V_I$, si può notare dal grafico che in realtà, a seconda dell'andamento della tensione V_I , il circuito possiede due diverse soglie logiche indicate con V_M e V_m . Se abbiamo in ingresso ad un buffer non invertente CMOS, accade che, a

patto che la soglia logica del circuito sia posizionata a $V_{DD}/2$, ogni volta che il segnale oltrepassa il valore di soglia, il circuito commuta da uno stato logico alto a uno stato logico basso e/o viceversa; per quanto riguarda invece il Trigger di Schmitt la commutazione non avviene al valore di soglia $V_{DD}/2$ ma al raggiungimento dei valori V_M o V_m . La caratteristica ingresso – uscita che abbiamo quindi definito in precedenza viene anche detta a isteresi, per cui il circuito completo prenderà il nome di Trigger di Schmitt a isteresi e può essere invertente (se l'uscita è in opposizione di fase con il segnale di ingresso dopo la commutazione) o non invertente. La realizzazione circuitale basata su tecnologia MOS complementare con l'utilizzo di transistori unipolari MOS a canale n e p è la seguente:

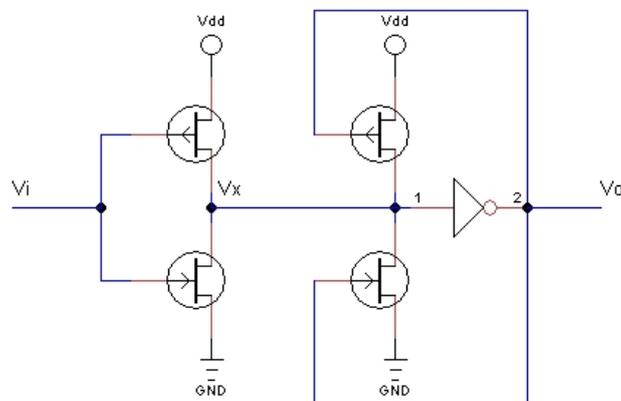


Figura 7: Schema circuitale di un trigger di Schmitt a CMOS

Se $V_I = 0 \Rightarrow V_X = V_{DD} \Rightarrow V_O = 0$. Quando V_I aumenta i due transistori del primo stadio e il transistore a canale p del secondo stadio conducono ($V_O = 0$ fino a quando l'uscita non commuterà). Il circuito equivalente del pilotaggio dell'invertitore di uscita equivale a:

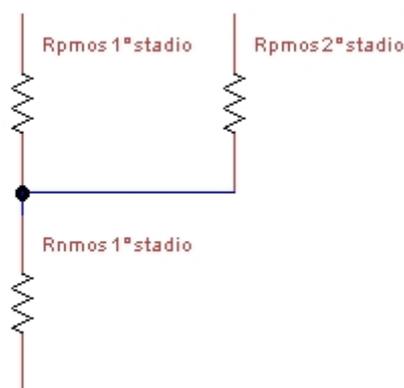


Figura 8: Stadio equivalente del circuito di figura 6 in condizioni di non commutazione

Il partitore resistivo si sbilancia verso l'alto e di fatto il nodo X fa più fatica a scendere. Per cui quando $V_X = V_{DD}/2$ (per $V_I > V_{DD}/2$) l'uscita commuta. La soglia logica dell'intero circuito è quindi aumentata. In maniera del tutto analoga si ha che quando V_I diminuisce la soglia risulta $< V_{DD}/2$. La configurazione appena studiata è di tipo non invertente, per cui se vogliamo ottenere un Trigger di Schmitt a isteresi in configurazione invertente sarà sufficiente togliere il gate NOT al nodo di uscita del circuito complessivo equivalente. I circuiti integrati della famiglia CMOS che eseguono questa tipologia di circuito sono il CD40106 (Schmitt Trigger inverters with hysteresis) e il CD4584 (Schmitt Trigger Buffers with hysteresis). Ma il fatto che questi dispositivi abbiano la proprietà di avere una caratteristica ingresso – uscita con isteresi, ci permette di utilizzarli come oscillatori in una configurazione detta a multivibratore astabile⁹, per cui se connettiamo una giunzione capacità/resistenza all'ingresso del gate con la resistenza in retroazione, il nostro gate caricherà e scaricherà alternativamente la capacità, provvedendo in uscita un'onda quadra. Il diagramma temporale nella figura sottostante ci mostra come lavorerà il concetto di randomicità (si suppone un utilizzo di quattro oscillatori):

⁹ Si definisce multivibratore astabile una particolare tipologia di circuito che esegue un'operazione di oscillazione basata su un innesco di tipo esponenziale e che non ha punti di riposo stabili, per cui esegue (teoricamente!) il moto oscillatorio fino all'infinito.

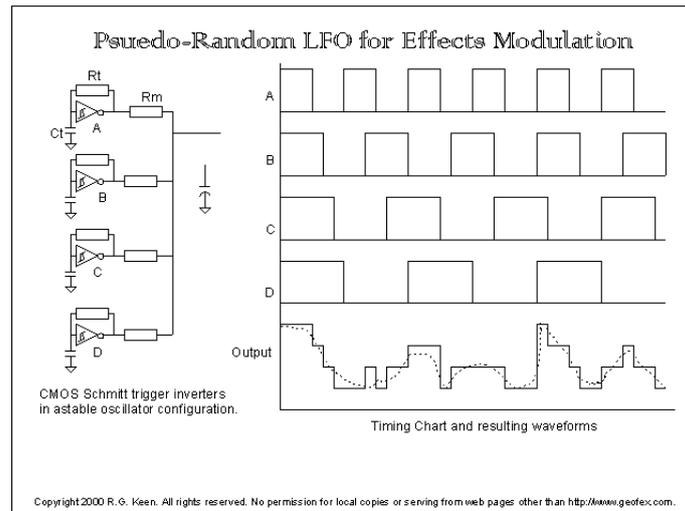


Figura 9: Diagramma temporale di un LFO pseudo - randomico a Trigger

Quando viene fornita alimentazione al circuito, tutte le capacità si trovano a 0V, in questo modo tutte le uscite dei singoli oscillatori saranno a livello alto. La tensione della capacità comincerà a crescere secondo una rampa esponenziale attraverso il resistore di feedback. Se le uscite sono tutte a livello alto, l'uscita del LFO sarà al massimo che si possa raggiungere (tipicamente la tensione di alimentazione). Quando il più veloce degli oscillatori, A in questo caso, commuta allo stadio con livello basso, l'uscita viene "tirata" anch'essa verso il basso, ma gli altri tre oscillatori cercano di mantenere l'uscita del LFO a circa $\frac{3}{4}$ del valore massimo. Quando l'oscillatore B commuta, l'uscita verrà riportata verso il basso di un altro step (portando l'uscita a circa $\frac{1}{2}$ del valore massimo). La cosa più interessante è che per questa famiglia di oscillatori, c'è un continuo interferimento delle condizioni di funzionamento del circuito, le quali porteranno ad avere in uscita delle LFW¹⁰ piuttosto particolari. L'uscita dello pseudorandom LFO può essere (dal momento che stiamo utilizzando la tecnologia digitale) troppo "spigolosa" nella sua forma d'onda e questo non va molto d'accordo con gli effetti per audio processing perché ciò introduce distorsione sul suono di uscita, ma connettendo una capacità di "smoothing" il problema viene limitato in buona parte. Bisogna fare particolare attenzione al fatto che la LFW prodotta può essere di ampiezza pari al massimo della tensione di alimentazione del circuito o a 0V e da ciò si evince che la maggior parte del tempo è speso a $V_{DD}/2$. Nell'utilizzare questo tipo di LFO con effetti di audio processing, sarà necessaria una operazione di DC Adjust per avere un matching perfetto con il circuito che vogliamo controllare. Un resistore connesso ad esempio verso massa, abbasserà il livello di segnale da V_{DD} a $V_{DD} * (R_{||} / R_{pulldown})$, dove $R_{||}$ è la combinazione parallelo di tutte le resistenze R_m e $R_{pulldown}$ è la resistenza aggiunta verso massa. Un resistore disaccoppiato ad una tensione di polarizzazione V_{bias}

¹⁰ Acronimo di Low Frequency Waveform.

a $V_{DD}/2$ ridurrà semplicemente i picchi di segnale, ma lascerà invariato in valore medio. Un resistore connesso alla tensione di alimentazione del LFO porterà uno shifting verso la tensione di alimentazione della LFW prodotta, ma riducendo in questo caso l'ampiezza della LFW spostata nei pressi della tensione di alimentazione. Ecco il listato PSPICE del LFO in funzionamento base:

Codice PSPICE per la realizzazione di un LFO pseudorandomico basato su quattro Trigger di Schmitt a isteresi (funz. base)

```
* modelli dei transistori MOS (per comodità si utilizzano transistori con medesimo
fattore di forma * (non reale))
.model mosp pmos level=1 vto=-1v kp=80u
.model mosn nmos level=1 vto=-1v kp=80u
* istanziazione dei sottocircuiti TRIGGER (Input Output Vdd)
.subckt trigger 1 5 3
M1 3 2 1 3 mosp L=1u W=1u
M2 1 2 0 0 mosn L=1u W=1u
M3 3 5 1 3 mosp L=1u W=1u
M4 1 5 0 0 mosn L=1u W=1u
.ends
* costruzione dei quattro oscillatori con calling dei quattro subckt e uscita comune
X1 6 10 14 trigger
X2 7 11 14 trigger
X3 8 12 14 trigger
X4 9 13 14 trigger
* piazzamento dei componenti passivi
* oscillatore nr.1 è dimensionato per funzionare a 15 Hz quindi omega = 47.1 rad/s e
fissando C = 47 * uF R = 1.42 KOhm
CT1 6 0 47u
RT1 6 10 1.42K
* oscillatore nr.2 è dimensionato per funzionare a 10 Hz quindi omega = 31.4 rad/s e
fissando C = 47 * uF R = 2.13 KOhm
CT2 7 0 47u
RT2 7 11 2.13K
* oscillatore nr.3 è dimensionato per funzionare a 5 Hz quindi omega = 15.7 rad/s e
fissando C = 47 * uF R = 4.26 KOhm
CT3 8 0 47u
RT3 8 12 4.26K
* oscillatore nr.4 è dimensionato per funzionare a 1 Hz quindi omega = 3.14 rad/s e
fissando C = 47 * uF R = 21.3 KOhm
CT4 9 0 47u
RT4 9 13 21.3K
*Connessione dei resistori di mixing
RM1 10 15 1K
RM2 11 15 1K
RM3 12 15 1K
```

```

RM4 13 15 1K
*Connessione delle alimentazioni
VDD 14 0 9V
*Analisi finale
.tran 100ms 1s
.probe
.end

```

Per ottenere delle LFW variabili ulteriormente, è sufficiente connettere dei potenziometri al posto dei quattro resistori RTx per il timing del trigger. In caso di connessione della capacità di smoothing in uscita, per regolarizzare i picchi della LFW prodotta, si ottiene aggiungendo la seguente riga di codice nel file .cir, una forma d'onda molto più morbida:

```

CSMOOTHING 15 0 47u

```

Un parametro molto importante che si può valutare tramite le simulazioni eseguite con PSPICE, è il consumo di potenza. Esso infatti, stabilisce non solo la quantità necessaria di corrente che il circuito assorbe in condizioni di funzionamento, ma anche l'energia richiesta e di conseguenza la problematica dello smaltimento del calore. Infatti, se si considera che un circuito di questo tipo (ovviamente valutato con le simulazioni) consuma all'incirca 160 mW per stadio di oscillazione, ci si accorge ben presto che, usando il teorema di Tellegen sulla potenza additiva, le potenze di ogni stadio si sommano e per una realizzazione a quattro stadi, il consumo di potenza può arrivare nell'intorno di 640 mW, il che consiste, ad esempio in un effetto stompboxes o rack – mounted ad alta densità di integrazione di componenti sulla scheda, con una tensione di alimentazione al massimo di 18 VDC, in un riscaldamento piuttosto dannoso per l'intero circuito. Ecco perché di solito, questa tipologia di oscillazione viene impiegata all'interno dei sintetizzatori, i quali, a scapito di un maggiore ingombro in termini di spazio, permettono un migliore smaltimento del calore e non hanno problemi per quanto riguarda la fornitura dell'alimentazione, dal momento che questi strumenti musicali sono collegati direttamente alla tensione di rete che verrà successivamente e opportunamente trasformata. Le simulazioni di questo circuito sono state svolte su un solo oscillatore e poi replicato, in quanto si dispone di una versione di PSPICE Student Edition, la quale impone delle limitazioni sul numero dei transistori all'interno del circuito da simulare, per cui chi avesse una versione di PSPICE con minori limitazioni, può usare direttamente il codice completo che si trova nella directory ../Codice/PSPICE con estensione .cir.

2.3.2 LFO digitali Regular Stair Stepped

Questa tipologia di oscillatori è particolarmente apprezzata nella creazione di effetti pedal – board, perché permette una migliore integrazione tra mondo analogico e digitale ed un ridotto consumo di potenza. Per generare delle LFW “quasi” perfettamente simili alle LFW che si otterrebbero con LFO analogici e VCLFO, si può utilizzare la tecnica Regular Stair Stepped¹¹ che utilizza come fulcro di funzionamento un Walking – Ring Counter, detto anche Twisted – Ring Counter, Johnson Counter o Mobius Counter.

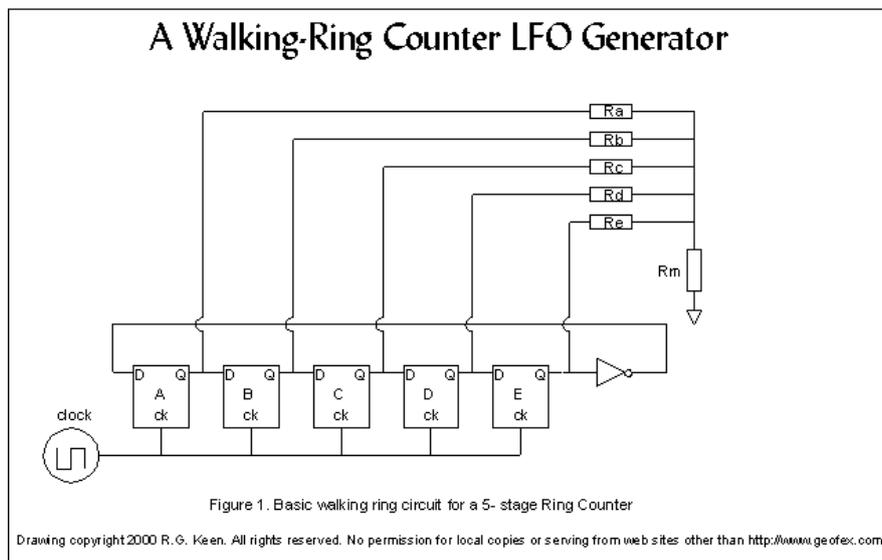


Figura 10: Schema circuitale di un Walking - Ring Counter a 5 stadi

Il contatore è una serie di flip – flop D – type tutti connessi ad un clock comune. Un flip – flop “D” accetta qualsiasi livello logico di ingresso presente sulla linea D (data) quando il clock diventa alto e effettua il latching del dato su Q. Con un array di questi dispositivi, il livello logico che appare all’ingresso D del primo flip – flop viene passato verso gli altri con un intervallo di tempo da un dispositivo all’altro pari ad un periodo di clock (+ gli eventuali ritardi introdotti dalla famiglia logica considerata e i ritardi di propagazione del circuito). Quello che fa diventare questo contatore un “Walking - Ring” è che l’uscita dell’ultimo flip – flop è invertita e retroazionata a D del primo flip – flop. Quando l’alimentazione viene fornita al circuito per la prima volta, prima del primo periodo di clock, assumiamo che tutti i flip – flop siano a “0”. Questo significa che il gate NOT ha in ingresso uno “0” logico e retroazionerà un “1” logico all’ingresso D del primo flip – flop della catena. Al primo periodo di clock, il valore “1” viene “latchato” su Q del primo stadio e tutti gli

¹¹ È una tecnica molto recente, sviluppata verso la fine degli anni ’80. In questi anni, Xilinx ne ha ripreso il suo utilizzo all’interno delle sue FPGA usate con gli effetti rack – mounted.

altri flip – flop vedranno al loro ingresso uno “0”. Il gate NOT vede ancora uno “0” e di conseguenza porterà ancora un “1” in ingresso alla catena. Al secondo periodo di clock, il primo stadio “latcha” un altro “1” insieme però in questo caso, al secondo stadio. Gli altri flip – flop rimangono ancora a “0”. Ad ogni periodo di clock quindi, un “1” viene traslato verso destra (ultimo stadio), fino a riempire la stringa di “1”. A questo punto il gate NOT retroazionerà uno “0” verso l’ingresso e così via...

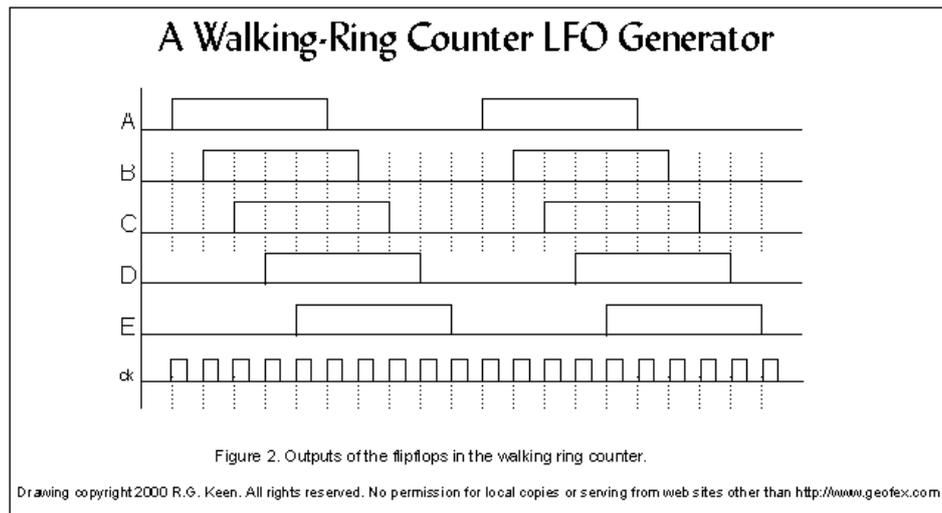


Figura 11: Uscita dei flip - flop di ogni stadio

Quello che rende veramente utile questo tipo di configurazione è che la natura “overlapped” delle uscite, le rende facilmente utilizzabili come generatori di forme d’onda, nel nostro caso di LFW. La tabella di funzionamento del circuito sarà quindi (supponendo di avere cinque stadi):

CLK Tick	Stadio A	Stadio B	Stadio C	Stadio D	Stadio E
1	0	0	0	0	0
2	1	0	0	0	0
3	1	1	0	0	0
4	1	1	1	0	0
5	1	1	1	1	0
6	1	1	1	1	1
7	0	1	1	1	1
8	0	0	1	1	1
9	0	0	0	1	1
10	0	0	0	0	1

11	0	0	0	0	0
----	---	---	---	---	---

Se consideriamo questi “1” e “0” come tensioni analogiche, possiamo usare resistori per ottenere differenti forme d’onda. Il metodo più semplice da ottenere è appunto il metodo Regular Stair Stepped. Questo si può fare utilizzando un resistore uguale in serie ad ognuno degli stadi del contatore e li connettiamo tutti ad un “mixing resistor” R_m . Con tutte le uscite a “0” la tensione ai capi della resistenza R_m è circa 0V, mentre con una delle uscite a “1”, la tensione ai capi di R_m è data dal partitore di tensione visto come $V_m = V_{out} * (R_m / R_m + R_{on})$, dove R_{on} rappresenta la somma dei resistori associati agli stadi del contatore pari a “1”. Il trucco nell’ottenere delle LFW prossime a quelle analogiche, sta nell’utilizzo di resistori di diversa grandezza, ad esempio per ottenere una buona forma d’onda sinusoidale si cerca di avere i valori di resistenza degli stadi intermedi più piccoli rispetto agli altri stadi e calcolati in modo da approssimare una sinusoide.

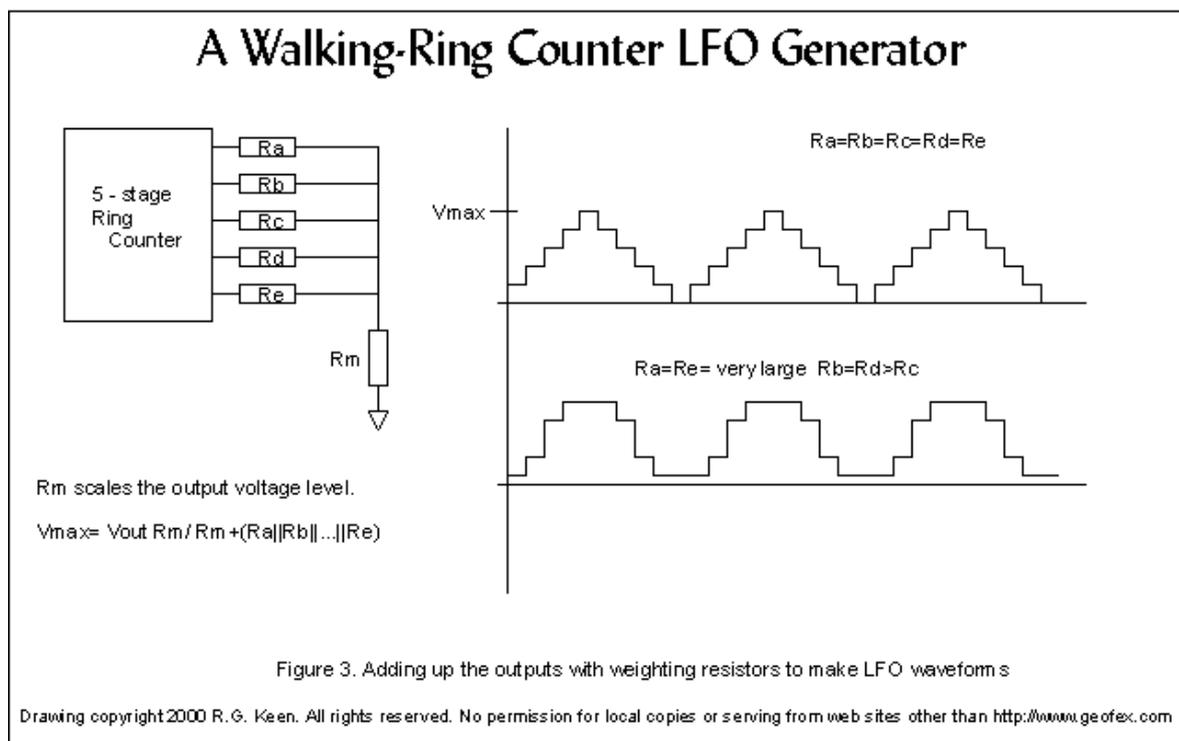


Figura 12: Metodi per ottenere diverse LFW

L’ approssimazione migliore è comunque sempre data dall’ utilizzo di Walking – Ring Counter con un numero di stadi molto elevato e di utilizzare delle capacità di “smoothing” in uscita per filtrare gli “spigoli” della LFW, ma l’ importante è comunque, quando si utilizzano LFO digitali Regular Stair Stepped, di cercare di non utilizzare troppo degli step non uniformi in modo da non cadere in una pessima approssimazione delle LFW analogiche. Per simulare il funzionamento di un Walking –

Ring Counter usiamo il software PSPICE con lo Schematic Editor incluso nel pacchetto OrCAD Demo e valutiamo le forme d'onda di uscita. Con lo schema circuitale mostrato nella figura in basso, possiamo ottenere un LFO che genererà una LFW pseudo – triangolare ad una frequenza di circa 10 Hz utilizzando Regular Stair Step:

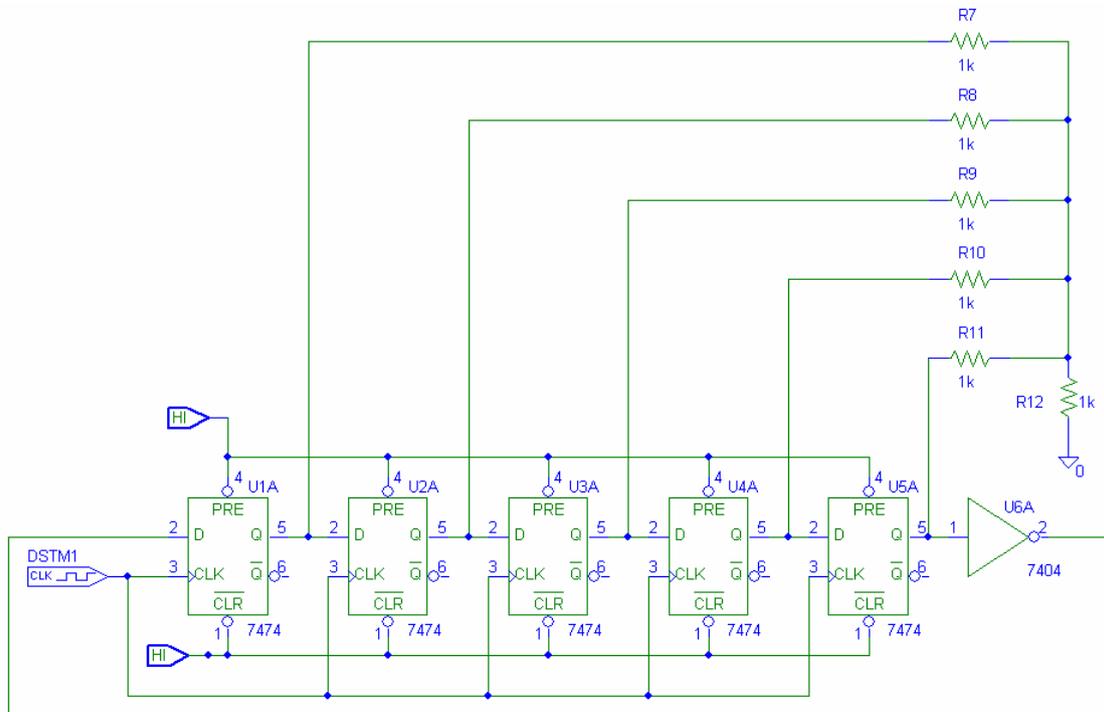


Figura 13: Schema del Walking - Ring Counter con RStadio uguali

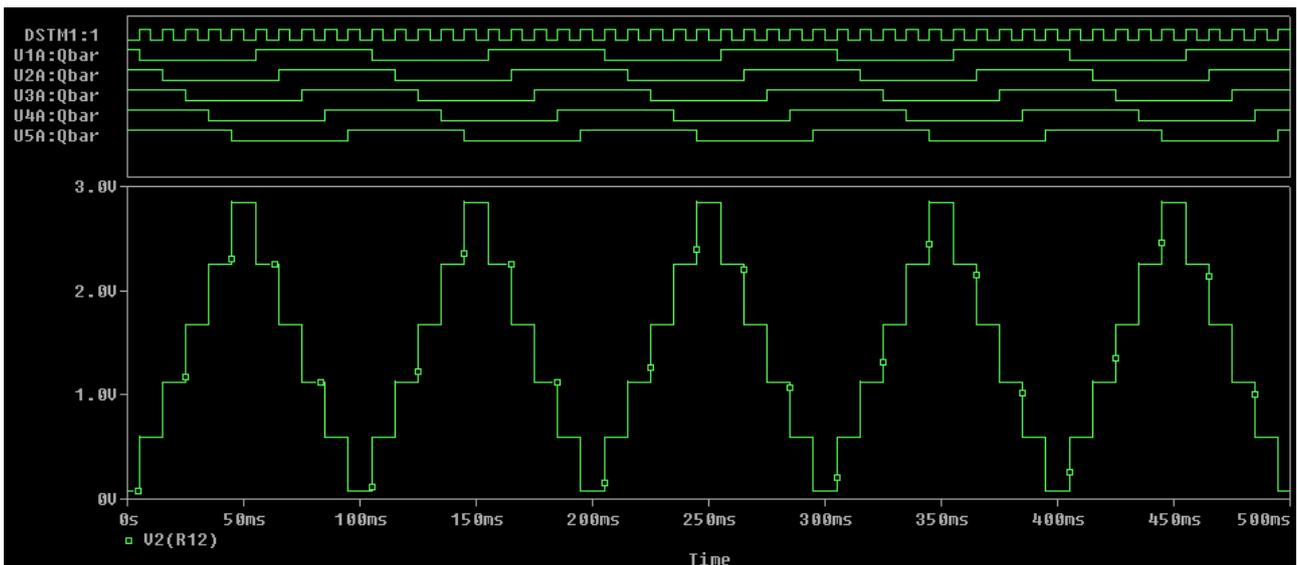


Figura 14: LFW risultante dalla simulazione dell' LFO

Utilizzando invece sempre lo stesso schema, ma modificando i valori delle resistenze dei vari stadi, è possibile ottenere una LFW pseudo – sinusoidale sempre alla stessa frequenza:

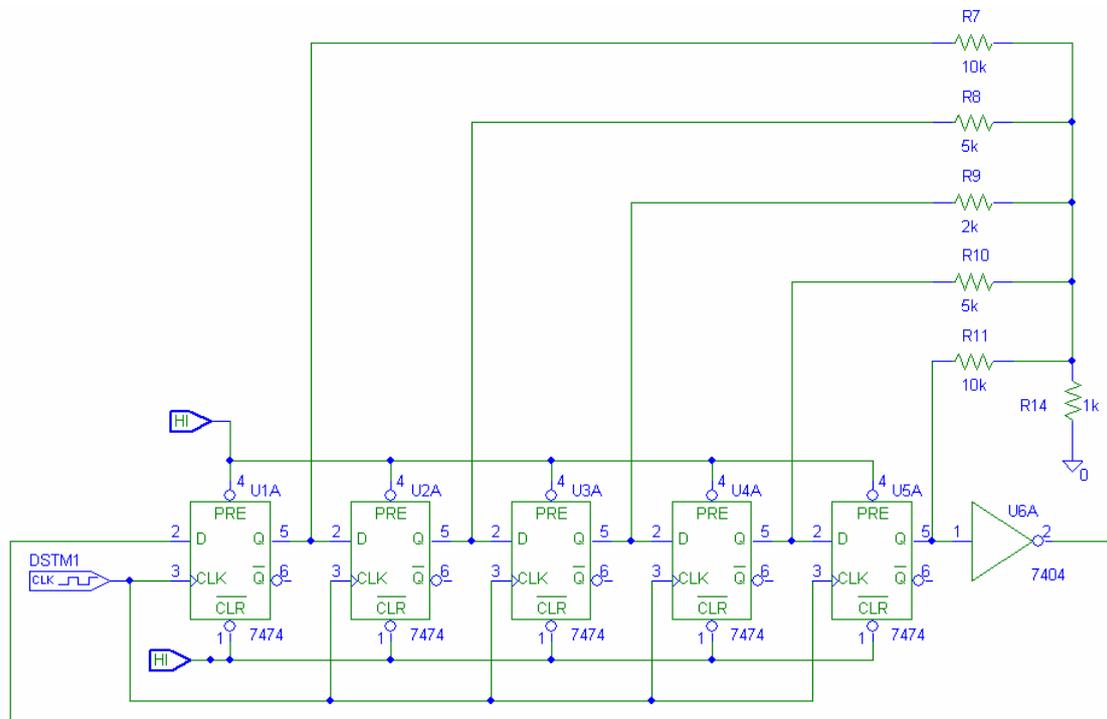


Figura 15: Schema del Walking - Ring Counter con RStadio diverse

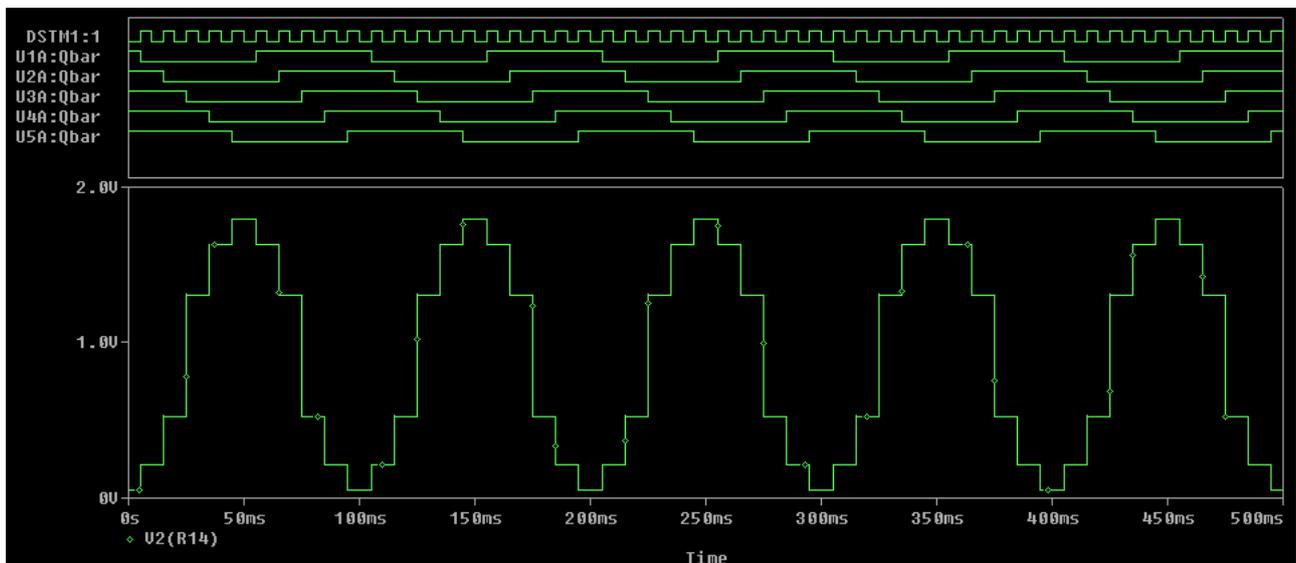


Figura 16: LFW risultante dalla simulazione dell' LFO

Si noti come cambiano le cose durante la fase di generazione delle due LFW aggiungendo una capacità di smoothing da $47 \mu\text{F}^{12}$ per “regolarizzare” gli scalini delle transizioni introducendo un filtraggio di tipo passa – basso del primo ordine con frequenza di cutoff a circa 3.38 Hz:

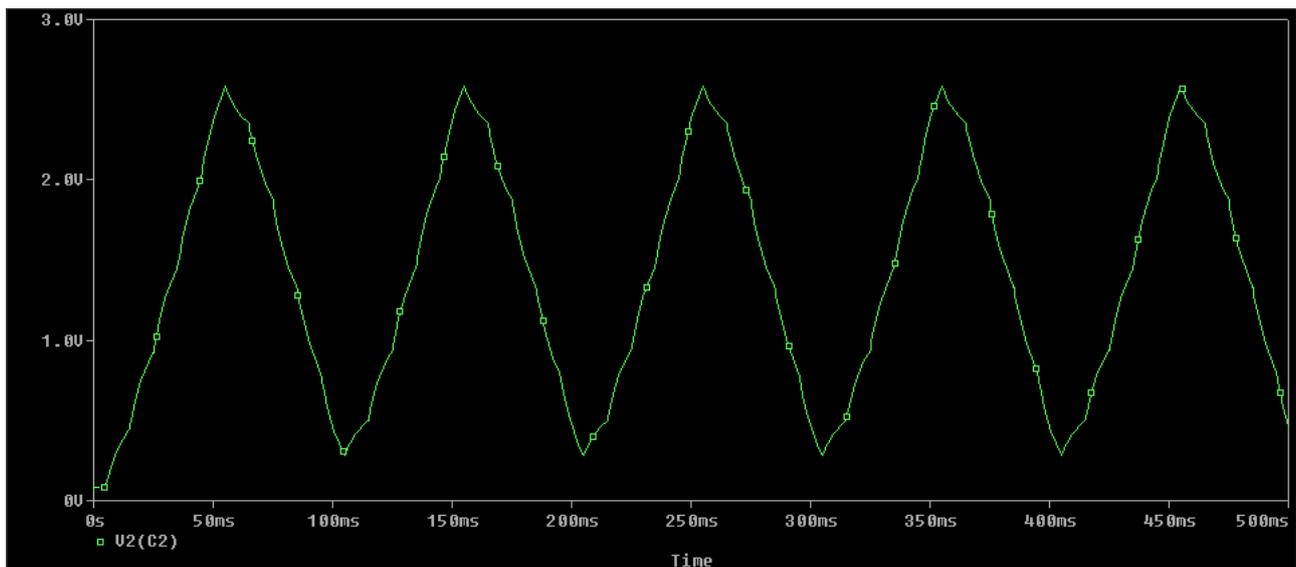
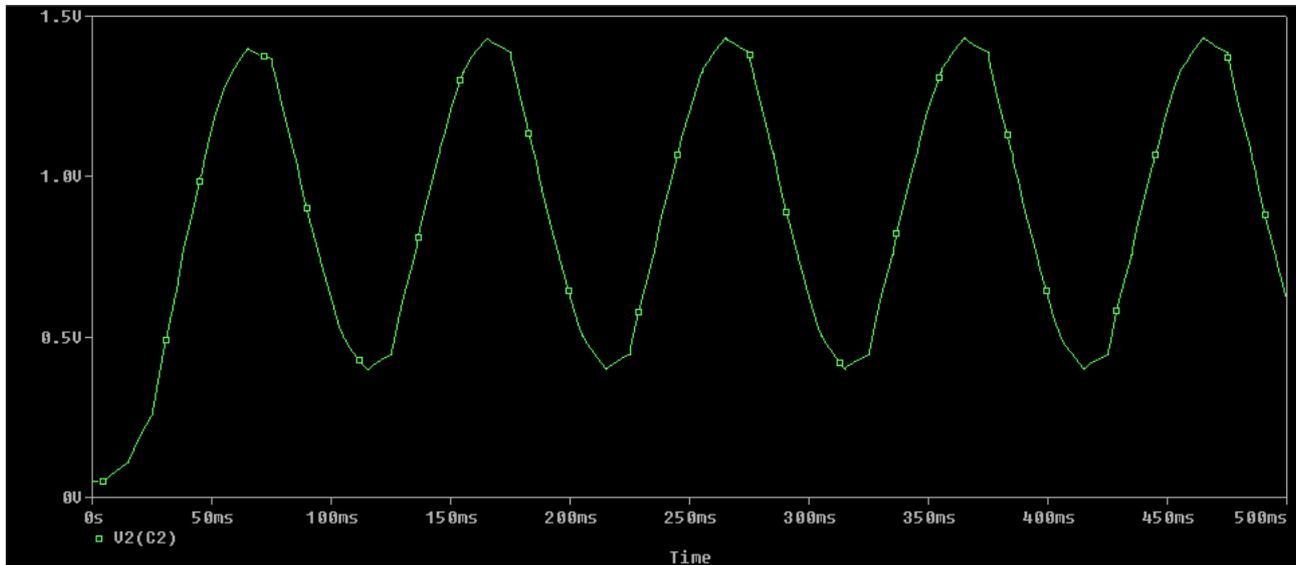


Figura 17: LFW ottenute con il procedimento di Smoothing

L'intero codice sviluppato è disponibile sul CD-ROM allegato alla tesi con percorso `../Codice/PSPICE` con estensione `.sch` per gli schematici e `.out / .prb` per i risultati delle simulazioni.

¹² Si è scelta una capacità non troppo grande per evitare il degrado dei fronti di salita della LFW.

2.4 Sviluppo dell' LFO – Maker con Matlab

Dopo aver analizzato la maggior parte delle realtà fisiche che stanno alla base della realizzazione degli LFO, entriamo nella prima fase di progetto della tesi: la creazione dell'applicazione LFO – Maker. La scelta di separare la creazione delle LFW in un altro programma non è stata fatta a caso. Questa soluzione permette infatti, di avere una separazione in termini di astrazione del codice che riguarderà gli algoritmi di modulazione del segnale audio ed inoltre ci permetterà di avere sotto controllo tutti i parametri relativi di un LFO, senza dover accettare dei compromessi relativi per l'occupazione della memoria e dello spazio fisico dell'applicazione. Si utilizzerà per questa applicazione il software Matlab, con un'estensione basata sullo sviluppo di applicazioni di tipo GUI detta GUIDE¹³, a cui si può accedere dalla console di Matlab digitando “guide” e l'editor/debugger principale, a cui si può accedere dalla console di Matlab digitando “edit”. Una volta attivato il GUIDE compare una “empty figure” sulla quale andremo a posizionare tutti i controlli e i grafici dell'applicazione e un “develop box” in cui si dovrà andare, man mano che si posizionano gli oggetti, ad istanziare gli stessi e definire il codice associato ad ogni azione compiuta all'interno della nostra applicazione. Come si intende sviluppare questa applicazione? Innanzitutto, abbiamo specificato che si tratta di uno stand – alone GUI che però dovrà stare all'interno di un environment completo per l'elaborazione dei segnali audio (che abbiamo chiamato Nova – Mod Tools), per cui se vogliamo avere la massima interazione con gli altri programmi che svilupperemo nel corso di questa trattazione, dobbiamo per forza imporre un funzionamento basato sul file – saving & parsing¹⁴; dopodiché abbiamo anche detto che vogliamo avere sottocontrollo tutte le funzioni di un LFO, per cui l'applicazione costruita dovrà avere i seguenti controlli¹⁵ base, che esistono in ogni LFO:

- Rate: rappresenta il controllo sulla frequenza della LFW in uscita e dovrà essere il più possibile user – friendly.
- Amplitude: rappresenta il controllo sull'ampiezza della LFW in uscita e dovrà essere il più possibile user – friendly.
- LFW Type: rappresenta il controllo che stabilisce la forma d'onda in uscita dal nostro LFO e permetterà di lavorare con le seguenti forme d'onda: Sinusoidale; Cosinusoidale;

¹³ Acronimo di Graphical User Interface Development Environment.

¹⁴ Procedura usata negli algoritmi per ricevere e salvare informazioni su file provenienti o in uscita verso altre applicazioni.

¹⁵ Per controlli si intendono gli oggetti con cui l'utente può interagire sulla figura.

Triangolare; Quadra; Esponenziale – Sinusoidale; Rampa su/giù e l'innovativa Pseudo – Random basata su una modalità di randomizzazione del moltiplicatore di rate¹⁶ e la sovrapposizione di ben 6 stadi di oscillazione sinusoidale, in modo da garantire sempre un segnale diverso.

- LFW Phase/Width/Duty: rappresenta il controllo che stabilisce la fase da 0° a 359° della LFW in uscita dal nostro LFO, l'ampiezza dei massimi del segnale se triangolare e il duty cycle se si tratta di un'onda quadra.
- LFW DC Offset: rappresenta il controllo che stabilisce l'offset della forma d'onda nei confronti della DC. Questo implica la generazione di un segnale che avrà una componente spettrale a frequenza nulla e quindi fortemente distorto, con un alto dispendio di energia nel sistema.

Inoltre, si disporrà delle seguenti innovative funzionalità:

- LFW rectification envelope: rappresenta il controllo che permette di dare alla forma d'onda un involuppo rettificato, in modo da garantire alla LFW un suono più pronunciato che può essere inserito o meno dall'utente.
- Simulate Regular Stair Stepped LFO: questa funzione simulerà il funzionamento di un LFO basato sulla configurazione a Regular Stair Step, con l'introduzione della capacità di smoothing.
- Simulate Wall – Mart power supply: questa funzione simulerà il funzionamento di un LFO alimentato con un trasformatore da rete a DC, portatile, con l'introduzione di rumore di alimentazione alla frequenza di 50 Hz.
- Algorithm precision: questa funzione permette di scegliere il metodo con cui rappresentare i dati della LFW nel calcolo del plot relativo al LFO.
- LFW Damping Factor: rappresenta il controllo che permette di stabilire l'involuppo esponenziale che deve seguire la LFW tramite il damping della forma d'onda principale.

¹⁶ Il moltiplicatore di rate è il numero che si moltiplica a rate usando la formula: $\text{signal}(2*\pi*\text{multiplier}*rate*t)$.

- LFO Glory of the past & the present: non si tratta in questo caso di un controllo, ma di una libreria di preset che simulano il funzionamento degli LFO più comuni delle unità passate e del presente.

Lo schema di funzionamento prevedrà quindi: creazione della forma d'onda, modifica della stessa tramite i controlli, salvataggio su file proprietario e trasferimento all'applicazione principale. Il formato con cui verranno salvate le LFW prodotte, sarà un'estensione proprietaria .lfo. Siccome la LFW prodotta dovrà poi essere utilizzata dall'applicazione principale per eseguire il processo di modulazione, ed essendo questa forma d'onda, ottenuta per interpolazione di valori campionati, risulterà necessario definire una temporizzazione del segnale LFW che permetta il matching perfetto con la temporizzazione del segnale audio arbitrario in ingresso al sistema di modulazione. Per avere un'ottima rappresentazione della forma d'onda in uscita dall'LFO si è scelto di utilizzare una frequenza di campionamento basata sulla frequenza di campionamento del file di cui si desidera effettuare la processazione, in modo che l'utente possa scegliere la capacità di memoria richiesta per la memorizzazione della forma d'onda¹⁷ in base alla rappresentazione e alla precisione scelta per lo sviluppo dell'algoritmo. Ora che si sono definiti tutti i parametri necessari allo sviluppo del LFO – Maker possiamo cominciare a scrivere il codice e posizionare i controlli:

¹⁷ La LFW verrà salvata su disco, quindi la capacità di memoria richiesta e la velocità di esecuzione dell'algoritmo dipendono dalla lunghezza del file da processare, dalla sua frequenza di campionamento e dalla precisione scelta per l'algoritmo di generazione LFO.

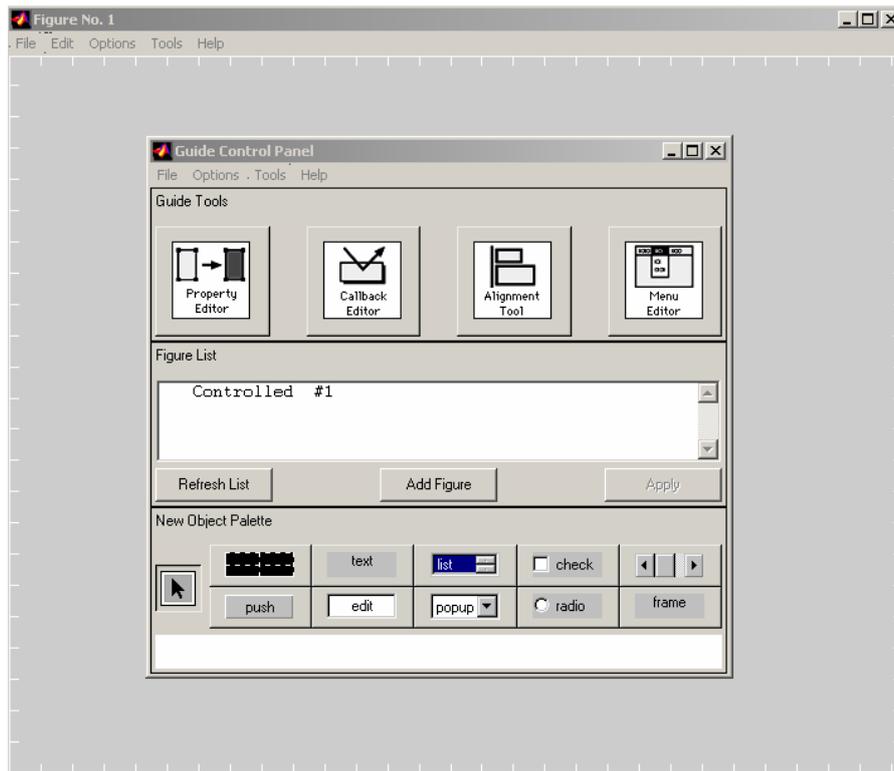


Figura 18: Ecco come si presenta la finestra principale di GUIDE

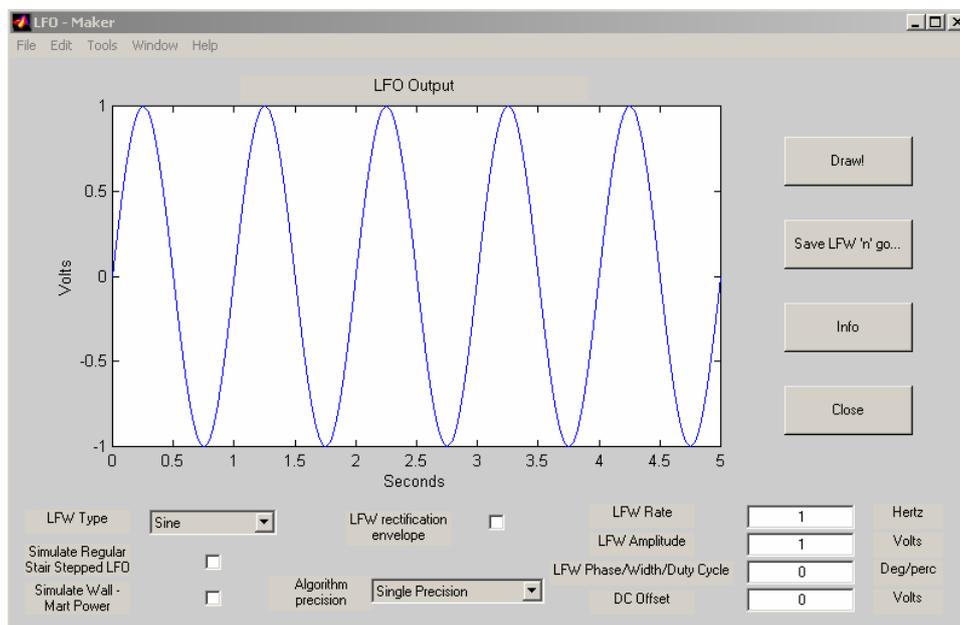


Figura 19: Ed ecco come si presenta la finestra principale di LFO - Maker dopo l'aggiunta dei controlli

È chiaro che ad ogni controllo dovremo associare un codice che definirà l'operazione da svolgere quando avviene la pressione o il cambiamento di stato di un controllo: questa operazione prende il nome di definizione dei Callback. Vediamo come si può fare con GUIDE a definire i Callback tramite il Callback Editor; per quanto riguarda il pulsante "Close" ad esempio, l'unica operazione

da svolgere sarà il catching della figura corrente e chiuderla, quindi si associa al pulsante il seguente Callback: `close(gcf)`,¹⁸

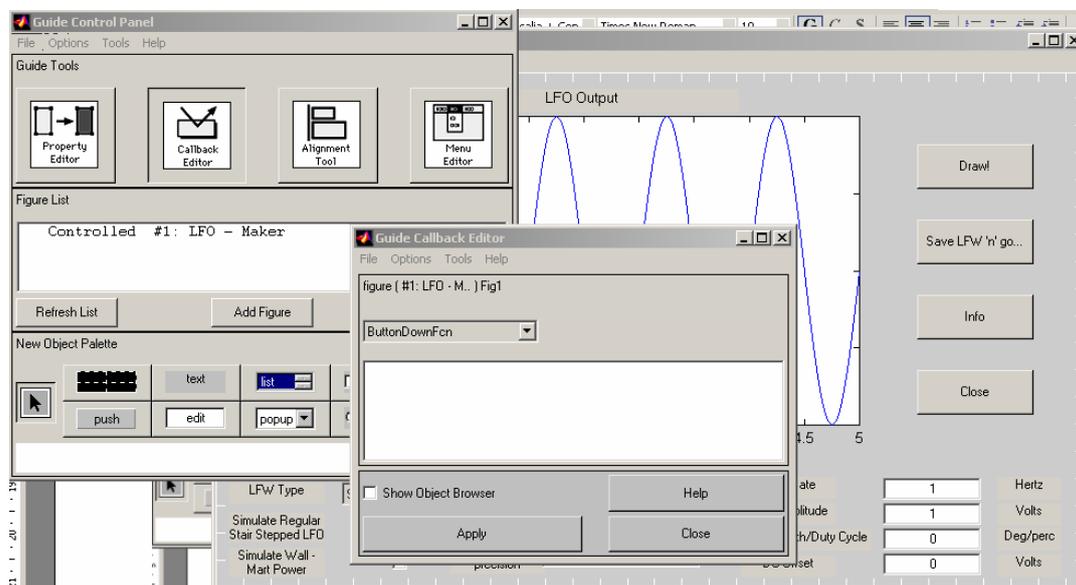


Figura 20: Come aprire il Callback editor e inserire il codice corrispondente

Una programmazione di questo tipo è però sconsigliabile in termini di occupazione di memoria e di velocità nella compilazione del codice, per cui si preferisce optare per una soluzione basata su switchyard programming, che permette l'organizzazione delle principali funzioni di Callback associate ad ogni pulsante, in particolari "case" di una funzione definita a priori che in questo caso abbiamo chiamato `fcnLFO`. Vediamo il codice particolare di questa funzione, cercando di valutarne tutte le relative sfumature.

Questa parte serve per l'inizializzazione della funzione, dichiarando le variabili globali relative ai controlli per la memorizzazione su supporto di massa.

```
%Function File esterno per i callback
%
%written by Zambelli Cristian
function fcnLFO(action)
global popup rsstep wallsupply rate amp phase dcoffset rectific damping;
```

Da qui comincia lo script vero e proprio basato sullo switch dell'azione che arriva al software. Il case 'plot' fa riferimento all'azione compiuta alla pressione del pulsante Draw! presente sulla figure

¹⁸ `gcf` è l'acronimo che usa Matlab per definire la `GetCurrentFigure`.

– board. La prima operazione è quella di ricavare l’handle¹⁹ del menu che va a scegliere che tipo di forma d’onda utilizzerà l’LFO e la precisione (dal secondo menu) con cui avverrà la rappresentazione della forma d’onda in termini di bit utilizzati per la rappresentazione di un valore numerico²⁰; dopodiché (utilizzando una replicazione sequenziale del codice), si valuta se si sta utilizzando una simulazione di un LFO Regular Stair Stepped e se ciò è vero si setta la frequenza di campionamento ad un valore drasticamente basso, in modo da avere una rappresentazione “a scalini” come nel sistema originale o in alternativa la frequenza di campionamento rimane inalterata; si ricavano gli handle di ogni controllo, li si salva in memoria e controlla se questi hanno un valore significativo, immesso dall’utente; infine, un ciclo for visualizzerà una “wait bar” mentre avviene il calcolo dei valori della LFW, in modo da far capire all’utente che il processo di computazione sta avvenendo e terminerà disegnando sul plot²¹ il valore finale della LFW.

```
switch(action)
case 'plot'
    puphandle=findobj(gcf,'Tag','PopupMenu1');
    pup=get(puphandle,'Value');
    puphandle2=findobj(gcf,'Tag','PopupMenu2');
    pup2=get(puphandle2,'Value');
    %LFW sinusoidale
    if pup==1
        rsstephandle=findobj(gcf,'Tag','Checkbox1');
        rsstep=get(rsstephandle,'Value');
        if rsstep==0
            samplerate=11025;
        elseif rsstep==1
            samplerate=50;
        end
        wallsupplyhandle=findobj(gcf,'Tag','Checkbox2');
        wallsupply=get(wallsupplyhandle,'Value');
        ratehandle=findobj(gcf,'Tag','EditText1');
        rate=str2num(get(ratehandle,'String'));
        if (rate<0.3)|(rate>16)
            h=warndlg('This is not a right rate value for an LFO please type a value from 0.3
to 16 Hz','Warning!!');
        end
        amphandle=findobj(gcf,'Tag','EditText2');
        amp=str2num(get(amphandle,'String'));
        if (amp<0)|(amp>18)
```

¹⁹ Rappresenta il codice associato dal software ad un particolare controllo.

²⁰ Si potrebbe pensare che una rappresentazione con un minor numero di bit, corrisponda ad avere una forma d’onda simile ad una LFW Regular Stair Stepped, ma non è così, perché in questo caso se i valori di picco della LFW fossero tra -1 e 1 si avrebbe la perdita dell’informazione, cosa che non avviene invece nel sistema originale comandato dai flip – flop.

²¹ Il plot in questo caso rappresenta gli assi della figura al centro della finestra del software.

```

    h=warndlg('This is not a right amplitude value for an LFO please type a value from
0 to 18 V','Warning!!');
end
phasehandle=findobj(gcf,'Tag','EditText3');
phase=(str2num(get(phasehandle,'String'))*2*pi)/360;
if (phase<0)|(phase>359)
    h=warndlg('This is not a right phase value for an LFO please type a value from 0 to
359 degrees','Warning!!');
end
dchandle=findobj(gcf,'Tag','EditText4');
dcoffset=str2num(get(dchandle,'String'));
if (dcoffset<-18)|(dcoffset>18)
    h=warndlg('This is not a right dcoffset value for an LFO please type a value from -
18 to 18 V','Warning!!');
end
dampinhandle=findobj(gcf,'Tag','EditText5');
damping=str2num(get(dampinhandle,'String'));
rectifhandle=findobj(gcf,'Tag','Checkbox3');
rectif=get(rectifhandle,'Value');
h=waitbar(0,'Performing Calculus on LFO... Please Wait');
for i=1:100,t=0:1/samplerate:5;
    waitbar(i/100);
end
close(h);
if wallsupply==0
    x=dcoffset+amp*exp(-(t*damping)).*sin(2*pi*rate*t+phase);
elseif wallsupply==1
    x=dcoffset+amp*exp(-(t*damping)).*sin(2*pi*rate*t+phase)+0.1*sin(2*pi*50*t);
end
if rectif==1
    y=abs(x);
elseif rectif==0
    y=x;
end
switch(popup2)
case 1
    i=int8(y);
case 2
    i=int16(y);
case 3
    i=int32(y);
case 4
    i=single(y);
case 5
    i=double(y);
end
plot(t,i,'EraseMode','xor');

```

```

end
%%LFW cosinusoidale
if popup==2
rsstephandle=findobj(gcf,'Tag','Checkbox1');
rsstep=get(rsstephandle,'Value');
if rsstep==0
    samplerate=11025;
elseif rsstep==1
    samplerate=50;
end
wallsupplyhandle=findobj(gcf,'Tag','Checkbox2');
wallsupply=get(wallsupplyhandle,'Value');
ratehandle=findobj(gcf,'Tag','EditText1');
rate=str2num(get(ratehandle,'String'));
if (rate<0.3)|(rate>16)
    h=warndlg('This is not a right rate value for an LFO please type a value from 0.3
to 16 Hz','Warning!!');
end
amphandle=findobj(gcf,'Tag','EditText2');
amp=str2num(get(amphandle,'String'));
if (amp<0)|(amp>18)
    h=warndlg('This is not a right amplitude value for an LFO please type a value from
0 to 18 V','Warning!!');
end
phasehandle=findobj(gcf,'Tag','EditText3');
phase=(str2num(get(phasehandle,'String'))*2*pi)/360;
if (phase<0)|(phase>359)
    h=warndlg('This is not a right phase value for an LFO please type a value from 0 to
359 degrees','Warning!!');
end
dchandle=findobj(gcf,'Tag','EditText4');
dcoffset=str2num(get(dchandle,'String'));
if (dcoffset<-18)|(dcoffset>18)
    h=warndlg('This is not a right dcoffset value for an LFO please type a value from -
18 to 18 V','Warning!!');
end
damphandle=findobj(gcf,'Tag','EditText5');
damping=str2num(get(damphandle,'String'));
rectifhandle=findobj(gcf,'Tag','Checkbox3');
rectif=get(rectifhandle,'Value');
h=waitbar(0,'Performing Calculus on LFO... Please Wait');
for i=1:100,t=0:1/samplerate:5;
    waitbar(i/100);
end
close(h);
if wallsupply==0
    x=dcoffset+amp*exp(-(t*damping)).*cos(2*pi*rate*t+phase);

```

```

elseif wallsupply==1
    x=dcoffset+amp*exp(-(t*damping)).*cos(2*pi*rate*t+phase)+0.1*sin(2*pi*50*t);
end
if rectific==1
    y=abs(x);
elseif rectific==0
    y=x;
end
switch(popup2)
case 1
    i=int8(y);
case 2
    i=int16(y);
case 3
    i=int32(y);
case 4
    i=single(y);
case 5
    i=double(y);
end
plot(t,i,'EraseMode','xor');
end
%%LFW triangolare
if popup==3
rsstephandle=findobj(gcf,'Tag','Checkbox1');
rsstep=get(rsstephandle,'Value');
if rsstep==0
    samplerate=11025;
elseif rsstep==1
    samplerate=50;
end
wallsupplyhandle=findobj(gcf,'Tag','Checkbox2');
wallsupply=get(wallsupplyhandle,'Value');
ratehandle=findobj(gcf,'Tag','EditText1');
rate=str2num(get(ratehandle,'String'));
if (rate<0.3)|(rate>16)
    h=warndlg('This is not a right rate value for an LFO please type a value from 0.3
to 16 Hz','Warning!!');
end
amphandle=findobj(gcf,'Tag','EditText2');
amp=str2num(get(amphandle,'String'));
if (amp<0)|(amp>18)
    h=warndlg('This is not a right amplitude value for an LFO please type a value from
0 to 18 V','Warning!!');
end
phasehandle=findobj(gcf,'Tag','EditText3');
phase=str2num(get(phasehandle,'String'));

```

```

if (phase<-0.5)|(phase>0.5)
    h=warndlg('This is not a right width value for an LFO please type a value from -0.5
to 0.5','Warning!!');
end
dchandle=findobj(gcf,'Tag','EditText4');
dcoffset=str2num(get(dchandle,'String'));
if (dcoffset<-18)|(dcoffset>18)
    h=warndlg('This is not a right dcoffset value for an LFO please type a value from -
18 to 18 V','Warning!!');
end
dampinhandle=findobj(gcf,'Tag','EditText5');
damping=str2num(get(dampinhandle,'String'));
rectifhandle=findobj(gcf,'Tag','Checkbox3');
rectif=get(rectifhandle,'Value');
h=waitbar(0,'Performing Calculus on LFO... Please Wait');
for i=1:100,t=0:1/samplerate:5;
    waitbar(i/100);
end
close(h);
if wallsupply==0
    x=dcoffset+amp*exp(-t*damping).*sawtooth(2*pi*rate*t,phase+0.5);
elseif wallsupply==1
    x=dcoffset+amp*exp(-t*damping).*sawtooth(2*pi*rate*t,phase+0.5)+0.1*sin(2*pi*50*t);
end
if rectif==1
    y=abs(x);
elseif rectif==0
    y=x;
end
switch(popup2)
case 1
    i=int8(y);
case 2
    i=int16(y);
case 3
    i=int32(y);
case 4
    i=single(y);
case 5
    i=double(y);
end
plot(t,i,'EraseMode','xor');
end
%%LFW Square
if popup==4
rsstephandle=findobj(gcf,'Tag','Checkbox1');
rsstep=get(rsstephandle,'Value');

```

```

if rsstep==0
    samplerate=11025;
elseif rsstep==1
    samplerate=50;
end
wallsupplyhandle=findobj(gcf,'Tag','Checkbox2');
wallsupply=get(wallsupplyhandle,'Value');
ratehandle=findobj(gcf,'Tag','EditText1');
rate=str2num(get(ratehandle,'String'));
if (rate<0.3)|(rate>16)
    h=warndlg('This is not a right rate value for an LFO please type a value from 0.3
to 16 Hz','Warning!!');
end
amphandle=findobj(gcf,'Tag','EditText2');
amp=str2num(get(amphandle,'String'));
if (amp<0)|(amp>18)
    h=warndlg('This is not a right amplitude value for an LFO please type a value from
0 to 18 V','Warning!!');
end
phasehandle=findobj(gcf,'Tag','EditText3');
phase=str2num(get(phasehandle,'String'));
if (phase<0)|(phase>100)
    h=warndlg('This is not a right phase value for an LFO please type a value from 0 to
100 %','Warning!!');
end
dchandle=findobj(gcf,'Tag','EditText4');
dcoffset=str2num(get(dchandle,'String'));
if (dcoffset<-18)|(dcoffset>18)
    h=warndlg('This is not a right dcoffset value for an LFO please type a value from -
18 to 18 V','Warning!!');
end
dampinghandle=findobj(gcf,'Tag','EditText5');
damping=str2num(get(dampinghandle,'String'));
rectifhandle=findobj(gcf,'Tag','Checkbox3');
rectif=get(rectifhandle,'Value');
h=waitbar(0,'Performing Calculus on LFO... Please Wait');
for i=1:100,t=0:1/samplerate:5;
    waitbar(i/100);
end
close(h);
if wallsupply==0
    x=dcoffset+amp*exp(-t*damping).*square(2*pi*rate*t,phase);
elseif wallsupply==1
    x=dcoffset+amp*exp(-t*damping).*square(2*pi*rate*t,phase)+0.1*sin(2*pi*50*t);
end
if rectif==1
    y=abs(x);

```

```

elseif rectif==0
    y=x;
end
switch(popup2)
case 1
    i=int8(y);
case 2
    i=int16(y);
case 3
    i=int32(y);
case 4
    i=single(y);
case 5
    i=double(y);
end
plot(t,i,'EraseMode','xor');
end
%%LFW esponenziale - sinusoidale
if popup==5
rsstephandle=findobj(gcf,'Tag','Checkbox1');
rsstep=get(rsstephandle,'Value');
if rsstep==0
    samplerate=11025;
elseif rsstep==1
    samplerate=50;
end
wallsupplyhandle=findobj(gcf,'Tag','Checkbox2');
wallsupply=get(wallsupplyhandle,'Value');
ratehandle=findobj(gcf,'Tag','EditText1');
rate=str2num(get(ratehandle,'String'));
if (rate<0.3)|(rate>16)
    h=warndlg('This is not a right rate value for an LFO please type a value from 0.3
to 16 Hz','Warning!!');
end
amphandle=findobj(gcf,'Tag','EditText2');
amp=str2num(get(amphandle,'String'));
if (amp<0)|(amp>18)
    h=warndlg('This is not a right amplitude value for an LFO please type a value from
0 to 18 V','Warning!!');
end
phasehandle=findobj(gcf,'Tag','EditText3');
phase=(str2num(get(phasehandle,'String'))*2*pi)/360;
if (phase<0)|(phase>359)
    h=warndlg('This is not a right phase value for an LFO
please type a value from 0 to 359 degrees','Warning!!');
end
dchandle=findobj(gcf,'Tag','EditText4');
dcoffset=str2num(get(dchandle,'String'));

```

```

if (dcoffset<-18)|(dcoffset>18)
    h=warndlg('This is not a right dcoffset value for an LFO please type a value from -
18 to 18 V','Warning!!');
end
dampinhandle=findobj(gcf,'Tag','EditText5');
damping=str2num(get(dampinhandle,'String'));
rectifhandle=findobj(gcf,'Tag','Checkbox3');
rectif=get(rectifhandle,'Value');
h=waitbar(0,'Performing Calculus on LFO... Please Wait');
for i=1:100,t=0:1/samplerate:5;
    waitbar(i/100);
end
close(h);
if wallsupply==0
    x=dcoffset+amp*exp(-(t*damping)).*exp(-sin(2*pi*rate*t+phase));
elseif wallsupply==1
    x=dcoffset+amp*exp(-(t*damping)).*exp(-sin(2*pi*rate*t+phase))+0.1*sin(2*pi*50*t);
end
if rectif==1
    y=abs(x);
elseif rectif==0
    y=x;
end
switch(popup2)
case 1
    i=int8(y);
case 2
    i=int16(y);
case 3
    i=int32(y);
case 4
    i=single(y);
case 5
    i=double(y);
end
plot(t,i,'EraseMode','xor');
end
%%LFW Sawtooth
if popup==6
    rsstephandle=findobj(gcf,'Tag','Checkbox1');
    rsstep=get(rsstephandle,'Value');
if rsstep==0
    samplerate=11025;
elseif rsstep==1
    samplerate=50;
end
wallsupplyhandle=findobj(gcf,'Tag','Checkbox2');

```

```

wallsupply=get(wallsupplyhandle,'Value');
ratehandle=findobj(gcf,'Tag','EditText1');
rate=str2num(get(ratehandle,'String'));
if (rate<0.3)|(rate>16)
    h=warndlg('This is not a right rate value for an LFO please type a value from 0.3
to 16 Hz','Warning!!');
end
amphandle=findobj(gcf,'Tag','EditText2');
amp=str2num(get(amphandle,'String'));
if (amp<0)|(amp>18)
    h=warndlg('This is not a right amplitude value for an LFO please type a value from
0 to 18 V','Warning!!');
end
phasehandle=findobj(gcf,'Tag','EditText3');
phase=str2num(get(phasehandle,'String'));
if (phase<0)|(phase>1)
    h=warndlg('This is not a right width value for an LFO please type a value from 0 to
1','Warning!!');
end
dchandle=findobj(gcf,'Tag','EditText4');
dcoffset=str2num(get(dchandle,'String'));
if (dcoffset<-18)|(dcoffset>18)
    h=warndlg('This is not a right dcoffset value for an LFO please type a value from -
18 to 18 V','Warning!!');
end
dampinhandle=findobj(gcf,'Tag','EditText5');
damping=str2num(get(dampinhandle,'String'));
rectifhandle=findobj(gcf,'Tag','Checkbox3');
rectif=get(rectifhandle,'Value');
h=waitbar(0,'Performing Calculus on LFO... Please Wait');
for i=1:100,t=0:1/samplerate:5;
    waitbar(i/100);
end
close(h);
if wallsupply==0
    x=dcoffset+amp*exp(-(t*damping)).*sawtooth(2*pi*rate*t,phase);
elseif wallsupply==1
    x=dcoffset+amp*exp(-(t*damping)).*sawtooth(2*pi*rate*t,phase)+0.1*sin(2*pi*50*t);
end
if rectif==1
    y=abs(x);
elseif rectif==0
    y=x;
end
switch(popup2)
case 1
    i=int8(y);

```

```

case 2
    i=int16(y);
case 3
    i=int32(y);
case 4
    i=single(y);
case 5
    i=double(y);
end
plot(t,i,'EraseMode','xor');
end
%%LFW Pseudo-Randomica
if popup==7
rsstephandle=findobj(gcf,'Tag','Checkbox1');
rsstep=get(rsstephandle,'Value');
if rsstep==0
    samplerate=100;
elseif rsstep==1
    samplerate=50;
end
wallsupplyhandle=findobj(gcf,'Tag','Checkbox2');
wallsupply=get(wallsupplyhandle,'Value');
ratehandle=findobj(gcf,'Tag','EditText1');
rate=str2num(get(ratehandle,'String'));
if (rate<0.3)|(rate>16)
    h=warndlg('This is not a right rate value for an LFO please type a value from 0.3
to 16 Hz','Warning!!');
end
amphandle=findobj(gcf,'Tag','EditText2');
amp=str2num(get(amphandle,'String'));
if (amp<0)|(amp>18)
    h=warndlg('This is not a right amplitude value for an LFO please type a value from
0 to 18 V','Warning!!');
end
phasehandle=findobj(gcf,'Tag','EditText3');
phase=(str2num(get(phasehandle,'String'))*2*pi)/360;
if (phase<0)|(phase>359)
    h=warndlg('This is not a right phase value for an LFO please type a value from 0 to
359 degrees','Warning!!');
end
dchandle=findobj(gcf,'Tag','EditText4');
dcoffset=str2num(get(dchandle,'String'));
if (dcoffset<-18)|(dcoffset>18)
    h=warndlg('This is not a right dcoffset value for an LFO please type a value from -
18 to 18 V','Warning!!');
end
dampinghandle=findobj(gcf,'Tag','EditText5');

```

```

damping=str2num(get(dampinhandle,'String'));
rectifhandle=findobj(gcf,'Tag','Checkbox3');
rectif=get(rectifhandle,'Value');
h=waitbar(0,'Performing Calculus on LFO... Please Wait');
for i=1:100,t=0:1/samplerate:5;
    waitbar(i/100);
end
close(h);
if wallsupply==0
    x=dcoffset+amp*exp(-
(t*damping)).*sin(2*pi*rand(1)*10*rate*t+phase)+sin(4*pi*rand(1)*10*rate*t+phase)+sin(8*pi
i*rand(1)*10*rate*t+phase)+sin(16*pi*rand(1)*10*rate*t+phase)+sin(32*pi*rand(1)*10*rate*t
+phase)+sin(64*pi*rand(1)*10*rate*t+phase);
elseif wallsupply==1
    x=dcoffset+amp*exp(-
(t*damping)).*sin(2*pi*rand(1)*10*rate*t+phase)+sin(4*pi*rand(1)*10*rate*t+phase)+sin(8*pi
i*rand(1)*10*rate*t+phase)+sin(16*pi*rand(1)*10*rate*t+phase)+sin(32*pi*rand(1)*10*rate*t
+phase)+sin(64*pi*rand(1)*10*rate*t+phase)+0.1*sin(2*pi*50*t+phase);
end
if rectif==1
    y=abs(x);
elseif rectif==0
    y=x;
end
switch(popup2)
case 1
    i=int8(y);
case 2
    i=int16(y);
case 3
    i=int32(y);
case 4
    i=single(y);
case 5
    i=double(y);
end
plot(t,i,'EraseMode','xor');
end

```

Questa parte dell'algorithmo invece, è relativa al salvataggio delle informazioni della LFW su un file proprietario con estensione .lfo, ed ha un funzionamento piuttosto intuitivo. Si catturano le informazioni su stringhe che verranno successivamente inviate al file e salvate con l'istruzione "fprintf". Il 'case credits' visualizza le informazioni relative all'autore del progetto e il 'case close' include una semplice operazione di catching figure, per poi chiuderla.

```

case 'save'
    [newfile,newpath]=uinputfile('*.lfo','Save LFO type');
    newfile1=[newfile];
    newpath1=[newpath];
    fid=fopen(strcat(newpath1,newfile1,'.lfo'),'wb');
    switch(popup)
    case 1
        ty='sin';
    case 2
        ty='cos';
    case 3
        ty='tri';
    case 4
        ty='squ';
    case 5
        ty='exp';
    case 6
        ty='saw';
    case 7
        ty='ran';
    end
    rsst=num2str(rsstep);
    wall=num2str(wallsupply);
    ra=num2str(rate);
    am=num2str(amp);
    ph=num2str(phase);
    dc=num2str(dcoffset);
    damp=num2str(damping);
    rec=num2str(rectif);
    h=waitbar(0,'Saving LFW Data on a file... Please Wait');
    for i=1:100,
        waitbar(i/100);
    end
    close(h);
    fprintf(fid,'%s\n%s\n%s\n%s\n%s\n%s\n%s\n%s\n%s\n',ty,rsst,wall,ra,am,ph,dc,damp,rec);
    status=fclose(fid);
    if status===-1
        errordlg('An error has occurred during the file writing -> Operation
Aborted','Error');
    end
case 'credits'
    h=helpdlg('This is the LFO - Maker inside of Nova - Mod Tools enviroment built by
Zambelli Cristian','Information');
case 'close'
    close(gcf);

```

end

La scelta di utilizzare un algoritmo di questo tipo, prevede un adattamento perfetto della LFW alla forma d'onda del segnale audio da trattare successivamente e questo permette quindi una migliore qualità di processing che va però a scapito della capacità su disco e di memoria richiesta. Oltre a questo file, Matlab genera automaticamente un file .m e un file .mat che contengono le informazioni relative alla posizione dei controlli e al setting delle loro proprietà. I sorgenti del codice sono disponibili sul CD-ROM allegato alla tesi nella directory ./Codice/Matlab con estensione .m. Per eseguire correttamente lo script, si consiglia di lanciare il Nova – Mod Tools o in alternativa, per avere una esecuzione in stand – alone ci si deve portare con il path browser di Matlab nella cartella dove risiede il codice e quindi eseguire il file lfomaker.m. Per quanto riguarda invece, la libreria LFO Glory of the Past & Present, si è cercato tramite LFO – Maker, di effettuare un' operazione di modelling di alcuni degli LFO più famosi sul mercato e di alcuni oggetti che sono entrati nella storia della musica. Ecco cosa è stato modellizzato:

- Roland – Boss CE-1 LFO Internal Chorus: costituito da un suono molto morbido formato da una sinusoide a 4 Hz ed un'ampiezza della LFW di 9 V di picco con un leggero DC offset. Quest'unità è entrata nella storia come il primo sistema di chorus. Il file del modello è Bossce1.lfo
- Modular Moog Kit Emerson Lead Saw: il tipico suono del tastierista del trio progressive rock degli anni '70 Kit Emerson nella fase di solo con il suo sintetizzatore Modular Moog che si può ammirare nel brano "Lucky Man". Il modello prevede una LFW triangolare a 12 V di picco con una frequenza di 2 Hz, completamente rettificata. Il file del modello è Emerson.lfo
- Voodoo Labs. Pulsar Tremolo: utilizza una simulazione di un sistema Regular Stair Stepped con una LFW quadra a 9 V di picco e una frequenza di 14 Hz per ottenere un suono che "inciampa". Il file del modello è Vodootrem.lfo
- Tony Banks sound: questo modello ricrea la LFW utilizzata nell'LFO interno del sintetizzatore Modular Moog, del tastierista Tony Banks del gruppo leggenda del prog-rock "Genesis". Il modello è una LFW esponenziale – sinusoidale a 0.6 Hz con un'ampiezza di picco di 5 V. Il file del modello è Tbanks.lfo

Capitolo 3

Progetto del sistema di acquisizione del segnale audio e metodi per l'analisi

3.1 Prefazione del sistema

In questa sezione della trattazione, andiamo a costituire il blocco principale che servirà all'environment Nova – Mod Tools, per lavorare con un segnale audio arbitrario: l'acquisizione del segnale stesso. Ovviamente questo processo dovrà essere interamente svolto nell'ambiente di sviluppo GUIDE per Matlab, in modo da garantire la compatibilità con il precedente LFO – Maker e con le altre funzionalità del software. Da esso infatti, dipenderanno le operazioni di valutazione dello spettro del segnale e gli algoritmi di esecuzione degli effetti di modulazione, i quali si baseranno fedelmente ai parametri imposti dal segnale audio. Le proprietà che contraddistinguono questi segnali sono:

- La frequenza di campionamento o sampling rate: la quale specifica la velocità (e quindi anche la precisione) con cui viene campionato (rappresentato digitalmente) il segnale audio, ed è una frequenza che può assumere secondo gli standard internazionali le seguenti opzioni principali: 8000 Hz (rappresentazione tipica di un segnale trasmesso via radio AM), 11025 Hz (qualità radio), 22050 Hz (qualità telefonica), 44100 Hz (qualità CD) e 48000 Hz (qualità DAT¹). Ovviamente non esistono delle frequenze ben definite, infatti un utente potrebbe decidere di campionare il proprio segnale a 4719 Hz, il problema è che la maggior parte delle applicazioni per la manipolazione del segnale ne permette la lettura, ma non la scrittura del segnale trattato alla stessa frequenza di campionamento. Negli ultimi anni inoltre, esiste una tecnica migliore per la rappresentazione dei segnali audio analogici in maniera digitale, che sfrutta il procedimento di upsampling² fino a 128000 Hz, ma questa riguarda tipicamente la processazione via DSP.
- Il numero di bit utilizzati per il processo di quantizzazione: trattandosi di un segnale sostanzialmente di tipo PCM, è implicito l'utilizzo di un segnale che dovrà seguire un processo di quantizzazione per la rappresentazione dei valori analogici assunti dal segnale stesso, tramite stringhe di bit. Gli standard internazionali prevedono una quantizzazione a 8

¹ Acronimo di Digital Audio Tape.

² Meglio conosciuto come il processo di sovracampionamento del segnale, con cui tramite un'operazione di filtraggio e un'operazione di zero – padding (aggiunta di campioni a valore nullo) si aumenta radicalmente la frequenza di campionamento di un segnale.

bit o 16 bit, anche se negli ultimi sviluppi su DSP, si utilizzano convertitori ADC Sigma – Delta che permettono una quantizzazione sino a 24 – bit.

Così come è stato pensato per l' LFO – Maker, si costruirà un' applicazione stand – alone GUI, che permetterà l' acquisizione del segnale, la visualizzazione dello stesso e la valutazione di alcune operazioni fondamentali quali: lo spettrogramma, le statistiche, ecc. La GUI sarà formata da un plot principale sul quale si potrà visualizzare direttamente la forma d' onda del segnale acquisito mediante uno script costruiti con Matlab e alcuni pushbutton che permetteranno o il cambio del plot nel dominio del tempo, con un plot nel dominio della frequenza o uno spettrogramma, o la visualizzazione dei dati statistici sul segnale quali la media, varianza, ecc. Questa applicazione è suddivisa principalmente in due grandi sezioni: la sezione di acquisizione del segnale tramite la scheda audio installata sul sistema che ospiterà l' applicazione o tramite un file presente su supporto di memorizzazione e la sezione di elaborazione del segnale mediante il calcolo della sua densità spettrale di potenza e del suo spettrogramma relativo. La realizzazione di questo capitolo prevederà quindi: una discussione principale sui metodi di acquisizione del segnale audio, una valutazione approfondita sui metodi di calcolo della densità spettrale di potenza del segnale esplicitando ogni algoritmo utilizzato e la discussione sui principali parametri statistici di interesse che vengono trattati con questa applicazione.

3.2 Metodi di Acquisizione del segnale audio arbitrario

3.2.1 By – File Acquisition (Wave Method)

Il primo metodo di acquisizione di un segnale audio arbitrario è senz' altro basato su un' acquisizione da file. Questo processo, sebbene sembri così relativamente semplice, nasconde delle insidie piuttosto grandi dal punto di vista computazionale e algoritmico. I file con cui la nostra applicazione andrà a trattare, sono segnali basati su una codifica di tipo PCM pura³, il che semplificherà notevolmente l' operazione di estrapolazione dei valori dei campioni del segnale, visto che si tratta di un file che contiene la descrizione della forma d' onda direttamente. La prima cosa che si può pensare è quindi, organizzare uno script in Matlab che si basi sulla funzione di libreria *wavread*⁴, estrapoli i valori dei campioni, effettui il plot e ... Nell' istante in cui abbiamo pensato a questa affermazione, non abbiamo però tenuto conto di un fattore fondamentale che sta alla base di ogni principio fisico e di ogni progetto: il tempo. È proprio il tempo che ci consente di dire che questa

³ Esistono infatti, altri tipi di codifica PCM come la ADPCM (Adaptive Delta Pulse Coded Modulation) di Microsoft e IBM.

⁴ Presente nella Signal Processing Toolbox function library di Matlab.

soluzione è una strategia non applicabile in questa situazione, perché aumenterebbe il tempo di analisi dell'oggetto principale di questa tesi (un segnale audio), facendo perdere all'utente medio di questo software, il quintuplo del normale tempo impiegato da un normale software per il sound processing. Si richiede quindi lo studio di una strategia alternativa, eventualmente anche basata su una sorta di benchmarking con altri metodi di acquisizione di questa tipologia di segnali, che permetta di velocizzare l'elaborazione complessiva. La questione quindi è questa: la funzione che Matlab utilizza per l'acquisizione di un segnale con estensione .wav con codifica PCM pura, è rallentata da una sorta di "operazione interna" che impedisce di sfruttare le potenzialità massime di un PC dotato di un microprocessore sufficientemente potente con una quantità di memoria piuttosto elevata e questa operazione è detta (dopo un approfondito debugging della funzione che utilizza a volte anche metodi poco ortodossi basati su "reverse – engineering") "normalizzazione". Un file wave che segue una codifica PCM è costituito da un numero ben definito di campioni discreti e quantizzati, cioè assumono un valore binario che si basa sul numero di bit che caratterizzano la codifica della forma d'onda (se il file segue una quantizzazione o profondità di bit a 1 byte, significa che il campione può assumere un range di valori che vanno da -128 a 127, cioè da -2^8 a (2^8-1)). Il processo della normalizzazione permette di "trasformare" un valore dal dominio dei valori che dipendono dalla quantizzazione al dominio dei valori nell'intervallo $[-1;1)$, ed è un'operazione che richiede un tempo piuttosto elevato, se si considera che la funzione di Matlab lo esegue in real-time campione per campione. Bisogna quindi utilizzare un metodo che permette di accedere al file wave in maniera veloce e con la stessa velocità bisogna cercare di estrapolare i campioni, per porli in una struttura di ricezione dei dati (una variabile per intenderci) che si utilizzerà per il plot. Quale metodo più veloce di una lettura il più "terra – terra" possibile, come una lettura basata su un accesso binario al file? L'unica controindicazione, è che una lettura binaria all'interno di un file, deve essere effettuata quando si ha la piena conoscenza di cosa si va a leggere dentro allo stesso (bit per bit), ma questo non è un problema, perché esistono degli standard internazionali che ci permettono di capire come funzionano file di questo tipo e uno di questi è il "RIFF Wave File Format" tratto dal documento "Multimedia Programming Interface and Data Specification v1.0". Questo documento ci dice che un file wave segue una struttura interna di questo tipo:

```
<Wave Format> → RIFF('WAVE', <fmt-ck>, [<fact-ck>], [<cue-ck>], [<playlist-ck>], [<assoc-data-list>], <wave data>)
```

I campi indicati solo tra i simboli di <> sono campi necessari per la lettura del file e quindi devono sempre esistere, mentre i campi compresi tra i simboli [] sono dipendenti dal tipo di file wave che si va a trattare (esistono infatti differenti tipologie di file wave che seguono codifiche diverse).

Analizziamo il primo campo, ovvero il <fmt-ck> che prende il nome di Wave Format Chunk e descrive il formato e i parametri del file che si sta acquisendo. A sua volta, il Wave Format Chunk è costituito da due sottocampi:

```
<fmt-ck> → fmt(<common fields>, <format-specific-fields>)
```

che sono definiti come il Format Common Fields e il Format Specific Fields (a noi interessano particolarmente i primi). I Format Common Fields sono i seguenti:

```
<common-fields> → struct{WORD wFormatTag; WORD wChannels;
    DWORD dwSamplesPerSec;          DWORD dwAvgBytesPerSec; WORD
                                wBlockAlign; }
```

ed assumono il seguente significato:

- WORD wFormatTag : è il numero che indica (con 16 bit), il formato del file wave. Il contenuto dei Format Specific Fields e l'intera lettura del file, dipendono strettamente da questo valore
- WORD wChannels : è il numero di canali (in 16 bit) rappresentati dalla forma d'onda. I valori possibili sono: 1 per un segnale monoaurale e 2 per un segnale stereofonico
- DWORD dwSamplesPerSec : questa quantità (espressa in 32 bit), specifica la frequenza di campionamento in campioni al secondo, con cui il file dovrebbe essere eseguito⁵
- DWORD dwAvgBytesPerSec : rappresenta il numero medio di bytes al secondo (in 32 bit), con cui la forma d'onda dovrebbe essere trasferita. Il software che esegue il playback del suono, deve avere la capacità di stimare l'ampiezza del buffer audio in base a questo dato

⁵ Per "eseguito", si intende inviato alla scheda audio che lo proporrà al mondo reale tramite i diffusori.

- WORD `wBlockAlign` : rappresenta l'allineamento che seguono i campioni della waveform (con 16 bit) in termini di bytes. È un dato che può essere usato per un corretto allineamento del buffer audio e per evitare cross – playing⁶

I Format Specific Fields invece, consistono in zero o più bytes che dipendono dal tipo di formato di file wave che si sta considerando. Il formato che a noi interessa (anche per ragioni di semplicità) è un formato cosiddetto “non proprietario” ed è il Microsoft WAVE FORMAT PCM. Questo tipo di formato prevede un unico Format Specific Fields che è: WORD `wBitsPerSample` il quale specifica il numero di bit utilizzati per il processo di quantizzazione dei campioni della forma d'onda e può assumere un valore qualsiasi che per comodità di software limiteremo a 8 e 16 bit. Perché i dati siano corretti e non ci sia errore all'interno del file, devono essere rispettate le seguenti formule:

$$wAvgBytesPerSec = wChannels \cdot wBitsPerSecond \cdot \frac{wBitsPerSample}{8} \quad (1)$$

$$wBlockAlign = \frac{wBitsPerSample}{8} \quad (2)$$

Perché la (1) e la (2) assumano un significato correttamente interpretabile da un normale PC, devono essere arrotondati al valore intero più vicino. Avendo capito come è strutturato un file Wave PCM Microsoft, possiamo vedere come sono inseriti i dati della waveform nel file. Analizziamo quindi il Data Packing for PCM Wave Files. In un file Microsoft WAVE FORMAT PCM a canale singolo (monoaurale), i campioni della forma d'onda sono memorizzati consecutivamente; per un segnale stereofonico invece, i campioni sono intervallati dallo “speaker mapping⁷”, cioè su quale canale si sta facendo riferimento. Per capirci meglio si notino questi esempi:

⁶ Fenomeno fastidioso che avviene nei sistemi di playback di file audio, che consiste nella sovrapposizione di campioni di porzioni differenti del file, dando origine ad una distorsione e ad un cambiamento del segnale in uscita rispetto a quello realmente acquisito.

⁷ Non è ben definito dagli standard, come avvenga lo speaker mapping in presenza di più di 2 canali per un file Microsoft WAVE FORMAT PCM, ma non è comunque fondamentale per la realizzazione finale del sistema di acquisizione.

Sample 1	Sample 2	Sample 3	Sample 4
Channel 0	Channel 0	Channel 0	Channel 0

Data Packing for 8-Bit Mono PCM

Sample 1		Sample 2	
Channel 0 (left)	Channel 1 (right)	Channel 0 (left)	Channel 0 (right)

Data Packing for 8-Bit Stereo PCM

Channel 0	Channel 0	Channel 0	Channel 0
low-order byte	high-order byte	low-order byte	high-order byte

Data Packing for 16-Bit Mono PCM

Sample 1			
Channel 0 (left)	Channel 0 (left)	Channel 1 (right)	Channel 1 (right)
low-order byte	high-order byte	low-order byte	high-order byte

Data Packing for 16-Bit Stereo PCM

Il formato dei campioni assume un ruolo fondamentale nell'acquisizione del file, per cui andiamo ora a studiare il Data Format of the Samples. Ogni campione è contenuto in un valore intero i e l'ampiezza di i è il più piccolo numero di bytes richiesto per contenere il valore specifico del

campione stesso. L' LSB⁸ è immagazzinato per primo (si segue la rappresentazione Little Endian⁹ di Intel) e i bits che rappresentano l'ampiezza del campione sono memorizzati nei MSB¹⁰ di i , mentre i rimanenti sono settati a zero. Per esempio, se l'ampiezza del campione (memorizzata nel campo `nBitsPerSample`) è 12 bits, allora ogni campione è salvato in un intero di 2 bytes. I quattro bits meno significativi del primo byte (LSB) sono settati a zero. Il Data Format e i valori massimi e minimi per i campioni di un Microsoft WAVE FORMAT PCM sono:

Sample Size	Data Format	Maximum Value	Minimum Value
One to eight bits	Unsigned integer	255 (0xFF)	0
Nine or more bits	Signed integer i	Largest positive value of i	Most negative value of i

For example, the maximum, minimum, and midpoint values for 8-bit and 16-bit PCM waveform data are as follows:

Format Value	Maximum Value	Minimum Value	Midpoint Value
8-bit PCM	255 (0xFF)	0	128 (0x80)
16-bit PCM (0x7FFF)	32767 (-0x8000)	-32768	0

Questi sono i campi che a noi interessano per la scrittura dell'algoritmo di acquisizione del file wave e che vengono valutati dal sistema di lettura binaria del file. L'intero algoritmo si trova alla fine di questo capitolo.

⁸ Acronimo di Least Significant Byte. È il byte di minor peso in una parola digitale.

⁹ Little Endian (usato dai processori Intel) è un metodo di immagazzinare i dati partendo dal LSB ed è il contrario del Big Endian (usato dai processori Motorola) nel quale l'LSB è memorizzato per ultimo.

¹⁰ Acronimo di Most Significant Bits. Sono i bit di maggior peso in una parola digitale.

3.2.2 By – File Acquisition (MP3 Method)

Questa parte del sistema di acquisizione del segnale, è basata sulla lettura di un file che risponde alla tipologia Motion Picture Expert Group (MPEG-1) Layer 3, meglio noto al mondo intero come mp3. Un file di questo tipo è caratterizzato da una cosiddetta codifica¹¹ a perdita, la quale sfrutta le proprietà fisiche dell'orecchio umano. Vediamo come è caratterizzato un file di questo tipo e soprattutto come è possibile svolgere l'operazione di codifica, cioè di trasformazione di un file PCM puro in un file di tipo mp3 e viceversa tramite l'operazione di decodifica. Il processo di codifica avviene seguendo lo schema della figura in basso:

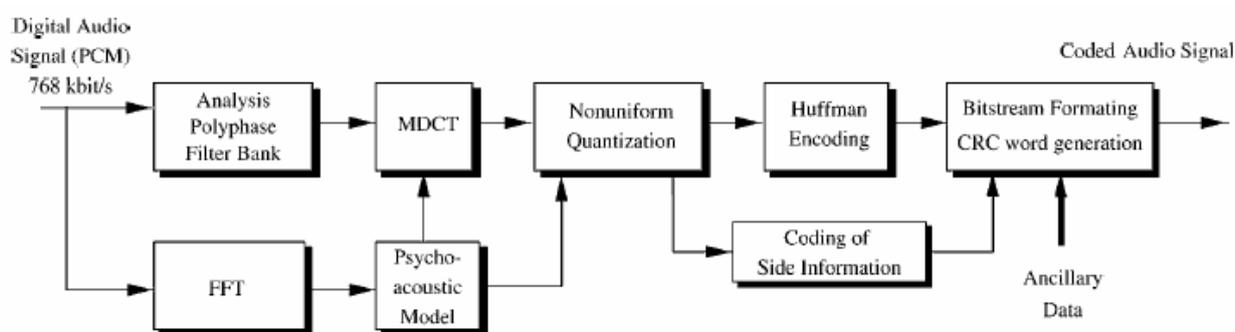


Figura 21: Diagramma a blocchi di un MPEG/Audio Encoder

supponiamo di avere un file PCM in ingresso al sistema di codifica, campionato a 48 KHz e quantizzato con 16 bit. Il codificatore audio/MPEG ha la funzione di comprimere il segnale di ingresso, in un bitstream codificato, riducendo il bitrate da 768 Kbit/s ad un valore arbitrario. I blocchi che deve attraversare il file PCM per essere codificato correttamente sono:

1. (Analysis Polyphase Filter Bank) Banco di filtri per l'analisi polifase: in questa fase di filtraggio preventivo, 1152 campioni del segnale PCM, sono filtrati da una struttura in parallelo costituita da 32 filtri passa – banda centrati ed equamente spaziati in sottobande e decimati di un fattore 32. Ogni sottobanda conterrà 36 campioni. Assumendo dei filtri puramente ideali, per il teorema di Nyquist, si garantisce che questi campioni possono essere correttamente ricostruiti, utilizzando un processo di interpolazione delle sottobande con le loro bande originarie, seguito dalla somma delle stesse. È chiaro che a causa della non

¹¹ O algoritmo di codifica, a seconda dei testi.

realizzabilità fisica di un filtro di questo tipo, verrà introdotto aliasing¹² durante il processo di decimazione. I campioni ricavati sono ancora nel dominio del tempo, anche se grazie al teorema di Parseval si può dimostrare che l'energia totale entro ogni banda, risulta uguale alla distribuzione nel dominio frequenziale. L'uscita dai filtri polifase, può essere vista quindi, come una rappresentazione spettrale grossolana dei campioni nel dominio temporale, usando solo 36 valori.

2. MDCT (Modified Discrete Cosine Transform): le 32 sottobande verranno successivamente mappate tramite la rappresentazione con la Trasformata a Coseno Discreto Modificata. Questo permetterà di aumentare la risoluzione spettrale delle 36 “righe” di frequenza per sottobanda. Prima della trasformazione, avviene però un processo di finestrazione dei campioni di sottobanda. Il procedimento di finestrazione, può utilizzare o una finestra lunga o una corta a seconda della dinamica di ogni sottobanda. Se i campioni della sottobanda presentano un comportamento piuttosto stazionario, allora verrà scelta una finestra lunga per aumentare la risoluzione spettrale della successiva MDCT. Se invece i campioni di sottobanda, presentano un comportamento con forti transienti, tre finestre corte consecutive vengono applicate ai campioni per aumentare la risoluzione temporale della successiva MDCT. Per ottenere un migliore adattamento quando sono richieste delle finestre di transizione, vengono definite due finestre dette di “Start – Stop”. Nella figura 2 si possono notare i diversi tipi di finestre applicabili ai campioni. 18 linee di frequenza, in uscita dal blocco di MDCT dopo l'applicazione di una finestra lunga o di tre consecutive, viene definito “block”, “short block” per una finestra corta e “start – end block” per le finestre di start – stop.

¹² Fenomeno che intercorre nella ricostruzione di campioni che sono stati campionati ad una frequenza teoricamente inferiore a due volte la banda del segnale campionato, ma che in realtà avviene ogni qualvolta una componente spettrale del segnale campionato, a causa delle repliche periodiche dello spettro, si sovrappone allo spettro originario.

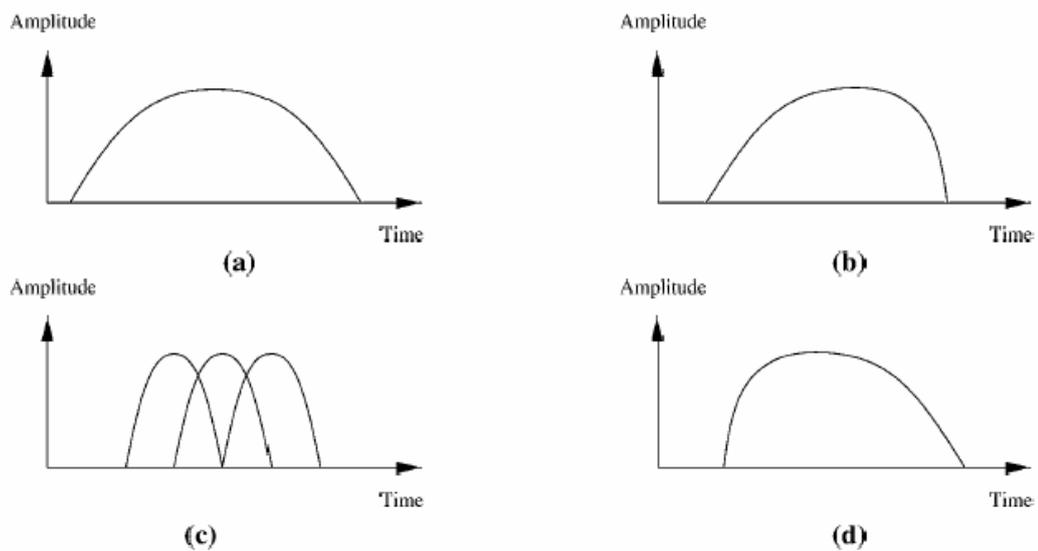


Figura 22: Illustrazione di quattro tipi di finestre: a) long window b) start window c) short window three consecutive d) stop window

La decisione di quale tipo di finestra utilizzare, spetta al modello Psico – Acustico. Questa figura mostra invece un esempio di sequenza di finestre, applicata ad una sottobanda

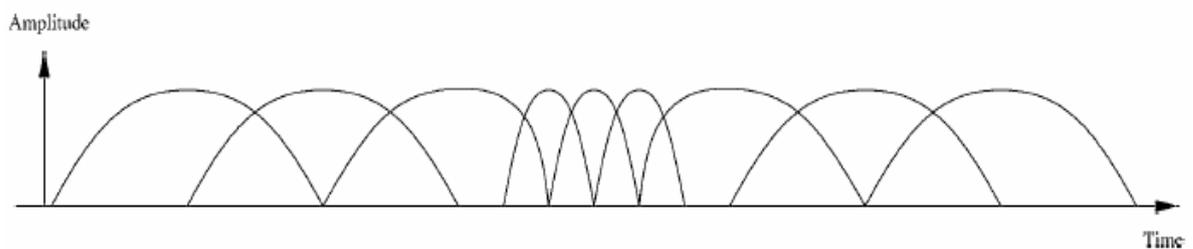


Figura 23: Esempio di sequenza di finestre

Svolgendo una MDCT su una sottobanda finestrata, con l'utilizzo di una finestra lunga, si producono 18 linee di frequenza, dovuto al 50% di overlap; con tre finestre corte consecutive, a causa del 50% di overlap, si producono 3 gruppi di 6 linee di frequenza, dove ogni gruppo dipende da differenti intervalli temporali. Avendo usato la MDCT, si produrranno 576 linee per un "granulo". La MDCT produce due granuli quando trasforma le 32 sottobande. Prima di passare però, alle linee di frequenza, viene svolta un'operazione preventiva di alias – reduction, tramite la computazione a butterfly¹³. A questo punto il segnale PCM iniziale è stato suddiviso in 32 sottobande con il segnale audio all'interno.

¹³ Processo usato comunemente nell'elaborazione numerica dei segnali per velocizzare le operazioni di computazione, facendo riferimento a campioni alterni, in modo da utilizzare una struttura di dati (grafo di flusso) che assume la struttura di una farfalla. Si usa spesso per il calcolo di FFT.

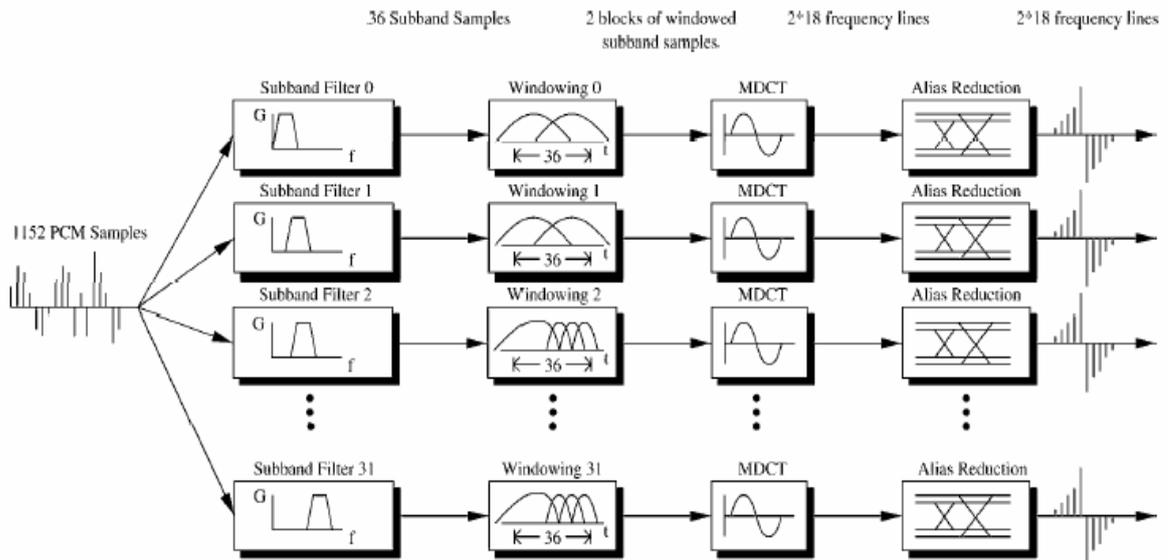


Figura 24: Sommario delle operazioni applicate ai 1152 campioni del segnale PCM

3. FFT (Fast Fourier Transform): è l'operazione concorrente all'analisi polifase, ed è svolta due volte sui 1152 campioni. Sono realizzate in entrambi i modi, le FFT a 1024 e a 256 punti per aumentare la risoluzione dello spettro e per garantire correttezza nelle informazioni anche per segnali con transienti molto ripidi. È ovvio che una rappresentazione FFT a 1024 punti non darà un'ottima risoluzione di 1152 campioni. Questo numero è stato scelto però dagli standard mondiali, in quanto è una potenza di 2 e si può dimostrare che comparato al calcolo di una DFT (Discrete Fourier Transform) richiede meno dell'1% di potenza di calcolo.
4. Psycho – Acoustic Model (Modello Psico – Acustico): questo blocco contiene il set di algoritmi necessario per la costituzione del modello psico – acustico. In sintesi, esso rappresenta come si comporta l'orecchio umano nella percezione dei suoni. È utilizzato solo in fase di codifica, perché in questo modo si può scegliere secondo il modello, quali sono le frequenze più rilevanti a livello percettivo. Il risultato del modello è utilizzato nel calcolo della MDCT e della Non – Uniform Quantization. Esso infatti dice al blocco di calcolo della MDCT, quale tipo di finestra bisogna applicare al segnale. La decisione è basata sulla misura delle differenze tra le due presenti FFT calcolate e gli spettri precedenti al calcolo. Se una certa differenza è molto incisiva il modello psico – acustico invierà un richiesta di una finestra corta di transizione. Quando le differenze scompaiono invece, viene fatta richiesta di una finestra lunga. Questo modello controlla anche la quantizzazione non uniforme, gestendo le informazioni su come bisogna quantizzare una certa linea di frequenza. La

quantizzazione delle linee è limitata e adattata alla percezione dell'orecchio umano. Alcuni esperimenti euristici hanno dimostrato che l'orecchio umano è particolarmente poco selettivo in 24 bande, chiamate bande critiche¹⁴. Quando una componente tonale è dominante in un segnale audio, le frequenze associate alle bande critiche non sono ben distinte e percepite. La componente dominante introduce quindi una sorta di soglia di mascheramento al di sotto della quale le bande critiche sono attenuate. Questo effetto permette una quantizzazione più grossolana attorno a queste frequenze, senza introdurre una degradazione udibile. Nello standard MPEG/audio, queste bande sono scalate con opportuni fattori. Il modello psico – acustico utilizza la FFT per capire quali sono le componenti tonali preponderanti e quando presenti viene chiamato in causa il processo di mascheramento di soglia. In base a questo fattore viene limitato superiormente il livello di quantizzazione richiesto per le bande scalate. Due modelli psico – acustici, applicati un encoder per Layer 3 sono descritti nel documento *ISO/IEC 11172 – 3, 1993*, anche se vanno bene modelli ben più semplici.

5. Nonuniform quantization (Quantizzazione non – uniforme): la non linearità è introdotta semplicemente elevando il valore di ogni campione per una potenza di valore $\frac{3}{4}$. Per ridurre ulteriormente il rumore di quantizzazione, è svolto lo scaling di ogni linea di frequenza secondo lo scalefactor di ogni banda. L'uscita di questo blocco è l'insieme dei valori delle linee di frequenza quantizzati e scalati per ogni scalefactor. Lo scaling e il raggruppamento delle linee di frequenza è svolto in accordo con il modello psico – acustico.

6. Huffman Encoding (Codifica di Huffman): in questo blocco viene svolta la codifica di Huffman delle linee di frequenza, usando l'algoritmo di codifica di basato su 32 tabelle di Huffman statiche. Questo schema di codifica è uno dei principali motivi del successo del Layer 3 che permette di avere poca perdita audio anche a bitrates piuttosto ridotti. Lo schema prevede una codifica senza perdita riducendo il numero di dati da trasmettere senza degradare la qualità. La riduzione dei dati è ottenuta sfruttando il concetto di entropia, ovvero di similitudine presente nella maggior parte dei dati. Nella codifica di Huffman l'entropia è modellizzata statisticamente distribuita su numero predefinito di valori. Per la statistica dei dati, una tabella di sostituzione stabilita a priori è associata a tutti i dati. In questa tabella, i valori che hanno una alta probabilità di essere presenti nel segnale sono associati con parole a codice corto mentre i dati a bassa probabilità di presenza sono

¹⁴ Esperimento e studio di Furui, 1989.

associati a parole lunghe. Codificare un blocco di dati in questo modo, prevede di ridurre il numero di bit per il processo intero, in maniera consistente. Per aumentare ulteriormente il rate di compressione, entrambi i granuli sono suddivisi in partizioni più piccole e associati a coppie o in quadruple. Il partizionamento permette inoltre di usare diverse tabelle di Huffman per avere diversi schemi di compressione.

7. Coding of Side Information: per abilitare il decoder a riprodurre il segnale audio, devono essere forniti tutti i parametri necessari dall'encoder. La codifica delle informazioni stabilisce ad esempio i boundaries dei blocchi di dati, passi di quantizzazione, quali finestre usare per la MDCT, ecc.
8. Bitstream Formatting CRC Word Generation: in questo blocco, la codifica di Huffman, le linee di frequenza e le informazioni di codifica sono raggruppate insieme, per formare il cosiddetto bitstream, il quale verrà partizionato in frames, ognuno rappresentante i 1152 campioni PCM. Per il controllo della correttezza dei dati nel decoder si introduce il CRC (Cyclic Redundant Check o Controllo di Ridondanza Ciclico basato su polinomi generatori ciclici).
9. Ancillary data: rappresentano le informazioni ausiliarie dette anche ID3 Tag per l'identificazione dell'artista del brano, la categoria di musica, ecc.

La procedura appena descritta, è valida per un segnale monoaurale, mentre per un segnale stereofonico si deve introdurre una particolare codifica detta MS (Middle Side) e anche una codifica di Intensità Stereo¹⁵. La decodifica di un file MP3 invece, segue il seguente schema di flusso che risulta autoesplicativo:

¹⁵ Questa codifica utilizza un canale unico per la trasmissione, consentendo un notevole risparmio di dati e aumentando in maniera drastica, la velocità di codifica.

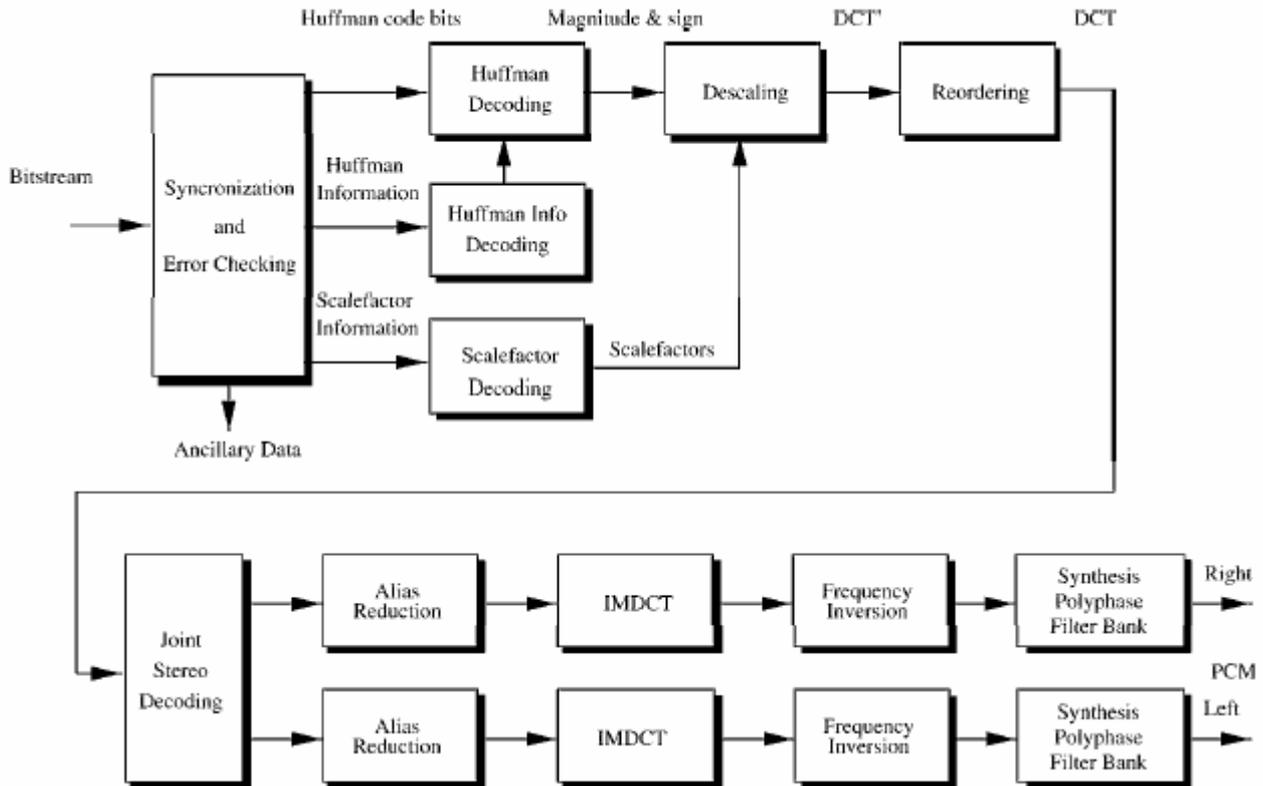


Figura 25: Schema di decodifica

Il nostro sistema userà un encoder/decoder esterno chiamato LAME, distribuito in freeware da sourceforge.net e chiamato da Matlab per il processo di acquisizione del segnale. LAME include un eseguibile e una DLL library che implementa il sistema di encoding e decoding, con molteplici possibilità di bitrate selection e di metodi di filtraggio.

3.2.3 Real – Time Acquisition

L'acquisizione in tempo reale, di un segnale audio arbitrario è il punto chiave (ma anche il più semplice) di un sistema di processing e deve essere implementato nella maniera più corretta possibile e soprattutto senza problemi che potrebbero corrompere l'intero processo di acquisizione. Per far sì che la nostra GUI associata al Signal Acquisition System, possa andare ad eseguire l'operazione di rilevazione di un segnale audio in ingresso al PC (ovviamente tramite la scheda audio) su cui risiederà il software e per far sì che tutto questo sia sviluppato con un codice Matlab, in grado di garantire l'interoperability tra le varie componenti dell'environment Nova – Mod Tools, si utilizza la libreria di funzioni DAQ¹⁶ Toolbox, presente nella maggior parte delle distribuzioni dei prodotti di MathWorks Inc. Per poter leggere dei dati in ingresso alla scheda audio del PC, sono

¹⁶ Acronimo di Data AcQuisition. È un acronimo utilizzato frequentemente nel campo dei sistemi di acquisizione dati e strumentazione virtuale.

necessari tre componenti chiave: un “data source”, un “data sensor” e un “data sink”. In questo caso il “data source” sarà il segnale audio arbitrario che arriva alla scheda audio del PC da uno strumento musicale, la voce, ecc.; il “data sensor” sarà il microfono del PC o il circuito di acquisizione e il “data sink” sarà il canale dell’oggetto di ingresso analogico del sistema di acquisizione di dati. Per prima cosa bisognerà quindi creare un canale e stabilire che esso appartiene ad una categoria di Analog Input object. Dopo questa operazione, bisogna stabilire dove dovranno essere immagazzinate le informazioni ricevute dall’acquisizione (disco o memoria), in che modo queste operazioni vanno salvate sul supporto (overwrite del vecchio segnale acquisito o appended cioè messo in coda al vecchio), e il nome del supporto di memorizzazione qualora si decida di effettuare un salvataggio delle informazioni su disco. Un’altra informazione di vitale importanza, da specificare per l’acquisizione dei dati, è il trigger. Un trigger è definito come un evento che permette al sistema di cominciare a immagazzinare i dati dell’acquisizione sul supporto di memorizzazione. Un ingresso analogico può acquisire i dati da un trigger immediato, un trigger manuale o trigger di tipo software. Il tipo di trigger è stabilito dalla proprietà dell’oggetto Analog Input “*TriggerType*”, che per default è uguale a *Immediate*. La tipologia di trigger immediato, che è quella che interessa a noi, inizia l’acquisizione del segnale non appena arriva il segnale di start e colleziona un numero di campioni che viene definito come *SamplesPerTrigger*. Molto spesso però, nelle applicazioni di acquisizione non c’è un tempo vero e proprio per l’acquisizione, per questo si preferisce effettuare una operazione di “*Trigger Repeat*”, cioè il trigger che dà il via all’acquisizione, viene ripetuto in maniera indefinita¹⁷. Dopo la definizione del Trigger è necessario specificare la frequenza di campionamento, ovvero la frequenza con cui dovranno essere prelevati i campioni dalla scheda audio. Questo valore lo imponiamo secondo gli standard internazionali alla scelta di valori tra 8000, 22050 e 44100 Hz. Ora si può dare il via all’acquisizione del segnale, il quale, una volta che viene completamente immagazzinato sul supporto di memorizzazione, può essere o salvato su disco seguendo lo standard Microsoft PCM WAVE FORMAT o codificato e salvato secondo lo standard Motion Picture Expert Group (MPEG – 1) Layer 3, tramite l’applicativo LAME. L’intero processo di acquisizione di un segnale audio arbitrario e i metodi con cui questo segnale verrà trattato e memorizzato, verranno spiegati meglio nel paragrafo 3.4, chiarendo i parametri fondamentali di campionamento e di codifica del segnale, qualora venga salvato in formato MP3.

¹⁷ L’operazione che si compie è sostanzialmente di limitazione dell’estremo superiore relativo al tempo di acquisizione, in genere stimato in base alla capacità computazionale della macchina. Ad esempio, il programma “Registratore di Suoni”, presente nella maggior parte dei sistemi operativi di Microsoft, impone un limite massimo di acquisizione a 60 secondi.

3.2.4 Speed Benchmark

Per testare il nostro sistema di acquisizione di segnali audio, si è voluto eseguire una sorta di benchmark con altre applicazioni e con altri metodi di acquisizione dei segnali, non solo per cercare di capire come funzionano i diversi sistemi di acquisizione, ma anche per dimostrare come in alcuni casi sia preferibile una strategia rispetto ad un'altra. I test sono stati eseguiti tutti con la stessa macchina: un Notebook Acer[®] Aspire 1203XC con un processore Intel[®] Celeron[®] da 1.3 GHz, dotato di 256 MBytes di SDRAM, un disco fisso Ultra ATA/100 e una scheda audio Windows Compatible VIA WAVE Technologies[®]. Sul PC sono presenti e installati correttamente i seguenti software: Matlab v5.3.0.10183 (R11) PCWIN con add – on per MP3 acquisition e Syntrillum Cool Edit Pro 2.0 evaluation – version.

1. Test di velocità per le funzioni di acquisizione da file Microsoft PCM WAVE FORMAT

	Signal Acquisition System (funzione di <i>wave acquire</i>)	Funzione built – in di Matlab <i>wavread.m</i>	Acquisizione con software Cool Edit Pro 2.0 di Syntrillum Software Corp.
Wave file Monaurale da 4.5 sec campionato a 44100 Hz con 8 bit di quantizzazione	2.05 sec compreso il plot time con lo scaling dei valori con le unità di misura	14.91 sec compreso il plot time, ma senza lo scaling dei valori con le unità di misura	0.78 sec compreso il plot time + 1.5 sec per il precaricamento in memoria con la generazione di un file = 2.28 sec
Wave file Stereofonico da 2 sec campionato a 22050 Hz con 16 bit di quantizzazione	1.59 sec compreso il plot time con lo scaling delle unità di misura	8.41 sec compreso il plot time, ma senza lo scaling dei valori con le unità di misura	1.78 sec compreso il plot time e il tempo di precaricamento in memoria
Wave file Stereofonico da 62 sec campionato a 44100 Hz con 16 bit di quantizzazione (condizione massima)	67.22 sec compreso il plot time con lo scaling delle unità di misura	407.60 sec per l'acquisizione del segnale + 14.81 sec per il plot time senza lo scaling delle unità di misura = 422.41 sec	6.72 sec compreso il plot time e il tempo di precaricamento in memoria

Questo test dimostra come il sistema progettato per questa tesi, sia di prestazioni elevate e comparabili ai migliori software di elaborazione per quanto riguarda i file di lunghezza ridotta, ma si trovi nettamente spiazzato per quanto riguarda i file a lunga durata. Sotto alcuni punti di vista

questo non può essere considerata una limitazione grave, dal momento che la maggior parte dei sound engineer che lavora con il processing dei segnali audio, per ottenere un risultato professionale, preferisce lavorare su file piuttosto corti (in maniera ragionevole), in modo anche da non sovraccaricare il sistema di elaborazione dei file, riducendo notevolmente la probabilità di errore durante la fase di acquisizione.

2. Test di velocità per le funzioni di decodifica di file Motion Picture Expert Group (MPEG – Audio1) Layer 3

	Signal Acquisition System (funzione di MP3 Acquire con decoder LAME)	Funzione <i>mp3read.m</i> vista come add – on dei Toolbox di Matlab per Signal Processing	Acquisizione con software Cool Edit Pro 2.0 di Syntrillium Software Corp.
MP3 file da 5 sec con bitrate VBR ¹⁸ medio da 60 Kbps monoaurale	1.56 sec compreso il tempo di decodifica e il plot time con lo scaling delle unità di misura	23.11 sec compreso il tempo di decodifica e il plot time senza lo scaling delle unità di misura (Errori introdotti in fase di decodifica sui campioni)	2.53 sec compreso il tempo di decodifica, il tempo di precaricamento e il plot time
MP3 file da 4 sec con bitrate VBR medio da 81 Kbps monoaurale	2.09 sec compreso il tempo di decodifica e il plot time con lo scaling delle unità di misura	32.47 sec compreso il tempo di decodifica e il plot time senza lo scaling delle unità di misura (Glitches introdotti nel plot, ma non presenti in origine)	1.66 sec compreso il tempo di decodifica, il tempo di precaricamento e il plot time
MP3 file da 62 sec con bitrate CBR 192 Kbps stereofonico (condizione massima)	65.84 sec compreso il tempo di decodifica e il plot time con lo scaling delle unità di misura	498.03 sec per il tempo di decodifica + 14.17 sec per il plot time senza lo scaling delle unità di misura = 512.20 sec	14.43 sec compreso il tempo di decodifica, il tempo di precaricamento e il plot time

Questo test invece dimostra come il Signal Acquisition System riesca ad eccellere nel tempo di decodifica di un file MP3 (tempo praticamente assente) e anche se Cool Edit Pro sia migliore rispetto alla tempistica, introduce comunque un tempo di decodifica piuttosto alto. Risultati da non commentare per le funzioni built – in di Matlab.

¹⁸ Acronimo di Variable Bit Rate. È il metodo di codifica di un file PCM che utilizza un bitrate variabile nel bitstream creato e si contrappone al sistema CBR o Constant Bit Rate.

3. Test di velocità stand – alone Motion Picture Expert Group (MPEG – Audio1) per decoder

	LAME Encoder/Decoder v3.96.1 available on http://lame.sourceforge.net/	MPG123 v0.59s-mh2 written by Michael Hipp and ported on Win32 v0.01 by Dmitry Rahalov
MP3 file da 5 sec con bitrate VBR ¹⁹ medio da 60 Kbps monoaurale	0.13 sec	0.16 sec
MP3 file da 4 sec con bitrate VBR medio da 81 Kbps monoaurale	0.51 sec	0.49 sec
MP3 file da 62 sec con bitrate CBR 192 Kbps stereofonico (condizione massima)	3.22 sec	6.49 sec

È chiaro che il decoder LAME ha la meglio su MPG123, dal momento che MPG123 non è un vero e proprio decoder di MP3, infatti questo applicativo è maggiormente orientato verso le funzionalità di stand – alone audio player in ambiente prompt dei comandi; al contrario, LAME è uno dei tools che offre la maggiore potenzialità di decodifica di MP3, ma anche di estrema versatilità nel processo di codifica degli stessi, derivanti da file PCM, inoltre esso è abilitato al supporto di tecnologie MPEG2 – Audio e MPEG3 – Audio.

3.3 Metodi di elaborazione sul segnale audio acquisito

3.3.1 Calcolo della PSD (Power Spectral Density)

Uno dei punti principali su cui permette di agire il nostro sistema di acquisizione dei dati è il calcolo della densità spettrale di potenza del segnale acquisito. Questo parametro è infatti fondamentale per capire l'andamento nel dominio della frequenza del segnale e ci permette di valutare quali sono i punti dello spettro in cui il segnale esprime più potenza di uscita. La PSD di un segnale viene calcolata con la seguente formula:

$$P_{XX} = \sum_{m=-\infty}^{\infty} \gamma_{XX}(m) e^{-j\omega m} \quad (3)$$

La (3) è un caso specializzato di calcolo della CSD tra due segnali ed ha formula:

¹⁹ Acronimo di Variable Bit Rate. È il metodo di codifica di un file PCM che utilizza un bitrate variabile nel bitstream creato e si contrappone al sistema CBR o Constant Bit Rate.

$$P_{XY} = \sum_{m=-\infty}^{\infty} \gamma_{XY}(m) e^{-j\omega m} \quad (4)$$

Esistono però vari metodi per il calcolo della PSD, ed essi vengono distinti in due particolari categorie: non parametrici e parametrici. Utilizzeremo per cui, le potenzialità di Matlab per sfruttare tutte queste tipologie a pieno e ne valutiamo le caratteristiche in termini di algoritmo e in termini di velocità, per ognuno di questi:

- PSD calcolata con il metodo di Burg (parametrico): si sfrutta la funzione *pburg* di Matlab che permette di calcolare la densità spettrale di potenza del segnale $x[n]$ contenuto nel vettore di dati che viene passato alla funzione. Questo metodo applica un modello AR (auto – regressivo) al segnale in esame, minimizzando (col metodo dei minimi quadrati) gli errori di predizione in avanti e in indietro, mentre allo stesso tempo si impone ai parametri del modello AR di soddisfare la ricorsione di Levinson – Durbin. Per lo studio del modello si utilizza la funzione *arburg* che calcola il modello adatto al segnale con la seguente formula:

$$H(z) = \frac{\sqrt{e}}{A(z)} = \frac{\sqrt{e}}{1 + a_2 z^{-1} + \dots + a_{(p+1)} z^{-p}} \quad (5)$$

Dove si stima che p sia l'ordine del sistema in cui l'uscita è pilotata da rumore bianco. Lo spettro stimato è l'ampiezza quadratica della risposta in frequenza del modello AR. La corretta scelta dell'ordine del modello indicata dalla variabile p è molto importante e noi abbiamo fissato pari a 4. Ciò significa che il modello AR è un sistema all – pass²⁰ del quarto ordine. I parametri richiesti da questa funzione sono: il numero di punti con cui si calcola la FFT²¹ (fissato a 256 punti), la frequenza di campionamento e come si deve plottare lo spettro (“intero” è in un intervallo $[0..Fs]$, mentre “metà” è $[0..Fs/2]$). Si tratta di un algoritmo piuttosto veloce ed è adatto per la maggior parte delle applicazioni, ma prevede la complessità di determinare un corretto ordine del sistema auto – regressivo e necessita un filtraggio preventivo del segnale

²⁰ Detti sistemi passa – tutto e si tratta di sistemi che presentano una caratteristica di ampiezza costante unitaria in frequenza.

²¹ Acronimo di Fast Fourier Transform. È il metodo di calcolo della trasformata di Fourier veloce, che tramite algoritmi a decimazione del tempo della Discrete Fourier Transform permette il calcolo della trasformata di Fourier con sommatorie basate su coefficienti a potenze di 2.

$$X(k+1) = \sum_{n=0}^{N-1} x(n+1) W_n^{kn} \quad x(n+1) = \frac{1}{N} \sum_{k=0}^{N-1} X(k+1) W_n^{-kn} \quad W_N = e^{-j\left(\frac{2\pi}{N}\right)}$$

dove N è il numero di campioni del segnale, X è nel dominio della frequenza e x è nel dominio del tempo.

- PSD calcolata con il metodo della covarianza (parametrico): analogamente al metodo di Burg, si utilizza la funzione *pcov* per la stima della densità spettrale di potenza del segnale $x[n]$ tramite il metodo della covarianza, il quale prevede anch'esso l'applicazione di un modello AR al segnale e tramite la formula *arcov*, permette, allo stesso modo della (5), di valutare il miglior sistema all – pass del quarto ordine da applicare al segnale. La differenza sostanziale è questo sistema parametrico effettua un calcolo della predizione dei valori, solo nella direzione avanti. È un algoritmo analogo che richiede più o meno lo stesso tempo di esecuzione del calcolo della PSD tramite il metodo di Burg
- PSD calcolata con il metodo della covarianza modificata (parametrico): metodo analogo ai due precedenti, che utilizza la funzione di calcolo *pmcov* e richiede la determinazione di un modello AR basato sul filtraggio del segnale tramite la funzione *armcov* con la formula (5). Il calcolo della predizione dei valori avviene come per il metodo di Burg. Richiede un tempo di esecuzione leggermente superiore ai precedenti, ma comunque di impercettibile durata, se riferito al calcolo di PSD per segnali di breve durata
- PSD calcolata con il metodo MTM (Multitaper Method) (non parametrico): si tratta del metodo che richiede più tempo di calcolo in assoluto e si basa sull'utilizzo di una tecnica estremamente complessa, descritta sul testo *Percival, D.B., and A.T. Walden. "Spectral Analysis for Physical Applications: Multitaper and Conventional Univariate Techniques". Cambridge: Cambridge University Press, 1993.* Il metodo in questione, utilizza finestre ortogonali chiamate "tapers", per ottenere stime spettrali piuttosto indipendenti e al termine della stima, le ricombina per il calcolo complessivo della PSD. Questa stima, permette più gradi di libertà nel calcolo di applicazioni fondamentali come il calcolo del "bias value"²² o della varianza. La maggior parte dei sistemi di stima spettrale, utilizzano una sola finestra predefinita, causando una inevitabile perdita di dati all'inizio e alla fine della serie di Fourier. Nel metodo multitaper, il fatto che si utilizzino più finestre, permette di ovviare a questa problematica e di salvare la maggior parte dei dati che verrebbero in alternativa trascurati. Il parametro principale per il calcolo di questo tipo di PSD è un prodotto tempo – larghezza di banda NW . Questo valore permette di definire la risoluzione correlata al numero di "tapers", per stimare lo spettro del segnale, ci sono infatti sempre $2NW-1$ "tapers" usati per il calcolo dello spettro. Ciò significa che aumentando il fattore prodotto, diminuisce la

²² La stima di una quantità si dice "biased" se il suo valore atteso non è uguale alla quantità da stimare.

varianza dello spettro rispetto al valore atteso, a scapito di un maggiore spectral – leakage²³ e di un biasing della quantità della stima. Questo metodo è senz'altro il più impegnativo dal punto di vista del microprocessore, in quanto si basa sull'utilizzo di sequenze sferoidali prolate discrete (conosciute anche come DPSS = Discrete Prolate Spheroidal Sequences o sequenze di Slepian). Per più di 10000 campioni, il calcolo di queste sequenze rallenta notevolmente. L'algoritmo usato nel software creato, viene creato utilizzando un adattamento di Thomson per combinazioni lineari, anziché una combinazione lineare di autovettori o di valori unitari

- PSD calcolata con il metodo MUSIC – EV: questo tipo di metodo di calcolo della PSD è basato sul sistema MUSIC – EV , il cui acronimo significa MUltiple SIngal Classification – Eigen Value. Il principio di questo sistema è definito dal metodo di Schmidt per l'autoanalisi e dal metodo degli autovettori sviluppato da Johnson. Entrambi i metodi sono stimatori di frequenza, basati su un metodo di autoanalisi che permette di scindere le informazioni di una matrice di autocorrelazione in un sottospazio di segnale o in un sottospazio di rumore. In termini più semplici, si supponga di avere un segnale corrotto da rumore bianco, per cui la sua matrice di autocorrelazione la possiamo scrivere come $R=S+W$. C'è una relazione stretta fra gli autovettori della matrice di autocorrelazione e i sottospazi di segnale e di rumore. Gli autovettori v di S sono i vettori del sottospazio di segnale. Se il sistema contiene M sinusoidi complesse e l'ordine della matrice di autocorrelazione è p , gli autovettori da v_{M+1} a v_{p+1} sono i vettori del sottospazio di rumore. Per generare la stima di frequenza, il metodo di autoanalisi calcola funzioni del sottospazio di segnale e di rumore. Entrambe le tecniche di stima dello spettro (MUSIC – EV), scelgono una funzione che teoricamente tende all'infinito a una delle frequenze delle componenti sinusoidali che costituiscono l'ingresso. Usando la tecnologia digitale, questa stima risulta in picchi molto bruschi alle frequenze di interesse e questo significa che nessuna componente dei sottospazi sarà infinita. La formula di stima del sistema MUSIC è:

$$P_{music}(f) = \frac{1}{e^H(f) \left(\sum_{k=p+1}^N v_k v_k^H \right) e(f)} = \frac{1}{\sum_{k=p+1}^N |v_k^H e(f)|^2} \quad (6)$$

dove N è l'ampiezza degli autovettori ed $e(f)$ è un vettore complesso sinusoidale:

²³ Rappresenta il fenomeno per cui, quando si calcola la DFT o una sua derivazione, per un segnale stazionario tramite procedimento di finestrazione, aumentando il numero di finestre, le ampiezze dei picchi (peak – mainlobe) risultano più elevate.

$$e(f) = |1 \cdot e(j2\pi f) \cdot e(j4\pi f) \cdot \dots \cdot e(j2\pi f \cdot (n-1))|^H \quad (7)$$

v rappresenta gli autovettori della matrice di autocorrelazione; v^k è il k -esimo autovalore; H è l'operatore di coniugazione trasposta. Gli autovettori usati nella sommatoria, corrispondono ai più piccoli autovalori che caratterizzano la divisione del sottospazio di rumore. L'espressione $v_k^H e(f)$ corrisponde ad una equivalente trasformata di Fourier. Questa forma è particolarmente utile per il calcolo della FFT, velocizzando le operazioni di somma. La variante del metodo MUSIC con lo studio degli EV pesa la somma degli autovettori della matrice di autocorrelazione con la seguente formula:

$$P_{ev}(f) = \frac{1}{\left(\sum_{k=p+1}^N |v_k^H e(f)|^2 \right) / \lambda_k} \quad (8)$$

Questa funzione sfrutta il metodo di scomposizione dei valori singolari e richiede un tempo di applicazione leggermente più elevato rispetto ai normali sistemi di calcolo della PSD²⁴

- PSD calcolata con il metodo di Welch: si tratta probabilmente dell'algoritmo più veloce per il calcolo della PSD di un segnale e il suo algoritmo lo possiamo sintetizzare in 5 semplici passaggi:
 1. Il sistema applica la finestra specificata dal vettore *window* della funzione, ad ogni sezione consecutiva del segnale $x[n]$. Nel nostro caso si utilizzerà una finestra che segue il modello di kaiser a 128 punti
 2. Trasforma ogni sezione in base al numero di punti di calcolo della FFT. Nel nostro caso si è posto il numero a 256 punti per evitare una complessità di calcolo troppo elevata
 3. Forma il periodogramma di ogni sezione effettuando lo "scaling" dell'ampiezza quadratica, per ogni sezione trasformata

²⁴ Il metodo di calcolo della PSD con il sistema MUSIC – EV, permette inoltre, un eventuale controllo della soglia dei sottospazi. Si è omesso nella realizzazione del software di acquisizione dei dati, questa particolare alternativa, anche perché richiederebbe calcoli matriciali e sull'applicazione di finestre, piuttosto notevoli. L'implementazione unica è quella che utilizza il sistema MUSIC

4. Media il periodogramma delle sezioni sovrapposte per formare $P_{xx}(f)$
5. Effettua lo “scaling” di $P_{xx}(f)$ di una quantità pari a $1/samplerate$ per formare $P_{xx}(f)/samplerate$ che è la PSD di $x[n]$. Il numero di sezioni che la funzione media è $k=fix((length(x[n])-noverlap)/length(window)-noverlap)$

Il plotting della funzione avviene seguendo una tecnica che permette di visualizzare la PSD in un intervallo di confidenza del 95%

- PSD calcolata con il metodo di Yule – Walker: è un metodo di calcolo della PSD non – parametrico, basato su un modello AR che assomiglia in tutto e per tutto ai modelli non – parametrici di Burg, covarianza e covarianza modificata. Si utilizza la risoluzione di un sistema di equazioni normali di questo tipo:

$$\begin{bmatrix} r(1) & r(2)^* & \dots & r(n)^* \\ r(2) & r(1) & \dots & r(n-1)^* \\ \dots & \dots & \dots & \dots \\ r(n) & \dots & r(2) & r(1) \end{bmatrix} \begin{bmatrix} a(2) \\ a(3) \\ \dots \\ a(n+1) \end{bmatrix} = \begin{bmatrix} -r(2) \\ -r(3) \\ \dots \\ -r(n+1) \end{bmatrix} \quad (9)$$

dove $a(\dots)$ è il vettore dei coefficienti autoregressivi, gli elementi del vettore $r(\dots)$ sono delle correlazioni e la matrice è una matrice Hermitiana e di Toeplitz positivamente definita. La densità spettrale viene successivamente stimata con la formula:

$$P_{YuleAR}(f) = \frac{1}{|a^H e(f)|^2} \quad (10)$$

dove $e(f)$ rappresenta una sinusoidale complessa e l'ordine del calcolo del sistema è 4. Si tratta di un algoritmo a media velocità, adatto per il calcolo della PSD per segnali puramente vocali

3.3.2 Calcolo dello spettrogramma

Un metodo molto utilizzato, per lo studio dei segnali audio arbitrari, che rientrano in tutto e per tutto nella categoria dei processi stocastici²⁵ (dall'acquisizione in tempo reale non sappiamo cosa ci darà in uscita), è l'utilizzo delle finestre. La finestrazione gioca infatti un ruolo fondamentale, specialmente per quanto riguarda l'eliminazione del fenomeno di Gibbs²⁶, nel calcolo della cosiddetta STFT (Short Time Fourier Transform) o spettrogramma. Lo spettrogramma è infatti il metodo più utilizzato per la valutazione di segnali audio, perché permette di valutare l'ensemble tempo – frequenza, in modo da capire come agiscono determinate componenti tonali di un segnale e a quale istante agiscono. Esso computa la trasformata di Fourier discreta e finestrata del segnale, utilizzando una finestra mobile. L'algoritmo di computazione dello spettrogramma è il seguente:

1. Si divide il segnale in sezioni sovrapposte e ad ogni sezione si applica il metodo di finestrazione scelto a priori
2. Viene computata la DFT di ogni sezione, con un numero di punti predefinito (nel nostro caso 256) per produrre una stima del contenuto frequenziale del segnale a breve termine (da qui deriva il nome STFT). Questa trasformazione crea le colonne della matrice che accoglierà il risultato della DFT. Se la lunghezza della finestra è minore del numero di punti con cui si calcola la DFT, allora la funzione esegue uno zero padding²⁷, in modo che sia possibile specificare all'algoritmo, di quanti campioni dovranno essere shiftate le finestre
3. Per segnali nel dominio dei numeri reali, lo spettrogramma viene troncato ai primi $(nfft + 1)/2$ punti se $nfft$ è dispari o ai $(nfft/2)+1$ punti se $nfft$ è pari, dove $nfft$ rappresenta il numero di punti con cui è calcolata la DFT di ogni sezione

È immediato capire, quanto sia fondamentale per la riuscita di un buon spettrogramma, utilizzare un metodo di finestrazione il più accurato possibile. Il software creato per l'acquisizione dei segnali permette di utilizzare i seguenti tipi di finestrazione per il calcolo di uno spettrogramma:

²⁵ Detti anche Processi Aleatori. Si definisce processo aleatorio, l'insieme di tutte le funzioni del tempo che costituiscono un'esperimento aleatorio, come l'acquisizione di un segnale audio. Se si definisce lo spazio campione Ω e prendiamo una generica realizzazione ω , dell'esperimento aleatorio, allora è possibile definire una variabile aleatoria $X(\omega, t)$ che permette di dare la definizione di processo aleatorio $X(t)$.

²⁶ Fenomeno di cattiva approssimazione dei transienti ripidi di un segnale, che vengono rappresentati con degli overshoot (sovraelongazioni).

²⁷ Detta anche operazione di riempimento con zeri. Viene comunemente eseguita nell'elaborazione numerica dei segnali, per eseguire l'operazione di sovracampionamento (oversampling).

- **Forme base (Base Windows):** rappresentano le finestre più semplici da applicare e prevedono una complessità computazionale quasi nulla. La più semplice di questa famiglia è la finestra rettangolare, la quale si comporta come un filtro passa – basso con cut – off delimitato dal numero di punti della finestra. Poi esistono la finestra di Bartlett e la finestra triangolare, che sono un caso particolare di applicazione della convoluzione tra due rettangoli. Queste due funzioni computano lo stesso valore, con la differenza che la finestra di Bartlett restituisce sempre una finestra con due zeri alla fine della sequenza, per cui per n dispari la sezione centrale di $bartlett(n+2)$ è uguale a $triang(n)$, mentre per n pari è la convoluzione di due sequenze rettangolari. La differenza principale sta nella trasformata di Fourier di queste funzioni: quella di Bartlett è negativa per n pari, mentre per una triangolare è sempre non negativa.
- **Finestre a coseno generalizzato (Generalized Cosine Windows):** le finestre di Blackman, Hanning, Hamming e rettangolare sono un caso di finestre a coseno generalizzato. Queste finestre sono una combinazione di sequenze sinusoidali a frequenze 0 , $2\pi/(N-1)$ e $4\pi/(N-1)$ dove N è la lunghezza della finestra. Un metodo per generarle è:

$$ind=(0:n-1)*2*pi/(n-1);$$

$$w=A-B*cos(ind)+C*cos(2*ind);$$

dove A, B e C sono costanti che si possono definire. Il concetto che sta dietro a queste finestre è che, sommando i termini individuali per formare la finestra, i picchi alle basse frequenze si combinano per decrementare il sidelobe²⁸ con l'effetto di aumentare però la larghezza del mainlobe. Le finestre di Hanning e Hamming sono degli esempi di finestre a coseno generalizzato con due termini con $A = 0.54$, $B = 0.46$ per Hamming e $A = 0.5$, $B = 0.5$ per Hanning. Il problema introdotto dal primo frammento di codice, è che ai campioni 1 e N , per quanto riguarda ad esempio Hanning, si formano degli zeri. Per eliminarli è sufficiente cambiare le frequenze delle sinusoidi in $2\pi/(N+1)$ e $4\pi/(N+1)$. La finestra di Blackman è invece un caso di finestra a tre termini con $A = 0.42$, $B = 0.5$ e $C = 0.08$.

²⁸ È il lobo secondario che si forma accanto al mainlobe seguendo un andamento tipico della funzione sinc. È dannoso averlo molto elevato perché causa spreco di energia nel sistema.

- Finestra di Kaiser (Kaiser Window): la finestra di Kaiser è un'approssimazione di finestre prolate – sferoidali, nelle quali il rapporto tra l'energia del mainlobe e l'energia del sidelobe è massimizzato. Per una finestra di Kaiser di particolare lunghezza, ciò che controlla l'altezza del sidelobe è il parametro β . Per un valore definito di questo parametro, l'altezza del sidelobe è definita rispetto alla lunghezza della finestra. Se il valore di β aumenta, l'altezza del sidelobe diminuisce, ma aumenta la larghezza del mainlobe.
- Finestre di Chebyshev (Equi – Ripple Window): la finestra di Chebyshev minimizza la larghezza del mainlobe, data una particolare altezza del sidelobe. Essa è caratterizzata da un comportamento di tipo Equi – Ripple, cioè i sidelobes hanno tutti la stessa altezza. Il problema dell'utilizzo di una finestra di questo tipo, è che ci sono dei forti spikes attorno ai valori all'inizio e al termine della sequenza.

3.4 Sviluppo del Signal Acquisition System con Matlab

Eccoci giunti al punto focale di questo capitolo: la costruzione del Signal Acquisition System, il software all'interno dell'environment, che si occuperà di svolgere tutte le funzioni appena definite lungo il corso del capitolo. L'applicazione sarà strutturata con i seguenti controlli:

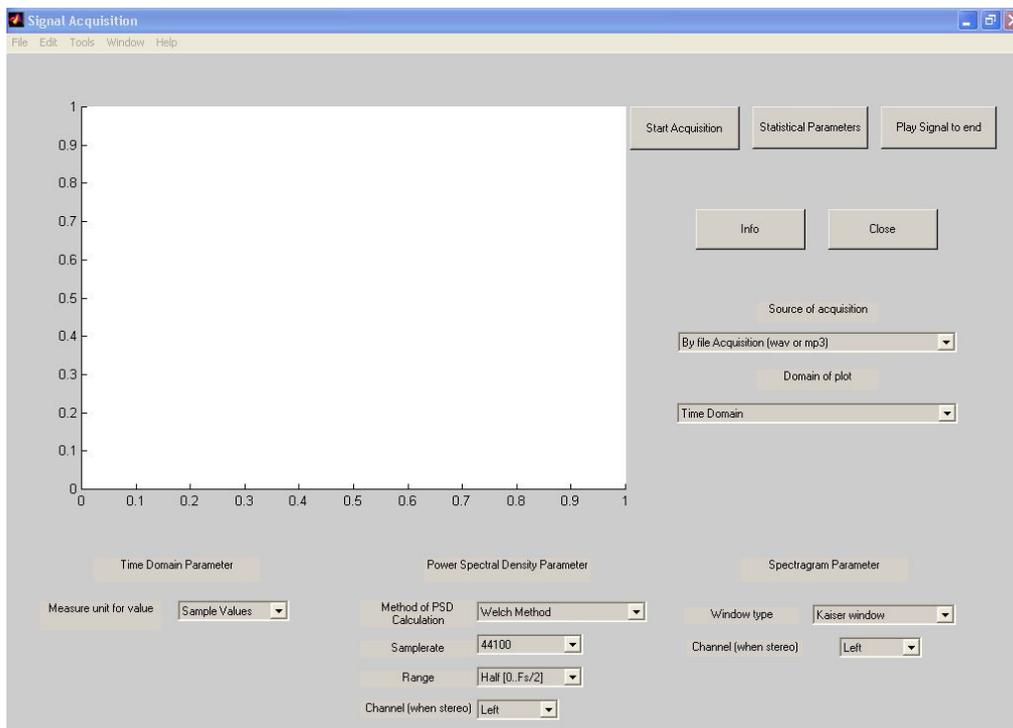


Figura 26: Immagine del Signal Acquisition System

- Start Acquisition: è il pulsante che permette di iniziare l'acquisizione del segnale audio arbitrario o via file o in real – time mode, a seconda dell'impostazione scelta
- Statistical Parameters: è il pulsante che una volta attivato, raccoglie tutte le informazioni statistiche relative al segnale acquisito: bitrate, profondità di quantizzazione, sample rate, numero di canali, valore medio del segnale, varianza²⁹ del segnale, campione a cui si trova il massimo valore, campione a cui si trova il minimo valore e mediana, che rappresenta l'offset che il segnale ha nei confronti del livello di riferimento³⁰.
- Play Signal to End: è il pulsante che consente di eseguire il playback del segnale acquisito sino al termine dello stesso. Una volta terminato, termina automaticamente anche il playback.
- Info: è il pulsante che permette di visualizzare le informazioni sul creatore del sistema e cosa serve per far funzionare questo software.
- Close: è il pulsante che chiude l'applicazione ed esegue il salvataggio dei dati acquisiti in un file che verrà utilizzato dal Nova – Mod Tools Environment per eseguire le altre applicazioni che richiedono la presenza del segnale
- Plot Zone: è l'area in cui vengono visualizzate tutte le forme d'onda, le PSD, gli spettrogrammi e cambia dinamicamente gli assi a seconda del segnale acquisito e della funzione richiesta.
- Source of Acquisition: è il popup menù che consente di scegliere la sorgente di acquisizione del segnale audio arbitrario. Controlla la maggior parte dei popup dell'applicazione.
- Domain of Plot: è il popup menù che permette di scegliere cosa si vuole plottare per il segnale acquisito. Controlla i popup relativi ai parametri per ogni dominio.

²⁹ O momento centrale del secondo ordine, calcolato con la formula: $\sigma = \frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2$, dove n è il numero di valori

contenuti nel vettore che contiene i dati del segnale.

³⁰ Il livello di riferimento in questo caso è 0 Volt, ma in alcuni casi può essere, come ad esempio per le trasmissioni di segnale audio su cavo e linea bilanciata XLR 770 mV.

- Time Domain Parameter → Measure unit for value: è il popup menù che funziona solo quando il Domain of Plot è impostato a “Time Domain” e permette di scegliere l’unità di misura per la visualizzazione dei campioni del segnale acquisito.
- Power Spectral Density Parameter → Method of PSD Calculation: è il popup menù che consente di scegliere uno tra i molteplici metodi a disposizione, per il calcolo della densità spettrale di potenza. Funziona solo quando il Domain of Plot è impostato sul valore “Power Spectral Density (Frequency Domain)”.
- Power Spectral Density Parameter → Samplerate: è il popup menù che consente di scegliere il valore con cui deve essere effettuato lo scaling della PSD. Non influisce la capacità o il calcolo ottenuto, ma modifica solamente la legenda dell’asse delle frequenze. Funziona solo quando il Domain of Plot è impostato sul valore “Power Spectral Density (Frequency Domain)”.
- Power Spectral Density Parameter → Range: è il popup menù che consente di scegliere se la PSD appena calcolata, deve essere visualizzata nel range di frequenze $[0..Fs/2]$ o $[0..Fs]$ tramite replicazione simmetrica. Funziona solo quando il Domain of Plot è impostato sul valore “Power Spectral Density (Frequency Domain)”.
- Power Spectral Density Parameter → Channel (when Stereo): è il popup menù che consente, solo quando il segnale acquisito è di tipo stereofonico, di scegliere su quale canale deve essere compiuto il calcolo della PSD. Funziona solo quando il Domain of Plot è impostato sul valore “Power Spectral Density (Frequency Domain)”.
- Spectrogram Parameter → Window Type: è il popup menù che consente di scegliere quale metodo di finestra applicare al calcolo della STFT. Funziona solo quando il Domain of Plot è impostato sul valore “Spectrogram (ensemble time/frequency Domain)”.
- Spectrogram Parameter → Channel (when Stereo): è il popup menù che ha la medesima funzione del corrispondente per i parametri della PSD. Funziona solo quando il Domain of Plot è impostato sul valore “Spectrogram (ensemble time/frequency Domain)”.

Questa applicazione è basata sullo scambio di file, in particolare, si utilizzerà un file chiamato *xchangedata.mat*, che alla chiusura dell'applicazione salverà tutte le informazioni relative al file acquisito, in modo da poterlo utilizzare con le altre applicazioni dell'environment. Per l'acquisizione di tipo real – time mode invece, si utilizza un ulteriore file chiamato *dataacquire.daq* che consente di memorizzare in maniera temporanea i dati raccolti. Analizziamo ora il codice dell'applicazione passo per passo e in particolare del function file esterno *fcnSignal.m*, che gestisce i callback, ricordando che i sorgenti li possiamo trovare nel CD-ROM allegato alla tesi, nella directory ../Codice/Matlab con estensione .m.

Questa sezione definisce la dichiarazione delle variabili globali e ricava l'handle del popup menù necessario per capire che metodo di acquisizione si vuole implementare.

```
%Function File esterno per i callback del Signal Catcher
%
%written by Zambelli Cristian
function fcnSignal(action)
global channel quantize y lefty righty samplerate count mode;
modehandle=findobj(gcf,'Tag','PopupMenu1');
mode=get(modehandle,'Value');
```

Qui inizia la swichyard programming. Se il callback è quello del pulsante Start Acquisition, allora controlla che modo di acquisizione si utilizza; se il modo è il modo 1 o real – time, richiede la frequenza di campionamento con i tre valori degli standard internazionali e il numero di bit di quantizzazione; si inizializza il dispositivo di acquisizione con un metodo basato sullo scambio di dati continui con il file *dataacquire.daq*; si dà il via all'acquisizione e si attende che passino o i 10 minuti massimi o che sia premuto il mouse o la tastiera per il termine; si richiede come si vuole salvare i dati: se via wave, si utilizza la funzione di *wavwrite*, se via MP3, allora chiama in causa l'encoder LAME con un comando del sistema operativo DOS e lo si setta per una codifica a qualità elevata (precisamente 256 Kbps) su un file temporaneo PCM *tmpfile.wav*, che verrà poi eliminato al termine della codifica, ricordando che per rientrare negli standard MPEG1 bisogna ricampionare il segnale a 44.1 KHz.

```
switch(action)
case 'acquire'
    if mode==1
        button1=questdlg('Which sample rate will be used for acquisition?','Sample
rate','8000 Hz (Low - Fidelity)','22050 Hz (Telephone Quality)','44100 Hz (CD
Quality)','22050 Hz (Telephone Quality)');
```

```

if strcmp(button1,'8000 Hz (Low - Fidelity)');
    rat=8000;
elseif strcmp(button1,'22050 Hz (Telephone Quality)');
    rat=22050;
elseif strcmp(button1,'44100 Hz (CD Quality)');
    rat=44100;
end
button2=questdlg('Which bit depth will be used for acquisition?','Bit Depth','8
bit','16 bit','16 bit');
if strcmp(button2,'8 bit');
    quan=8;
elseif strcmp(button2,'16 bit');
    quan=16;
end
ai=analoginput('winsound');
addchannel(ai,1);
ai.LogFileName = 'dataacquire.daq';
ai.LogToDiskMode = 'overwrite';
ai.LoggingMode = 'Disk&Memory';
ai.TriggerType = 'Immediate';
ai.TriggerRepeat = rat*600;
ai.SamplesPerTrigger = 8192;
ai.SampleRate = rat;
h=warndlg('Now plug the output cable of your source (e.g. Active Musical Instrument
or CD Player) to the input of your sound card. When you are ready to acquire data, push
the OK button and the acquisition will begin immediately (The acquisition will stop when
you reach the limit of 10 mins. of recording or by pushing a key or a mouse click over
the figure).','Go for Acquire');
uiwait(h);
start(ai);
k=waitforbuttonpress;
if (k==0) | (k==1)
    stop(ai);
    delete(ai);
end
y = daqread('dataacquire.daq');
l=length(y);
t=(1:l)/rat;
plot(t,y);
title('Acquired Signal');
ylabel('Normalized values');
xlabel('Seconds');
legend('Monaural Signal');
button3=questdlg('Would you like to save this file for further processing?','Save
file...','Microsoft PCM WAVE FORMAT','Motion Picture Expert Group (MPEG-1) Layer
3','No','No');
if strcmp(button3,'Microsoft PCM WAVE FORMAT')

```

```

[newfile,newpath] = uiputfile('*.wav','Save as...');
newpath1=[newpath];
newfile1=[newfile];
h=waitbar(0,'Saving in Progress... Please Wait');
for i=1:100,wavwrite(y, rat, quan, strcat(newpath1,newfile1));
    waitbar(i/100);
end
close(h);
elseif strcmp(button3,'Motion Picture Expert Group (MPEG-1) Layer 3')
    [newfile,newpath] = uiputfile('*.mp3','Save as...');
    newpath1=[newpath];
    newfile1=[newfile];
    wavwrite(y, rat, quan, 'tmpfile.wav');
    string=strcat(newpath1,newfile1, '.mp3');
    h=waitbar(0,'Encoding and Saving in Progress... Please Wait');
    cmd=['lame --quiet -b 256 --resample 44.1 tmpfile.wav', ' ',string];
    dos(cmd);
    for i=1:100,
        waitbar(i/100);
    end
    close(h);
    delete('tmpfile.wav');
end
end
end

```

Questo blocco corrisponde ai metodi di acquisizione del segnale audio via file. Si fa comparire una dialog box che permette di ricavare il nome e il path del file e con alcuni accorgimenti di manipolazione delle stringhe, anche l'estensione; se l'estensione è di tipo *.wav*, controlla che il file sia di tipo Microsoft PCM WAVE FORMAT, se lo è, esegui l'algoritmo (relativo al paragrafo 3.2.1) che ricava le informazioni principali dal file (numero di canali, sample rate, numero di bit di quantizzazione) e ricava la forma d'onda per poi plottarla sul plot principale, altrimenti chiudi l'applicazione, mostrando una error box; se il file è invece di tipo MP3, svolgi la decodifica tramite il decoder LAME e carica i dati ottenuti dalla decodifica, in un file PCM temporaneo *tmpfile.wav*; calcola il plot ed elimina il file temporaneo, liberando un po' di risorse del sistema. Terminato il plot, salva su un file di scambio *xchangedata.mat* i valori di tutte le variabili globali che servono per l'environment, in modo da garantire la comunicazione con gli stadi successivi del Nova – Mod.

```

if mode==2
    [filename,filepath]=uigetfile('*.wav;*.mp3','Acquire from which file?');
    path=[filepath];
    name=[filename];
    lsr=length(name);

```

```

ext=[filename(lsr-3:lsr)];
assignin('base','path',path);
assignin('base','name',name);
if strcmp(ext, '.wav')
fid=fopen(strcat(path,name), 'rb');
stat=fseek(fid,20, 'bof');
type=fread(fid,1, 'int16');
if (type>1) | (type<1)
    h=errorDlg('This is not a correct Microsoft PCM WAVE FORMAT, please check the
file...', 'Fatal Error');
    uiwait(h);
    status=fclose(fid);
    close(gcf);
end
stat=fseek(fid,22, 'bof');
channel=fread(fid,1, 'int16');
stat=fseek(fid,24, 'bof');
samplerate=fread(fid,1, 'int32');
stat=fseek(fid,28, 'bof');
bytessec=fread(fid,1, 'int32');
stat=fseek(fid,32, 'bof');
align=fread(fid,1, 'int16');
stat=fseek(fid,34, 'bof');      quantize=fread(fid,1, 'int16');
assignin('base','channel',channel);
assignin('base','quant',quantize);
assignin('base','rate',samplerate);
stat=fseek(fid,0, 'eof');
pos=ftell(fid);
if quantize==8
    precision='uint8';
    offset=1;
elseif quantize==16
    precision='int16';
    offset=2;
end
if channel==1
    stat=fseek(fid,58, 'bof');
    h=waitbar(0, 'Acquisition in Progress... Please Wait');
    [y,count]=fread(fid,pos-59,precision);
    for i=1:100,
        waitbar(i/100);
    end
close(h);
status=fclose(fid);
t=(1:count)/samplerate);
plot(t,y);
title('Plot of the Acquired Signal');

```

```

ylabel('Sample values');
xlabel('Seconds');
legend('Monaural signal');
assignin('base','data',y);
if status==-1
    errordlg('An error has ocured during the file reading -> Operation
Aborted','Error');
end
elseif channel==2
    h=waitbar(0,'Acquisition in Progress Left Channel... Please Wait');
    stat=fseek(fid,44,'bof');
    [lefty,count]=fread(fid,(pos/4)-24,precision,offset);
    for i=1:100,
        waitbar(i/100);
    end
    close(h);
    h=waitbar(0,'Acquisition in Progress Right Channel... Please Wait');
    stat=fseek(fid,44+offset,'bof');
    [righty,count]=fread(fid,(pos/4)-24,precision,offset);
    for i=1:100,
        waitbar(i/100);
    end
    close(h);
    status=fclose(fid);
    t=(1:count)/samplerate);
    y=[lefty,righty];
    plot(t,y);
    title('Plot of the Acquired Signal');
    ylabel('Sample values');
    xlabel('Seconds');
    legend('Left Channel','Right Channel');
    assignin('base','data',y);
if status==-1
    errordlg('An error has ocured during the file reading -> Operation
Aborted','Error');
end
end
save('xchangedata');
elseif strcmp(ext,'.mp3')
string=strcat(path,name);
h=waitbar(0,'Decoding and Converting to PCM in Progress... Please Wait');
cmd=['lame --quiet --decode ',string,' ','tmpfile.wav'];
dos(cmd);
for i=1:100,
    waitbar(i/100);
end
close(h);

```

```

fid=fopen('tmpfile.wav','rb');
stat=fseek(fid,22,'bof');
channel=fread(fid,1,'int16');
stat=fseek(fid,24,'bof');
samplerate=fread(fid,1,'int32');
stat=fseek(fid,28,'bof');
bytessec=fread(fid,1,'int32');
stat=fseek(fid,32,'bof');
align=fread(fid,1,'int16');
stat=fseek(fid,34,'bof');      quantize=fread(fid,1,'int16');
assignin('base','channel',channel);
assignin('base','quant',quantize);
assignin('base','rate',samplerate);
stat=fseek(fid,0,'eof');
pos=ftell(fid);
if quantize==8
    precision='uint8';
    offset=1;
elseif quantize==16
    precision='int16';
    offset=2;
end
if channel==1
    stat=fseek(fid,58,'bof');
    h=waitbar(0,'PCM Reading in Progress... Please Wait');
    [y,count]=fread(fid,pos-59,precision);
    for i=1:100,
        waitbar(i/100);
    end
    close(h);
    status=fclose(fid);
    t=(1:count)/samplerate;
    plot(t,y);
    title('Plot of the Acquired Signal');
    ylabel('Sample values');
    xlabel('Seconds');
    legend('Monaural signal');
    assignin('base','data',y);
    if status==-1
        errordlg('An error has occurred during the file reading -> Operation
Aborted','Error');
    end
elseif channel==2
    h=waitbar(0,'PCM Reading in Progress Left Channel... Please Wait');
    stat=fseek(fid,44,'bof');
    [lefty,count]=fread(fid,(pos/4)-24,precision,offset);
    for i=1:100,

```

```

        waitbar(i/100);
    end
    close(h);
    h=waitbar(0, 'PCM Reading in Progress Right Channel... Please Wait');
    stat=fseek(fid,44+offset, 'bof');
    [righty, count]=fread(fid, (pos/4)-24, precision, offset);
    for i=1:100,
        waitbar(i/100);
    end
    close(h);
    status=fclose(fid);
    t=(1:count)/samplerate);
    y=[lefty, righty];
    plot(t, y);
    title('Plot of the Acquired Signal');
    ylabel('Sample values');
    xlabel('Seconds');
    legend('Left Channel', 'Right Channel');
    assignin('base', 'data', y);
    if status==-1
        errordlg('An error has occurred during the file reading -> Operation
Aborted', 'Error');
    end
    end
    delete('tmpfile.wav');
    save('xchangedata');
else
    status=fclose(fid);
    h=errordlg('This is not a right file for elaboration!!!', 'Fatal Error');
    uiwait(h);
    close(gcf);
end
end
end

```

Questo case è associato al callback del popup menù, che serve per il cambiamento del dominio di visualizzazione del plot. Si ricava il valore del popup menù; se il valore è uguale a 1, significa che l'utente vuole lavorare nel dominio del tempo, per cui plotta il segnale nella forma che sull'asse delle x pone i secondi e sull'asse delle y, i valori dei campioni; se il valore è uguale a 2, significa che l'utente vuole utilizzare il dominio frequenziale con il calcolo dello spettrogramma, per cui calcolo la PSD con il metodo di Welch, con un intervallo di confidenza dello 0.95 e la plotto; se il segnale è a due canali controlla il valore del popup menù che stabilisce quale canale deve essere plottato per la PSD; se il valore è uguale 3 invece, allora l'utente vuole effettuare il calcolo dello

spetrogramma, che verrà calcolato con una finestrazione di kaiser a 128 punti e anche qui se il segnale è stereofonico, si controlla quale canale si vuole analizzare.

```

case 'domainchange'
    domainhandle=findobj(gcf,'Tag','PopupMenu2');
    domain=get(domainhandle,'Value');
    if domain==1
        if channel==1
            t=(1:count)/samplerate);
            plot(t,y);
            title('Plot of the Acquired Signal');
            ylabel('Sample values');
            xlabel('Seconds');
            legend('Monaural signal');
        elseif channel==2
            t=(1:count)/samplerate);
            plot(t,y);
            title('Plot of the Acquired Signal');
            ylabel('Sample values');
            xlabel('Seconds');
            legend('Left Channel','Right Channel');
        end
        h=waitbar(0,'Replotting in domain of time...');
        for i=1:100,
            waitbar(i/100);
        end
        close(h);
    elseif domain==2
        h=waitbar(0,'Awaiting for calculus of Power Spectral Density...');
        for i=1:100,
            waitbar(i/100);
        end
        close(h);
        if channel==1
            pwelch(y,256,samplerate,kaiser(128,5),0,0.95,'half');
            legend('Monaural Plot','95 Percentual confidence interval','95 Percentual
confidence interval');
        elseif channel==2
            popup6handle=findobj(gcf,'Tag','PopupMenu6');
            popup6=get(popup6handle,'Value');
            if popup6==1
                pwelch(lefty,256,samplerate,kaiser(128,5),0,0.95,'half');
                legend('Left Channel Plot','95 Percentual confidence interval','95 Percentual
confidence interval');
            elseif popup6==2

```

```

        legend('Right Channel Plot','95 Percentual confidence interval','95
Percentual confidence interval');
        pwelch(righty,256,samplerate,kaiser(128,5),0,0.95,'half');
    end
end
elseif domain==3
    if channel==1
        h=waitbar(0,'Awaiting for calculus of Spectrogram...');
        for i=1:100,
            waitbar(i/100);
        end
        close(h);
        specgram(y,256,samplerate,kaiser(128,5));
        title('Monaural Spectrogram');
        colormap(jet);
    elseif channel==2
        popup20handle=findobj(gcff,'Tag','PopupMenu20');
        popup20=get(popup20handle,'Value');
        h=waitbar(0,'Awaiting for calculus of Spectrogram...');
        for i=1:100,
            waitbar(i/100);
        end
        close(h);
        if popup20==1
            specgram(lefty,256,samplerate,kaiser(128,5));
            title('Left Channel Spectrogram');
            colormap(jet);
        elseif popup20==2
            specgram(righty,256,samplerate,kaiser(128,5));
            title('Right Channel Spectrogram');
            colormap(jet);
        end
    end
end
end
end

```

Questo case è molto semplice e consente il playback del segnale audio sino alla fine, utilizzando l'istruzione `soundsc`, che consente di effettuare lo scaling dei valori dei dati presenti in `y`, in modo da renderli compatibili con i livelli della scheda audio (ovviamente con un previo controllo di quanti canali ci sono nel segnale). Inoltre è stata inserita la funzionalità che permette la visualizzazione di una `waitbar` sincronizzata con il playback del file.

```

case 'play'
    if channel==1
        tr=count/samplerate;

```

```

inc=0.01;
texe=clock;soundsc(y,samplerate,quantize);e=etime(clock,texe);
texe2=clock;h=waitbar(0,'Playback time...');e2=etime(clock,texe2);
for i=0:inc:(tr-e-e2),
    waitbar(i/(tr-e-e2));
    pause(inc);
end
close(h);
elseif channel==2
tr=count/samplerate;
inc=0.01;
texe=clock;soundsc([lefty,righty],samplerate,quantize);e=etime(clock,texe);
texe2=clock;h=waitbar(0,'Playback time...');e2=etime(clock,texe2);
for i=0:inc:(tr-e-e2),
    waitbar(i/(tr-e-e2));
    pause(inc);
end
close(h);
end
end

```

Questo case è relativo al cambiamento della visualizzazione per l'asse y, nel caso di dominio temporale. Si ricava il valore del popup menù che stabilisce l'unità di misura scelta per l'asse y; se il valore è uguale a 1, allora si visualizza l'asse y con i valori dei campioni a seconda del livello di quantizzazione del segnale; se il valore è uguale a 2, effettua la conversione dei valori dei campioni dipendenti dal livello di quantizzazione del segnale, in valori normalizzati in un range di valori [-1;1); se il valore è uguale a 3, allora visualizza l'asse y con i valori percentuali, che altro non sono che i valori normalizzati moltiplicati per 100.

```

case 'timechange'
    domainhandle=findobj(gcf,'Tag','PopupMenu2');
    domain=get(domainhandle,'Value');
    if domain==1
        popup3handle=findobj(gcf,'Tag','PopupMenu3');
        popup3=get(popup3handle,'Value');
        if popup3==1
            h=waitbar(0,'Replotting with sample value...');
            for i=1:100,
                waitbar(i/100);
            end
            close(h);
            if channel==1
                t=(1:count)/samplerate;
                plot(t,y);
            end
        end
    end
end

```

```

    title('Plot of the Acquired Signal');
    ylabel('Sample values');
    xlabel('Seconds');
    legend('Monaural signal');
elseif channel==2
    t=(1:count)/samplerate);
    plot(t,y);
    title('Plot of the Acquired Signal');
    ylabel('Sample values');
    xlabel('Seconds');
    legend('Left Channel','Right Channel');
end
elseif popup3==2
h=waitbar(0,'Replotting with normalized value...');
for i=1:100,
    waitbar(i/100);
end
close(h);
if channel==1
    if quantize==8
        t=(1:count)/samplerate);
        d=(y-128)/128;
        plot(t,d);
        title('Plot of the Acquired Signal');
        ylabel('Normalized values');
        xlabel('Seconds');
        legend('Monaural signal');
    elseif quantize==16
        t=(1:count)/samplerate);
        d=y/32768;
        plot(t,d);
        title('Plot of the Acquired Signal');
        ylabel('Normalized values');
        xlabel('Seconds');
        legend('Monaural signal');
    end
elseif channel==2
    if quantize==8
        t=(1:count)/samplerate);
        d=(y-128)/128;
        plot(t,d);
        title('Plot of the Acquired Signal');
        ylabel('Normalized values');
        xlabel('Seconds');
        legend('Left Channel','Right Channel');
    elseif quantize==16
        t=(1:count)/samplerate);

```

```

    d=y/32768;
    plot(t,d);
    title('Plot of the Acquired Signal');
    ylabel('Normalized values');
    xlabel('Seconds');
    legend('Left Channel','Right Channel');
end
end
elseif popup3==3
h=waitbar(0,'Replotting with percentage value...');
for i=1:100,
    waitbar(i/100);
end
close(h);
if channel==1
    if quantize==8
        t=(1:count)/samplerate);
        d=((y-128)/128)*100;
        plot(t,d);
        title('Plot of the Acquired Signal');
        ylabel('Percentage values');
        xlabel('Seconds');
        legend('Monaural signal');
    elseif quantize==16
        t=(1:count)/samplerate);
        d=(y/32768)*100;
        plot(t,d);
        title('Plot of the Acquired Signal');
        ylabel('Percentage values');
        xlabel('Seconds');
        legend('Monaural signal');
    end
elseif channel==2
    if quantize==8
        t=(1:count)/samplerate);
        d=((y-128)/128)*100;
        plot(t,d);
        title('Plot of the Acquired Signal');
        ylabel('Percentage values');
        xlabel('Seconds');
        legend('Left Channel','Right Channel');
    elseif quantize==16
        t=(1:count)/samplerate);
        d=(y/32768)*100;
        plot(t,d);
        title('Plot of the Acquired Signal');
        ylabel('Percentage values');

```

```

        xlabel('Seconds');
        legend('Left Channel','Right Channel');
    end
end
end
end

```

Questo case consente il cambiamento del metodo di analisi della Power Spectral Density, quando si è nel dominio frequenziale. Si ricava il valore del popup menù che consente di scegliere il metodo con cui si vuole calcolare la PSD e a seconda del valore ottenuto, per replicazione del codice, si setta il samplerate per lo scaling dell'asse x del plot della PSD, il range di visualizzazione (intero o metà samplerate) e quale canale si vuole plottare. Tutti i metodi qui implementati, sono descritti dettagliatamente al paragrafo 3.3.1 e calcolati con FFT a 256 punti.

```

case 'changeepsdtype'
    domainhandle=findobj(gcf,'Tag','PopupMenu2');
    domain=get(domainhandle,'Value');
    if domain==2
        popup4handle=findobj(gcf,'Tag','PopupMenu4');
        popup4=get(popup4handle,'Value');
        if popup4==1
            sampleedit=findobj(gcf,'Tag','PopupMenu8');
            sampl=get(sampleedit,'Value');
            if sampl==1
                freq=8000;
            elseif sampl==2
                freq=11025;
            elseif sampl==3
                freq=22050;
            elseif sampl==4
                freq=44100;
            elseif sampl==5
                freq=48000;
            end
            popup5handle=findobj(gcf,'Tag','PopupMenu5');
            popup5=get(popup5handle,'Value');
            if popup5==1
                range='whole';
            elseif popup5==2
                range='half';
            end
            h=waitbar(0,'Awaiting for calculus of Power Spectral Density with Burg
Method...');
            for i=1:100,

```

```

        waitbar(i/100);
    end
close(h);
    if channel==1
        a=arburg(y,4);
        y1=filter(1,a,y);
        pburg(y1,4,256,freq,range);
        legend('Monaural Plot');
    elseif channel==2
        popup6handle=findobj(gcf,'Tag','PopupMenu6');
        popup6=get(popup6handle,'Value');
        if popup6==1
            a1=arburg(lefty,4);
            lefty1=filter(1,a1,lefty);
            pburg(lefty1,4,256,freq,range);
            legend('Left Channel Plot');
        elseif popup6==2
            a2=arburg(righty,4);
            righty1=filter(1,a2,righty);
            pburg(righty1,4,256,freq,range);
            legend('Right Channel Plot');
        end
    end
end
if popup4==2
    sampleedit=findobj(gcf,'Tag','PopupMenu8');
    sampl=get(sampleedit,'Value');
    if sampl==1
        freq=8000;
    elseif sampl==2
        freq=11025;
    elseif sampl==3
        freq=22050;
    elseif sampl==4
        freq=44100;
    elseif sampl==5
        freq=48000;
    end
    popup5handle=findobj(gcf,'Tag','PopupMenu5');
    popup5=get(popup5handle,'Value');
    if popup5==1
        range='whole';
    elseif popup5==2
        range='half';
    end
    h=waitbar(0,'Awaiting for calculus of Power Spectral Density with Covariance
Method...');

```

```

for i=1:100,
    waitbar(i/100);
    end
close(h);
    if channel==1
        a=arcov(y,4);
        y1=filter(1,a,y);
        pcov(y1,4,256,freq,range);
        legend('Monaural Plot');
    elseif channel==2
        popup6handle=findobj(gcf,'Tag','PopupMenu6');
        popup6=get(popup6handle,'Value');
        if popup6==1
            a1=arcov(lefty,4);
            lefty1=filter(1,a1,lefty);
            pcov(lefty1,4,256,freq,range);
            legend('Left Channel Plot');
        elseif popup6==2
            a2=arcov(righty,4);
            righty1=filter(1,a2,righty);
            pcov(righty1,4,256,freq,range);
            legend('Right Channel Plot');
        end
    end
end
end
if popup4==3
    sampleedit=findobj(gcf,'Tag','PopupMenu8');
    sampl=get(sampleedit,'Value');
    if sampl==1
        freq=8000;
    elseif sampl==2
        freq=11025;
    elseif sampl==3
        freq=22050;
    elseif sampl==4
        freq=44100;
    elseif sampl==5
        freq=48000;
    end
    popup5handle=findobj(gcf,'Tag','PopupMenu5');
    popup5=get(popup5handle,'Value');
    if popup5==1
        range='whole';
    elseif popup5==2
        range='half';
    end
end

```

```

h=waitbar(0,'Awaiting for calculus of Power Spectral Density with Modified
covariance Method...');
for i=1:100,
    waitbar(i/100);
end
close(h);
if channel==1
    a=armcov(y,4);
    y1=filter(1,a,y);
    pmcov(y1,4,256,freq,range);
    legend('Monaural Plot');
elseif channel==2
    popup6handle=findobj(gcf,'Tag','PopupMenu6');
    popup6=get(popup6handle,'Value');
    if popup6==1
        a1=armcov(lefty,4);
        lefty1=filter(1,a1,lefty);
        pmcov(lefty1,4,256,freq,range);
        legend('Left Channel Plot');
    elseif popup6==2
        a2=armcov(righty,4);
        righty1=filter(1,a2,righty);
        pmcov(righty1,4,256,freq,range);
        legend('Right Channel Plot');
    end
end
end
if popup4==4
    sampleedit=findobj(gcf,'Tag','PopupMenu8');
    sampl=get(sampleedit,'Value');
    if sampl==1
        freq=8000;
    elseif sampl==2
        freq=11025;
    elseif sampl==3
        freq=22050;
    elseif sampl==4
        freq=44100;
    elseif sampl==5
        freq=48000;
    end
    popup5handle=findobj(gcf,'Tag','PopupMenu5');
    popup5=get(popup5handle,'Value');
    if popup5==1
        range='whole';
    elseif popup5==2
        range='half';

```

```

end
    h=waitbar(0,'Awaiting for calculus of Power Spectral Density with Multitaper
Method (longest)...');
for i=1:100,
    waitbar(i/100);
    end        close(h);
if channel==1
y1=y;
pmtm(y1,4,256,freq,'adapt');
legend('Monaural Plot');
elseif channel==2
popup6handle=findobj(gcf,'Tag','PopupMenu6');
popup6=get(popup6handle,'Value');
if popup6==1
    lefty1=lefty;
    pmtm(lefty1,4,256,freq,'adapt');
    legend('Left Channel Plot');
elseif popup6==2
    righty1=righty;
    pmtm(righty1,4,256,freq,'adapt');
    legend('Right Channel Plot');
end
end
end
if popup4==5
sampleedit=findobj(gcf,'Tag','PopupMenu8');
sampl=get(sampleedit,'Value');
if sampl==1
    freq=8000;
elseif sampl==2
    freq=11025;
elseif sampl==3
    freq=22050;
elseif sampl==4
    freq=44100;
elseif sampl==5
    freq=48000;
end
popup5handle=findobj(gcf,'Tag','PopupMenu5');
popup5=get(popup5handle,'Value');
if popup5==1
    range='whole';
elseif popup5==2
    range='half';
end
    h=waitbar(0,'Awaiting for calculus of Power Spectral Density with MUSIC
Eigenvector Method...');

```

```

for i=1:100,
    waitbar(i/100);
    end
close(h);
if channel==1
    y1=y;
    pmusic(y1,4,256,freq);
    legend('Monaural Plot');
elseif channel==2
    popup6handle=findobj(gcf,'Tag','PopupMenu6');
    popup6=get(popup6handle,'Value');
    if popup6==1
        lefty1=lefty;
        pmusic(lefty1,4,256,freq);
        legend('Left Channel Plot');
    elseif popup6==2
        righty1=righty;
        pmusic(righty1,4,256,freq);
        legend('Right Channel Plot');
    end
end
end
if popup4==6
    sampleedit=findobj(gcf,'Tag','PopupMenu8');
    sampl=get(sampleedit,'Value');
    if sampl==1
        freq=8000;
    elseif sampl==2
        freq=11025;
    elseif sampl==3
        freq=22050;
    elseif sampl==4
        freq=44100;
    elseif sampl==5
        freq=48000;
    end
    popup5handle=findobj(gcf,'Tag','PopupMenu5');
    popup5=get(popup5handle,'Value');
    if popup5==1
        range='whole';
    elseif popup5==2
        range='half';
    end
    h=waitbar(0,'Awaiting for calculus of Power Spectral Density with Welch
Method...');
    for i=1:100,
        waitbar(i/100);

```

```

end
close(h);
if channel==1
    pwelch(y,256,samplerate,kaiser(128,5),0,0.95,'half');
    legend('Monaural Plot','95 Percentual confidence interval','95 Percentual
confidence interval');
    elseif channel==2
        popup6handle=findobj(gcf,'Tag','PopupMenu6');
        popup6=get(popup6handle,'Value');
        if popup6==1
            pwelch(lefty,256,samplerate,kaiser(128,5),0,0.95,'half');
            legend('Left Channel Plot','95 Percentual confidence interval','95 Percentual
confidence interval');
            elseif popup6==2
                pwelch(righty,256,samplerate,kaiser(128,5),0,0.95,'half');
                legend('Right Channel Plot','95 Percentual confidence interval','95
Percentual confidence interval');
            end
        end
end
end
if popup4==7
    sampleedit=findobj(gcf,'Tag','PopupMenu8');
    sampl=get(sampleedit,'Value');
    if sampl==1
        freq=8000;
    elseif sampl==2
        freq=11025;
    elseif sampl==3
        freq=22050;
    elseif sampl==4
        freq=44100;
    elseif sampl==5
        freq=48000;
    end
    popup5handle=findobj(gcf,'Tag','PopupMenu5');
    popup5=get(popup5handle,'Value');
    if popup5==1
        range='whole';
    elseif popup5==2
        range='half';
    end
    h=waitbar(0,'Awaiting for calculus of Power Spectral Density with Yule-Walker
AR Method...');
    for i=1:100,
        waitbar(i/100);
    end
close(h);

```

```

if channel==1
    a=aryule(y,4);
    y1=filter(1,a,y);
    pyulear(y1,4,256,samplerate,range);
    legend('Monaural plot');
elseif channel==2
    popup6handle=findobj(gcf,'Tag','PopupMenu6');
    popup6=get(popup6handle,'Value');
    if popup6==1
        a1=aryule(lefty,4);
        lefty1=filter(1,a1,lefty);
        pyulear(lefty1,4,256,samplerate,range);
        legend('Left Channel plot');
    elseif popup6==2
        a2=aryule(righty,4);
        righty1=filter(1,a2,righty);
        pyulear(righty1,4,256,samplerate,range);
        legend('Right Channel plot');
    end
end
end
end
end

```

Questo case consente il cambiamento del metodo di finestrazione con cui si calcola la STFT o spettrogramma e funziona solo se si è nel dominio dello spettrogramma. Si ricava il valore del popup menù che consente di scegliere il tipo di finestrazione richiesto, poi, per replicazione del codice, si calcola lo spettrogramma rispetto al metodo prescelto e al canale richiesto, quando il segnale è stereofonico. Le finestre sono tutte settate per 128 punti.

```

case 'changewindspectra'
    domainhandle=findobj(gcf,'Tag','PopupMenu2');
    domain=get(domainhandle,'Value');
    if domain==3
        popup11handle=findobj(gcf,'Tag','PopupMenu11');
        popup11=get(popup11handle,'Value');
        if popup11==1
            if channel==1
                h=waitbar(0,'Awaiting for calculus of Spectrogram with Bartlett window...');
                for i=1:100,
                    waitbar(i/100);
                end
                close(h);
                specgram(y,256,samplerate,bartlett(128));
                title('Monaural Spectrogram');
            end
        end
    end
end

```

```

        colormap(jet);
elseif channel==2
    h=waitbar(0,'Awaiting for calculus of Spectrogram with Bartlett window...');
    for i=1:100,
        waitbar(i/100);
    end
    close(h);
    popup20handle=findobj(gcf,'Tag','PopupMenu20');
    popup20=get(popup20handle,'Value');
    if popup20==1
        specgram(lefty,256,samplerate,bartlett(128));
        title('Left Channel Spectrogram');
        colormap(jet);
    elseif popup20==2
        specgram(righty,256,samplerate,bartlett(128));
        title('Right Channel Spectrogram');
        colormap(jet);
    end
end
end
if popup11==2
if channel==1
    h=waitbar(0,'Awaiting for calculus of Spectrogram with Blackman window...');
    for i=1:100,
        waitbar(i/100);
    end
    close(h);
    specgram(y,256,samplerate,blackman(128));
    title('Monaural Spectrogram');
    colormap(jet);
elseif channel==2
    h=waitbar(0,'Awaiting for calculus of Spectrogram with Blackman window...');
    for i=1:100,
        waitbar(i/100);
    end
    close(h);
    popup20handle=findobj(gcf,'Tag','PopupMenu20');
    popup20=get(popup20handle,'Value');
    if popup20==1
        specgram(lefty,256,samplerate,blackman(128));
        title('Left Channel Spectrogram');
        colormap(jet);
    elseif popup20==2
        specgram(righty,256,samplerate,blackman(128));
        title('Right Channel Spectrogram');
        colormap(jet);
    end
end
end

```

```

end
end
if popup11==3
if channel==1
    h=waitbar(0,'Awaiting for calculus of Spectrogram with Rectangular window...');
    for i=1:100,
        waitbar(i/100);
    end
    close(h);
    specgram(y,256,samplerate,boxcar(128));
    title('Monaural Spectrogram');
    colormap(jet);
elseif channel==2
    h=waitbar(0,'Awaiting for calculus of Spectrogram with Rectangular window...');
    for i=1:100,
        waitbar(i/100);
    end
    close(h);
    popup20handle=findobj(gcf,'Tag','PopupMenu20');
    popup20=get(popup20handle,'Value');
    if popup20==1
        specgram(lefty,256,samplerate,boxcar(128));
        title('Left Channel Spectrogram');
        colormap(jet);
    elseif popup20==2
        specgram(righty,256,samplerate,boxcar(128));
        title('Right Channel Spectrogram');
        colormap(jet);
    end
end
end
if popup11==4
if channel==1
    h=waitbar(0,'Awaiting for calculus of Spectrogram with Chebyshev window...');
    for i=1:100,
        waitbar(i/100);
    end
    close(h);
    specgram(y,256,samplerate,chebwin(128,3));
    title('Monaural Spectrogram');
    colormap(jet);
elseif channel==2
    h=waitbar(0,'Awaiting for calculus of Spectrogram with Chebyshev window...');
    for i=1:100,
        waitbar(i/100);
    end
    close(h);

```

```

        popup20handle=findobj(gcf,'Tag','PopupMenu20');
popup20=get(popup20handle,'Value');
if popup20==1
    specgram(lefty,256,samplerate,chebwin(128,3));
    title('Left Channel Spectrogram');
    colormap(jet);
elseif popup20==2
    specgram(righty,256,samplerate,chebwin(128,3));
    title('Right Channel Spectrogram');
    colormap(jet);
end
end
end
if popup11==5
if channel==1
    h=waitbar(0,'Awaiting for calculus of Spectrogram with Hamming window...');
    for i=1:100,
        waitbar(i/100);
    end
    close(h);
    specgram(y,256,samplerate,hamming(128));
    title('Monaural Spectrogram');
    colormap(jet);
elseif channel==2
    h=waitbar(0,'Awaiting for calculus of Spectrogram with Hamming window...');
    for i=1:100,
        waitbar(i/100);
    end
    close(h);
    popup20handle=findobj(gcf,'Tag','PopupMenu20');
popup20=get(popup20handle,'Value');
    if popup20==1
        specgram(lefty,256,samplerate,hamming(128));
        title('Left Channel Spectrogram');
        colormap(jet);
    elseif popup20==2
        specgram(righty,256,samplerate,hamming(128));
        title('Right Channel Spectrogram');
        colormap(jet);
    end
end
end
if popup11==6
if channel==1
    h=waitbar(0,'Awaiting for calculus of Spectrogram with Hanning window...');
    for i=1:100,
        waitbar(i/100);

```

```

end
close(h);
    specgram(y,256,samplerate,hanning(128));
    title('Monaural Spectrogram');
    colormap(jet);
elseif channel==2
    h=waitbar(0,'Awaiting for calculus of Spectrogram with Hanning window...');
for i=1:100,
    waitbar(i/100);
end
    close(h);
    popup20handle=findobj(gcf,'Tag','PopupMenu20');
popup20=get(popup20handle,'Value');
    if popup20==1
        specgram(lefty,256,samplerate,hanning(128));
        title('Left Channel Spectrogram');
        colormap(jet);
    elseif popup20==2
        specgram(righty,256,samplerate,hanning(128));
        title('Right Channel Spectrogram');
        colormap(jet);
    end
end
end
if popup11==7
if channel==1
    h=waitbar(0,'Awaiting for calculus of Spectrogram with Triangular window...');
for i=1:100,
    waitbar(i/100);
end
    close(h);
    specgram(y,256,samplerate,triang(128));
    title('Monaural Spectrogram');
    colormap(jet);
elseif channel==2
    h=waitbar(0,'Awaiting for calculus of Spectrogram with Triangular window...');
for i=1:100,
    waitbar(i/100);
end
    close(h);
    popup20handle=findobj(gcf,'Tag','PopupMenu20');
popup20=get(popup20handle,'Value');
    if popup20==1
        specgram(lefty,256,samplerate,triang(128));
        title('Left Channel Spectrogram');
        colormap(jet);
    elseif popup20==2

```

```

        specgram(righty,256,samplerate,triang(128));
        title('Right Channel Spectrogram');
        colormap(jet);
    end
end
end
if popup11==8
if channel==1
    h=waitbar(0,'Awaiting for calculus of Spectrogram with Kaiser window...');
    for i=1:100,
        waitbar(i/100);
    end
    close(h);
    specgram(y,256,samplerate,kaiser(128,5));
    title('Monaural Spectrogram');
    colormap(jet);
elseif channel==2
    h=waitbar(0,'Awaiting for calculus of Spectrogram with Kaiser window...');
    for i=1:100,
        waitbar(i/100);
    end
    close(h);
    popup20handle=findobj(gcf,'Tag','PopupMenu20');
    popup20=get(popup20handle,'Value');
    if popup20==1
        specgram(lefty,256,samplerate,kaiser(128,5));
        title('Left Channel Spectrogram');
        colormap(jet);
    elseif popup20==2
        specgram(righty,256,samplerate,kaiser(128,5));
        title('Right Channel Spectrogram');
        colormap(jet);
    end
end
end
end
end

```

Questo case consente di visualizzare le informazioni statistiche relative al segnale acquisito, tramite la pressione del pulsante Statistical Parameters, il cui callback associato è relativo a questo case. Si ricava, tramite le variabili globali, inizialmente definite, il numero di bit di quantizzazione, il numero di canali, la frequenza di campionamento, si calcola la media, i valori dei campioni massimi e minimi, la varianza (o momento centrale del secondo ordine), la mediana e infine si compone la stringa con tutti questi valori, che verranno poi inviati ad una message box. È presente anche una funzione che calcola il bitrate del segnale, sia PCM che MP3, per gli ultimi si utilizza un

programma esterno chiamato MP3INFO, che permette di determinare le informazioni del file tramite lo scanning degli “ancillary data” dello stesso.

```

case 'statview'
    if strcmp(ext, '.wav')
        h=waitbar(0, 'Gathering Statistical information...');
        for i=1:100,
            waitbar(i/100);
        end
        close(h);
        q=sprintf('Bit Depth is %d\n', quantize);
        r=sprintf('Samplerate in Hertz is %d\n', samplerate);
        b=sprintf('Bitrate in kbps for this PCM file is
%.1f\n', (samplerate*quantize*channel)/1000);
        if channel==1
            m=sprintf('Mean Value is %.3f\n', mean(y));
            maxs=sprintf('Maximum Sample is @ %d sample\n', max(y));
            mins=sprintf('Minimum Sample is @ %d sample\n', min(y));
            v=sprintf('Variance is %.3f\n', var(y));
            med=sprintf('Median (Offset to DC Value established) is %d\n', median(y));
            startins=sprintf('\n\n%s\n\n', 'This is a monaural signal with this parameter:');
            complex=strcat(startins, q, r, b, m, maxs, mins, v, med);
            msgbox(complex, 'Statistical Information');
        elseif channel==2
            ml=sprintf('Mean Value (Left Channel) is %.3f\n', mean(lefty));
            mr=sprintf('Mean Value (Right Channel) is %.3f\n', mean(righty));
            maxsl=sprintf('Maximum Sample (Left Channel) is @ %d sample\n', max(lefty));
            maxsr=sprintf('Maximum Sample (Right Channel) is @ %d sample\n', max(righty));
            minsl=sprintf('Minimum Sample (Left Channel) is @ %d sample\n', min(lefty));
            minsr=sprintf('Minimum Sample (Right Channel) is @ %d sample\n', min(righty));
            vl=sprintf('Variance (Left Channel) is %.3f\n', var(lefty));
            vr=sprintf('Variance (Right Channel) is %.3f\n', var(righty));
            medl=sprintf('Median (Offset to DC Value established on Left Channel) is
%d\n', median(lefty));
            medr=sprintf('Median (Offset to DC Value established on Right Channel) is
%d\n', median(righty));
            startins=sprintf('\n\n%s\n\n', 'This is a stereo (two channel) signal with this
parameter:');
            complex=strcat(startins, q, r, b, ml, mr, maxsl, maxsr, minsl, minsr, vl, vr, medl, medr);
            msgbox(complex, 'Statistical Information');
        end
    elseif strcmp(ext, '.mp3')
        h=waitbar(0, 'Gathering Statistical information...');
        for i=1:100,
            waitbar(i/100);
        end
    end

```

```

close(h);
cmd = ['mp3info -r m -p "%r" ',string];
[s,w]=dos(cmd);
q=sprintf('Bit Depth is %d\n',quantize);
r=sprintf('Samplerate in Hertz is %d\n',samplerate);
b=sprintf('Bitrate in kbps for original PCM file was
%.1f\n',((samplerate*quantize*channel)/1000));
x=w((1+max([0,findstr(w,10)])):end);
nb=sprintf('Bitrate in kbps (median) for encoded MP3 file is now %s\n',x);
if channel==1
    m=sprintf('Mean Value is %.3f\n',mean(y));
    maxs=sprintf('Maximum Sample is @ %d sample\n',max(y));
    mins=sprintf('Minimum Sample is @ %d sample\n',min(y));
    v=sprintf('Variance is %.3f\n',var(y));
    med=sprintf('Median (Offset to DC Value established) is %d\n',median(y));
    startins=sprintf('\n\n%s\n\n','This is a monaural signal with this parameter:');
    complex=strcat(startins,q,r,b,nb,m,maxs,mins,v,med);
    msgbox(complex,'Statistical Information');
elseif channel==2
    ml=sprintf('Mean Value (Left Channel) is %.3f\n',mean(lefty));
    mr=sprintf('Mean Value (Right Channel) is %.3f\n',mean(righty));
    maxsl=sprintf('Maximum Sample (Left Channel) is @ %d sample\n',max(lefty));
    maxsr=sprintf('Maximum Sample (Right Channel) is @ %d sample\n',max(righty));
    minsl=sprintf('Minimum Sample (Left Channel) is @ %d sample\n',min(lefty));
    minsr=sprintf('Minimum Sample (Right Channel) is @ %d sample\n',min(righty));
    vl=sprintf('Variance (Left Channel) is %.3f\n',var(lefty));
    vr=sprintf('Variance (Right Channel) is %.3f\n',var(righty));
    medl=sprintf('Median (Offset to DC Value established on Left Channel) is
%d\n',median(lefty));
    medr=sprintf('Median (Offset to DC Value established on Right Channel) is
%d\n',median(righty));
    startins=sprintf('\n\n%s\n\n','This is a stereo (two channel) signal with this
parameter:');
    complex=strcat(startins,q,r,b,nb,ml,mr,maxsl,maxsr,minsl,minsr,vl,vr,medl,medr);
    msgbox(complex,'Statistical Information');
end
end

```

Questo case è associato al callback del pulsante Info e visualizza una help dialog, con le informazioni sul creatore del software e come usarlo al meglio.

```

case 'credits'
    h=helpdlg('This is the Signal Acquisition system inside of Nova - Mod Tools enviroment
built by Zambelli Cristian. For using this feature you need an audio sound card Microsoft

```

```
Windows compatible for acquiring external signals or you can only process pre-recorded  
file', 'Information');
```

Qui si conclude il codice, con il case di chiusura dell'applicazione. Le istruzioni svolte sono: il salvataggio dei dati sul file di interscambio, la pulizia delle variabili dalla memoria e dal workspace di Matlab e la chiusura dell'applicazione.

```
case 'close'  
    save('xchangedata');  
    clear variables;  
    close(gcf);  
end
```

Capitolo 4

Sviluppo degli algoritmi di modulazione diretta del segnale

4.1 Prefazione del sistema

Siamo arrivati ad un punto del sistema, in cui bisogna cominciare a “mettere le mani” attivamente sul segnale audio acquisito. E questo cosa significa? Significa che dobbiamo parlare del processo di modulazione diretta di un segnale audio. La differenza fondamentale che c'è tra il procedimento di modulazione diretta di un segnale e l'applicazione di un effetto di modulazione al segnale audio è che: il primo metodo consiste nella modifica (o meglio nell'alterazione) dell'inviluppo¹ e dello spettro del segnale, senza andare ad agire su una versione del segnale ritardato e precedentemente biforcuto, ma considerando direttamente la forma d'onda del segnale acquisito (o al massimo una sua versione opportunamente filtrata); il secondo metodo invece, prevede l'utilizzo di una combinazione di più versioni ritardate del segnale di ingresso, in modo da creare l'effetto di modulazione desiderato. Si dovrà realizzare ancora una volta quindi, un'applicazione GUI in linguaggio Matlab, che si adatti all'environment Nova – Mod Tools e in particolare alla parte del sistema che è costituita dall'applicazione Signal Acquisition System e dal sistema di creazione delle forma d'onda a frequenza sub – sonica, per la generazione della modulazione, cioè LFO – Maker. Gli oggetti principali richiesti dal sistema di modulazione diretta del segnale, per poter agire sulla manipolazione del contenuto informativo dello stesso, sono:

- Un segnale audio correttamente acquisito con il sistema di Signal Acquisition, presente all'interno dell'environment. Di questo segnale, si dovranno estrarre alcune caratteristiche fondamentali come: il samplerate (che dovrà essere uguale in uscita dall'effetto di modulazione diretta), la profondità di quantizzazione in bit (anche questa resterà invariata), il numero di canali con cui il segnale è formato e ovviamente, i valori dei campioni che ne costituiscono il contenuto.
- Un LFO. In questo caso ci viene in soccorso, l'applicazione LFO – Maker, la quale permette non solo di simulare ogni tipo di LFO presente sul mercato oggi, ma anche qualsiasi LFW si voglia creare autonomamente, avendo a disposizione una schiera piuttosto ampia di parametri su cui si può agire, per ottenere la forma d'onda desiderata, da applicare al segnale

¹ Per inviluppo del segnale audio, si intende la stessa definizione data ad un generico segnale, nella teoria delle comunicazioni elettriche.

per il processo di modulazione². Dalla LFW scelta per l'applicazione della modulazione diretta, si ricaveranno i parametri fondamentali di: frequenza (che per l'applicazione di modulazione ad anello, verrà modificata secondo alcune regole), offset con lo zero di riferimento, fase, smorzamento, parametri aggiuntivi (RSS Simulation, Wall – Mart Simulation, ecc.) e ovviamente il tipo di onda. Da notare che non è stata citata l'ampiezza della LFW, perché molto spesso si utilizzerà un LFW normalizzata tra i valori $[-1+\text{offset}; 1+\text{offset}]$, in modo da rientrare nei parametri di playback di un segnale audio³.

- Banchi di filtri progettati in modo digitale (operazione opzionale). Per alcuni effetti di modulazione diretta, il filtraggio del segnale risulta essere una componente chiave. Ad esempio, se l'LFOCoder non avesse i filtri su cui far agire gli amplificatori controllati in tensione, non si riuscirebbe ad ottenere determinate sonorità. Si sceglie di progettare questi filtri con una soluzione digitale, per una questione di compatibilità di dominio: dal momento che stiamo lavorando con campioni di un segnale (e quindi in un dominio discreto), è giusto lavorare su filtri che lavorano sui campioni del segnale, cioè filtri digitali. Parleremo meglio nel prossimo paragrafo, dei metodi utilizzati per il design dei filtri e di come questi vengano implementati in un sistema come il nostro.

Andremo a realizzare in particolare, tre tipi di effetti di modulazione diretta del segnale, che andremo a valutare meglio nei paragrafi successivi e sono: il Tremolo, il Ring Modulator e l'LFOCoder. L'ultimo di questi effetti in particolare, rappresenta una nuova metodologia di modulazione diretta dei segnali audio, che deriva direttamente da un'applicazione di un vecchio sistema conosciuto come VoCoder.

4.2 La modulazione di ampiezza sui segnali audio con LFW

4.2.1 Il Tremolo: cos'è e come fa ...

Uno dei principali metodi, conosciuti da quando l'uomo ha inventato la chitarra elettrica, per variare l'ampiezza di un segnale audio, in un uscita da uno strumento musicale durante l'esecuzione di un brano, è quello di andare a muovere il potenziometro che regola il volume di uscita dello strumento,

² In questo capitolo si utilizza il termine modulazione, per indicare la modulazione diretta del segnale. Qualora si utilizzi il termine per altri significati, ne verrà esplicitamente riferito il contesto e il significato.

³ Quando si utilizza la funzione *sound* di Matlab, per eseguire il playback di un segnale audio, si intende che i valori dei campioni che costituiscono il segnale varino in un intervallo normalizzato tra $[-1;1]$. Ecco perché la forma d'onda che dovrà eseguire la modulazione diretta del segnale, dovrà anch'essa essere compresa in questo intervallo.

in modo da avere un suono “pulsante”, a tratti interrotto dal movimento della mano. La maggior parte delle volte, ad esempio nel bel mezzo di un solo, può però non essere comodo andare ad eseguire questo movimento, per cui ci si deve aiutare con quello che l’elettronica ci può fornire. In sostanza, quello che noi andiamo a fare, è andare a modulare l’ampiezza del nostro segnale audio, a seconda della posizione in cui si trova il potenziometro del volume, regolato dalla nostra mano: ci serve quindi un modulatore di ampiezza. Niente di più semplice (per il momento ...): si prende un oscillatore in grado di simulare il movimento della nostra mano, tipicamente un LFO, lo si connette con un moltiplicatore al segnale di ingresso, ed ecco avvenuta la modulazione. Fino agli anni ’40⁴, all’interno degli amplificatori per strumenti musicali che incorporavano la soluzione circuitale (a volte costruita in modo puramente meccanico da molle), che permetteva di simulare questa metodologia di modulazione, si dava un nome all’effetto creato, che è rimasto a tutt’oggi: il Tremolo. Proprio per la caratteristica pulsante a volte leggermente instabile di quest’oggetto, mai nome fu più azzeccato di questo. Il fatto però di applicare, una modulazione di ampiezza al segnale, dalla teoria delle comunicazioni elettriche, ci dice che in uscita all’effetto, ci sarà un segnale, le cui componenti frequenziali saranno modificate dal segnale modulante, infatti bisogna fare una considerazione attenta sul fenomeno distruttivo principale, quando si usa un effetto di questo tipo: la generazione dei sub – armonici⁵. Dal momento che il segnale modulante è un segnale a frequenza molto bassa, una frequenza che l’orecchio umano non riesce a percepire, e questa frequenza di va ad insinuare nel prodotto di questo segnale con il segnale modulato, si generano delle righe spettrali a frequenze non percepibili dall’orecchio umano. Ma fino a qui non ci sarebbe nessun problema, se non le posso sentire tanto meglio ... Ebbene non è così. Consideriamo ad esempio un sistema costituito da un amplificatore per grandi segnali in classe A⁶ con una tensione di uscita a vuoto di $60 V_{pp}$, collegato ad una cassa acustica con una impedenza di carico di circa 4Ω . Dalla teoria dei circuiti, si sa che l’impedenza del carico, è variabile con la frequenza e in particolare, per f che tende ad un valore pressoché nullo (DC), l’impedenza del carico tende ad annullarsi, quindi tendiamo ad essere nella situazione di corto circuito. Ciò significa che per frequenze molto basse, la tensione di uscita tende a circolare interamente su un corto circuito, causando un innalzamento di corrente elevato, che porta quasi sicuramente alla distruzione del sistema. Ma anche se non arriviamo proprio al punto critico di corto circuito, il fatto che nel segnale siano sempre presenti delle componenti a bassa frequenza, porta a “stressare” il funzionamento del sistema, accorciandone quindi la vita utile. Per questo, almeno per quanto riguarda la soluzione adottata nella simulazione

⁴ Il primo prototipo di Tremolo risale agli anni ’40 con il marchio della DanElectro.

⁵ In alcuni testi sono anche definiti come “righe spettrali sub – soniche”.

⁶ Classe di amplificatori di potenza in cui l’angolo di circolo della corrente è pari a $\theta=360^\circ$, ma che possiedono una scarsa efficienza in termini di prestazioni, dal momento che dissipano molta energia.

dell'effetto con Matlab, si è introdotto un filtro di Butterworth passa – alto del nono ordine, con una frequenza di cut – off di circa 20 Hz, in modo da preservare la dinamica del segnale degli strumenti come il basso elettrico a 5 corde, eliminando però i sub – armonici prodotti dal processo di modulazione.

4.2.2 Ring Modulator

Un'applicazione che viene generalmente conosciuta come un effetto “figlio” del Tremolo, è il Ring Modulator o modulatore ad anello a portante soppressa. Il nome di modulatore ad anello, deriva dal fatto che uno dei primi metodi utilizzati per l'implementazione di questa unità effetti, consisteva in un anello di quattro diodi e una coppia di trasformatori. Questo sistema è anche conosciuto come modulatore lattice – type e può essere sostituito anche con un circuito passivo. Un altro metodo di costruzione di queste unità (tipicamente usato in soluzioni che richiedono circuiti integrati), è quello di usare un “moltiplicatore a quattro quadranti”. Come nel Tremolo, anche in questo caso si va ad effettuare una modulazione di ampiezza sul segnale audio, ma invece di utilizzare come segnale modulante una LFW, si utilizza un segnale ad una frequenza che è circa 100 volte la frequenza caratteristica di un LFW. Ciò significa che il range di frequenze di questo effetto varia tra 30 e 1800 Hz. La modulazione ad anello consiste nella moltiplicazione di due segnali per ottenere l'uscita in questo modo:

$$out(t) = in_1(t) \cdot in_2(t) \quad (1)$$

$$out(t) = in_1(t) \cdot \sin(2\pi ft) \quad (2)$$

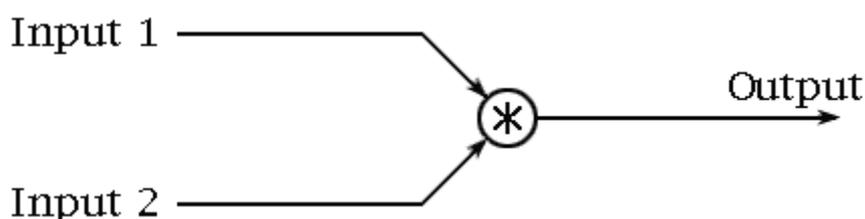


Figura 27: Schema di modulazione ad anello

abituamente, il primo segnale arriva da uno strumento musicale, mentre il secondo è un'onda sinusoidale generata internamente all'effetto (2). In termini frequenziali, i toni prodotti da questo processo di modulazione diretta sono: il prodotto e la somma delle frequenze originali. Supponendo

che il segnale di ingresso dello strumento musicale, abbia una componente frequenziale a 500 Hz e che la frequenza del segnale di modulazione sia 300 Hz, l'uscita sarà composta da due "righe spettrali" a 200 e a 800 Hz. Per una migliore applicazione dell'effetto, si dovrebbe avere dei segnali a DC Offset Null⁷, altrimenti sarà sempre presente nel segnale di uscita, una componente a 300 Hz, anche quando il segnale ha livello 0. Alcune unità effetti come le unità Rack – Mounted o i sintetizzatori analogici come il Modular Moog (dove questo effetto è applicato maggiormente), utilizzano un singolo gate XOR per svolgere la moltiplicazione digitalmente. Per fare questo con i DSP, è necessario convertire i segnali di ingresso in valori binari e mandarli in XOR insieme. Un'operazione di filtraggio finale, potrebbe essere necessaria, dal momento che il suono di uscita risulta fortemente distorto, specialmente quando il divario di frequenza tra il segnale di ingresso e l'oscillatore, è piuttosto elevato.

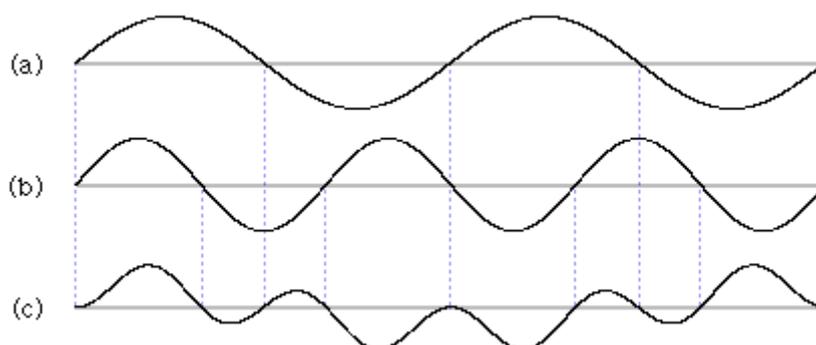


Figura 28: Esempio di modulazione ad anello, il segnale (a) la sinusoide proveniente dallo strumento musicale, (b) è il segnale modulato e (c) è il risultato della modulazione

Questo non è un effetto di armonizzazione in generale, in quanto le note prodotte dal ring modulator non sono legate al segnale di ingresso mediante intervalli musicali specifici. Il suono prodotta da queste unità è generalmente dissonante e sporco, spesso paragonabile al suono di un gong⁸.

4.2.3 Speed Enhancement

Prima di eseguire il benchmark con il software Cool Edit Pro, in base alla velocità di applicazione degli algoritmi sviluppati per la modulazione diretta del segnale, è necessario espletare un ragionamento che sta alla base di tutto il lavoro svolto e probabilmente servirà anche ai blocchi

⁷ Sono segnali audio, che hanno il livello di riferimento perfettamente a 0 VDC, quindi privi di offset.

⁸ Il suono delle campane spesso contiene, durante la risonanza, molte componenti non armoniche, cioè non legate da precisi intervalli musicali.

futuri dell'environment Nova – Mod Tools. L'operazione più comune che si va a svolgere, quando è necessaria una modulazione di ampiezza sul segnale è in sostanza un prodotto tra gli elementi interni di due matrici. In particolare, supponendo di voler applicare l'effetto di Tremolo al segnale audio acquisito, dovremo compiere la seguente operazione:

$$\begin{bmatrix} in(1,1) & in(1,2) \\ in(2,1) & in(2,2) \\ \dots & \dots \\ in(n,1) & in(n,2) \end{bmatrix} \cdot \times \begin{bmatrix} lfo(1) \\ lfo(2) \\ \dots \\ lfo(n) \end{bmatrix} = \begin{bmatrix} out(1,1) & out(1,2) \\ out(2,1) & out(2,2) \\ \dots & \dots \\ out(n,1) & out(n,2) \end{bmatrix} \quad (3)$$

dove n rappresenta il numero di campioni che costituiscono il segnale acquisito e la LFW.

Un'operazione di questo tipo, che richiede una moltiplicazione elemento per elemento, può essere implementata in Matlab, ma richiede un accorgimento principale, al fine di ridurre il tempo di esecuzione dell'operazione stessa e “snellire” la memoria del sistema su cui è installata l'applicazione. Se infatti si dovesse provare dal prompt di Matlab, a svolgere questa banale operazione, tra due matrici di 10000 componenti, non si otterrebbe di certo un risultato in tempi brevi, anzi ... Tanto peggio quando il segnale è di tipo stereofonico, perché il numero delle operazioni raddoppia, dal momento che esistono due canali. L'idea di fondo allora parte dalla definizione di FFT. Cosa c'entra la Trasformata di Fourier “veloce” con una modulazione di ampiezza? Apparentemente niente. Infatti non si esegue la FFT del segnale, piuttosto si segue il ragionamento che applica la FFT per trasformare un segnale da dominio del tempo al dominio delle frequenze. La trasformata di Fourier veloce, si chiama così perché permette di calcolare con un sistema a microprocessore, in modo molto più rapido del tradizionale, la trasformata di Fourier di un segnale arbitrario qualsiasi. Lo fa utilizzando un algoritmo chiamato a decimazione di tempo⁹, il quale esegue la trasformata di Fourier su blocchi di campioni pari e su blocchi di campioni dispari, implementando un processo computazionale di tipo butterfly (che è stato definito nelle note del capitolo 3). Perché allora non si può fare la stessa cosa con Matlab? Proviamoci. Il primo tentativo è quello di separare il segnale audio arbitrario in due parti: la parte con i campioni pari e la parte con i campioni dispari. Eseguendo questa separazione e provando ad effettuare l'operazione di moltiplicazione, si nota che le prestazioni migliorano fino al punto in cui, la memoria del nostro calcolatore, che è costretto in contemporanea a moltiplicare e ad aggiornare gli indici della matrice che dovrà contenere i valori dei campioni del segnale di uscita, crolla. Anche in questo caso le prestazioni prima o poi crollano. Ci viene allora in aiuto, un'altra definizione che apparentemente

⁹ In realtà questo non è l'unico algoritmo, si prende questo perché è l'esempio più semplice. Esistono comunque metodi migliori come ad esempio l'Overlap and Add e l'Overlap and Save.

può sembrare inutile, il “circular buffer” presente nei DSP. Il “circular buffer” o buffer ad anello, tipicamente presente nei processori digitali per segnali, permette, durante l’acquisizione di un dato in tempo reale, di salvare i campioni successivi in un buffer che assomiglia ad una struttura ad anello, in cui l’ultimo campione acquisito viene spostato in avanti di una posizione nel buffer, fino ad essere eliminato (dopo un giro completo), per far posto ai nuovi campioni in arrivo. Questo è il ragionamento giusto per Matlab, ma bisogna implementare la divisione in blocchi del segnale audio, senza perdere campioni dovuti al fenomeno dell’overflow¹⁰ e del buffer overrun¹¹. Dopo alcuni esperimenti, pare che la dimensione giusta di ogni blocco, sia stimabile in un numero di campioni pari a 32. Per cui per dividere in blocchi si usa la seguente formula:

$$blocknum = \text{fix}\left(\frac{nsample}{32}\right) \quad (4)$$

che ci dà il numero corretto di blocchi in cui dividere il segnale. L’operazione di *fix* serve a restituire solamente la parte intera della divisione. In questo modo si può usare la seguente formula per eseguire la moltiplicazione delle matrici, tramite circular buffering:

$$\text{out}(\text{currentblock} \cdot 32 : \text{successiveblock} \cdot 32) = \text{in}(\dots) \cdot \text{lfo}(\dots) \quad (5)$$

dove questa notazione in Matlab significa, prendi gli elementi della matrice da *currentblock**32 a *successiveblock**32. In questo modo però, ci si accorge subito che l’algoritmo ha un difetto: vengono persi i primi 32 campioni e gli ultimi campioni, qualora la formula (4) abbia un resto. Per ovviare al problema, si impone a Matlab di cominciare il calcolo con l’indice a 0 e variare la formula (5) in:

$$\text{out}(1:32) = \text{in}(1:32) \cdot \text{lfo}(1:32) \quad (6)$$

quando l’indice è ancora a 0; mentre per il resto, lo si può calcolare con l’operazione di modulo anziché di *fix* nella (4) e variare ancora una volta la formula (5) con gli indici che variano dall’ultimo dei blocchi successivi al resto della divisione. Anche in questo modo però Matlab non è al massimo della velocità. La soluzione finale, consiste nel inizializzare la matrice che dovrà contenere i dati dei campioni del segnale di uscita, mediante l’istruzione *zeros*, che permette di

¹⁰ Fenomeno per cui, il dato che viene acquisito, è troppo grande per stare all’interno del buffer e di conseguenza viene troncato.

¹¹ Fenomeno per cui, non essendo il buffer di acquisizione, sufficientemente veloce a trasferire le informazioni nella locazione voluta per la memorizzazione dei dati, sovrascrive i campioni che devono ancora compiere un giro all’interno del buffer (cioè correttamente trasferiti).

porre i valori interni di una matrice assegnata, tutti a 0. Così si è nettamente velocizzato le prestazioni delle applicazioni di modulazione diretta di un segnale audio arbitrario e ne abbiamo una prova dalle seguenti tabelle:

1. Speed Enhancement per l'applicazione dell'effetto di Tremolo ad un segnale audio

	Direct Modulation Tools con la funzione Tremolo attivata per una LFW sinusoidale pura a 2 Hz (con buffer)	Direct Modulation Tools con la funzione Tremolo attivata per una LFW sinusoidale pura a 2 Hz (senza buffer)
Segnale Microsoft PCM WAVE FORMAT Monaurale campionato a 44100 Hz e quantizzato a 16 bit da 15 sec.	25.34 sec compreso il filtraggio per l'eliminazione dei sub – armonici e il plot	701.21 sec senza il filtraggio per l'eliminazione dei sub – armonici e senza il plot
Segnale Microsoft PCM WAVE FORMAT Stereofonico campionato a 22050 Hz e quantizzato a 16 bit da 2 sec.	5.66 sec compreso il filtraggio per l'eliminazione dei sub – armonici e il plot	204.40 sec compreso il filtraggio per l'eliminazione dei sub – armonici e il plot

2. Speed Enhancement per l'applicazione dell'effetto di Ring Modulator ad un segnale audio

	Direct Modulation Tools con la funzione Ring Modulator attivata per una LFW sinusoidale pura a 2 Hz (con buffer)	Direct Modulation Tools con la funzione Ring Modulator attivata per una LFW sinusoidale pura a 2 Hz (senza buffer)
Segnale Microsoft PCM WAVE FORMAT Monaurale campionato a 44100 Hz e quantizzato a 16 bit da 15 sec.	24.55 sec compreso il plot	528.76 sec senza il plot
Segnale Microsoft PCM WAVE FORMAT Stereofonico campionato a 22050 Hz e quantizzato a 16 bit da 2 sec.	5.59 sec compreso il plot	256.40 sec compreso il plot

3. Speed Enhancement per l'applicazione dell'effetto LFOCoder ad un segnale audio

	Direct Modulation Tools con la funzione LFOCoder attivata per tre LFW diverse (con buffer)	Direct Modulation Tools con la funzione LFOCoder attivata per tre LFW diverse (senza buffer)
Segnale Microsoft PCM WAVE FORMAT Monaurale campionato a 44100 Hz e quantizzato a 16 bit da 15 sec.	29.69 sec compreso il mixing e il plot	N/A a causa di un errore di “out of memory” che impedisce a Matlab di proseguire nei calcoli
Segnale Microsoft PCM WAVE FORMAT Stereofonico campionato a 22050 Hz e quantizzato a 16 bit da 2 sec.	8.83 sec compreso il mixing e il plot	N/A a causa di un errore di “out of memory” che impedisce a Matlab di proseguire nei calcoli

Si è poi pensato di utilizzare la strategia dell'in – place computation¹² per tentare di velocizzare ulteriormente le operazioni di applicazione dell'effetto e il risultato è stato veramente strepitoso. Utilizzando la tecnologia della riallocazione degli spazi lineari, introdotta da MathWorks, la quale prevede di utilizzare un vettore come indicizzatore di ciclo (la tecnica vera è disponibile andando ad osservare il codice nativo di Matlab) con la strategia combinata DSP – type, le prestazioni sono migliorate fino al 400%, portando un aumento notevole della capacità di calcolo e di velocità di applicazione degli algoritmi.

4.3 LFOCoder

4.3.1 VoCoding e idea di fondo

Il VoCoding¹³ è un effetto che ha la funzione principale di sovrimporre tipicamente la voce umana, al di sopra di un altro strumento musicale. Questo effetto speciale, studiato e realizzato in forma di prototipo negli anni '70, ha la funzione tipica, di far sembrare che il proprio strumento musicale sia in grado di parlare. Probabilmente, i migliori espositori di questo tipo di effetto di modulazione diretta del segnale sono: Peter Frampton (lui è solito chiamare questo effetto come “The Pig”, anche se più che a un Vocoder assomiglia a un Talk Box¹⁴) e i Kraftwerk nel brano “The Man Machine”. L'equipaggiamento tipico di un Vocoder, consiste in un due set di filtri passabanda. Filtri di questa tipologia, come dice il nome, permettono il passaggio di un determinato range di frequenze e il valore centrale di questo range viene definito “Center Frequency”, mentre la differenza tra il limite superiore ed inferiore del filtro, viene definito “Bandwidth”. Un singolo set di questi filtri, costituisce quindi un banco, le cui frequenze di centro banda e la larghezza di banda, sono accordate in modo da coprire più o meno, tutto lo spettro occupato dai segnali in banda base audio (orecchio umano perfetto). Un possibile esempio di configurazione di banco è dato dall'utilizzo di filtri che hanno la loro frequenza di centro banda settata a un ottava¹⁵ di distanza per ogni filtro. Uno dei filtri ad esempio, potrebbe avere una frequenza centrale di 1 KHz, in modo che i suoi “parenti” abbiano frequenza centrale pari alla metà e al doppio del primo filtro. Questa configurazione permette un overlapping delle bande a metà ottava, ciò significa che c'è una copertura di un range che va da 250

¹² Tecnica di calcolo utilizzata nei DSP per velocizzare le operazioni e risparmiare memoria, ottenuta rimpiazzando i valori del segnale di ingresso con i valori già affetti dal calcolo del segnale di uscita.

¹³ In alcuni testi viene definito Voice Encoder.

¹⁴ Altro famoso effetto di modulazione diretta di un segnale audio, che sovrimprime la voce su uno strumento musicale, direttamente parlando attraverso un tubo di plastica posto in bocca dell'esecutore, che modula meccanicamente il suono.

¹⁵ In termini controllistici e elettronici, un'ottava rappresenta uno sweep di frequenze che varia in multipli di due. Questo nome deriva dalla notazione musicale, in quanto una nota di un'ottava superiore alla fondamentale, è di frequenza doppia rispetto alla frequenza della nota fondamentale. È una notazione che si contrappone alla classica “decade o logaritmica”.

Hz a 4 KHz. È ovvio che, più stretta sarà la larghezza di banda, più selettivi saranno i filtri e quindi il range di frequenze sarà più accurato nella selezione. Ora si supponga che un segnale proveniente da uno strumento musicale, venga passato attraverso questo banco di filtri. Il segnale è applicato in parallelo ad ognuno di questi, allo stesso tempo e l'uscita dei filtri viene mandata ad un elemento di controllo dell'ampiezza, che prende il nome di VCA (Voltage Controlled Amplifier), prima di ricombinare l'intero segnale. Non si tratta di un filtraggio come in un normale equalizzatore grafico, piuttosto si tratta di un sistema di attenuazione controllabile, che normalmente è spento.

L'applicazione di un controllo a questi VCA, causerà una differente trattazione delle frequenze che sono affette dal filtraggio del segnale, prima di arrivare all'uscita:

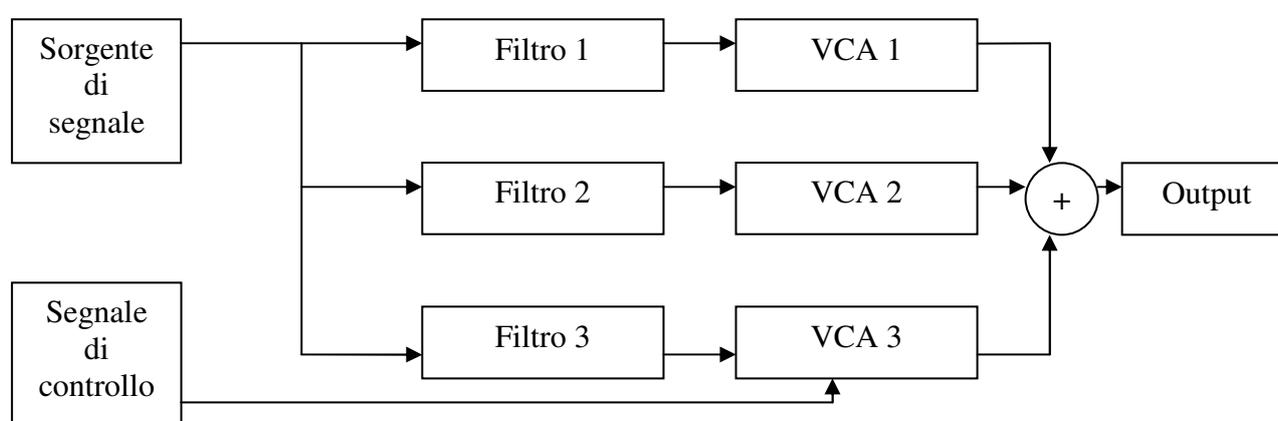


Figura 29: Schema di un Vocoder base con applicazione del controllo sul terzo VCA

Ora supponiamo di avere un altro segnale che arriva da un'altra sorgente, ad esempio un microfono. Questo segnale viene passato ancora una volta dal banco di filtri, ma in questo caso, più che passare direttamente dai VCA, il segnale filtrato, verrà usato per controllare il guadagno dei VCA. Nei Vocoder analogici, questa operazione è ottenuta semplicemente rettificando il segnale di uscita dai filtri, in modo da ottenere una tensione DC, la cui ampiezza è proporzionale alla quantità del range di frequenze, presenti nello spettro del segnale di controllo. In sostanza, la DC Control Voltage, associata ad ogni filtro del banco di controllo, è applicata ai VCA associati al banco dei filtri di sorgente, in modo che l'energia totale dello spettro del segnale di controllo, sia in grado di controllare l'energia totale dello spettro del segnale sorgente. In altre parole, gli spettri dei due segnali, sono separati da qualche grado di libertà e la ricostruzione dello spettro del segnale sorgente è contingente al peso relativo delle porzioni di spettro del segnale di controllo, cioè l'involuppo formante caratteristico del segnale di controllo è sovrimposto allo spettro di sorgente. Quando il VoCoding è svolto nel dominio digitale (come nel caso del software Cool Edit Pro), i

filtri sono digitalmente sostituiti dalle trasformate di Fourier del segnale, in modo che solo le componenti frequenziali del segnale di sorgente vengono affette dal segnale di controllo.

4.3.2 Realizzazione del LFOCoder

Dall'idea principale, di come è stato realizzato un Vocoder, abbiamo deciso di trarre alcune soluzioni algoritmiche, per la realizzazione di un nuovo effetto di modulazione diretta su un segnale audio arbitrario: l'LFOCoder che significa Low Frequency Oscillator Coder. L'idea di base è sostanzialmente basata sulla suddivisione del segnale in tre parti tramite filtraggio, la cui frequenza di centro banda è determinata dall'utente, con una larghezza di banda stabilita in ottave (è lo stesso schema di un Vocoder). La novità fondamentale che sta alla base di questo effetto è che, il segnale di controllo che dovrebbe arrivare da un microfono, in realtà non arriva da esso, ma da tre differenti oscillatori a frequenza sub – sonica (LFO) che pilotano i VCA. Questo permette all'utente di questo effetto, di usare tre differenti tipi di forme d'onda, per il controllo dello spettro del segnale, in modo da esplorare diversi tipi di sonorità a seconda della LFW scelta. Si tratta quindi di un effetto che miscela un triplo Tremolo con tre filtri passa – banda, ed è un sistema che permette da un semplice segnale di chitarra elettrica o di basso elettrico, di ricavare dei suoni tipo sintetizzatore.

4.3.3 Analisi dei metodi di filtraggio usati dagli effetti di modulazione diretta

Apriamo ora una parentesi sui metodi di filtraggio utilizzati da questo oggetto. LFOCoder utilizza tre filtri di Butterworth IIR¹⁶ passa – banda del quarto ordine, con la frequenza di centro banda stabilita dall'utente, mentre per l'eliminazione dei sub – armonici il Tremolo usa un filtro del nono ordine con cut – off vicino a 20 Hz. Si è scelto di utilizzare questo tipo di filtro piuttosto che altre metodologie perché, esso permette di avere la migliore approssimazione della serie di Taylor della risposta del filtro passa – basso ideale a $\Omega=0$ e a $\Omega=\infty$. Per ogni ordine N del filtro, la risposta in ampiezza ha 2N-1 derivate nulle in queste locazioni (risposta completamente piatta), mentre:

$$|H(j\Omega)| = \frac{1}{\sqrt{2}} \text{ per } \Omega^{17}=1 \quad (7)$$

¹⁶ Acronimo di Infinite Impulse Response. Si tratta di filtri digitali caratterizzati da una risposta impulsiva infinita. Sono contrapposti ai filtri FIR Finite Impulse Response, che presentano una maggiore precisione di filtraggio, ma a scapito di richiedere capacità di calcolo maggiori.

¹⁷ Si utilizza questo simbolo per definire la frequenza normalizzata. Si tratta di una frequenza che ha un massimo in 1 e viene calcolata con la formula:

cioè si ha una risposta monotona decrescente. Per svolgere questa funzione con il software Matlab, si utilizza una funzione chiamata *butter* che permette di mettere in una matrice i valori del numeratore e del denominatore della funzione di trasferimento che costituiscono la funzione di trasferimento digitale del filtro (in funzione delle trasformate zeta):

$$H(z) = \frac{B(z)}{A(z)} = \frac{b(1) + b(2)z^{-1} + \dots + b(n+1)z^{-n}}{1 + a(2)z^{-1} + \dots + a(n+1)z^{-n}} \quad (8)$$

L'algoritmo utilizzato dal metodo di filtraggio secondo Butterworth è:

1. Ricava il guadagno, gli zeri e i poli, della funzione di trasferimento del filtro passa – basso prototipo, calcolato con la funzione *buttap*.
2. Converti queste quantità con la trasformazione nello spazio degli stati¹⁸.
3. Esegui la trasformazione da filtro passa – basso a filtro passa – banda, utilizzando il cambiamento dello spazio degli stati con le frequenze di cut – off settate per il filtro.
4. Applica il metodo della trasformazione bilineare (vedi più avanti nel testo) con il prewarping della frequenza.
5. Converti lo stato degli spazi della funzione di trasferimento o nella forma zeri – poli – guadagno.

La trasformazione bilineare che viene applicata al punto 4 dell'algoritmo, è il metodo più comune per l'implementazione di questi filtri, ed è associata ad una operazione matematica che permette di effettuare il mapping di una variabile matematica in un'altra, in particolare si passa dal dominio z a quello s con la seguente formula:

$$\frac{\text{currentfrequency}}{\text{datasamplerate}}$$

$$2$$

¹⁸ È lo spazio ottenuto dalla combinazione delle seguenti equazioni in n : $x[n+1]=Ax[n]+Bu[n]$ e $y[n]=Cx[n]+Du[n]$.

$$H(z) = H(s) \Big|_{s=2f_s \frac{z-1}{z+1}} \quad (9)$$

Questa trasformazione, mappa l'asse $j\Omega$ ripetutamente attorno al cerchio di raggio unitario ($e^{j\omega}$), quindi i limiti degli assi non sono più da $-\infty$ a $+\infty$ ma da $-\pi$ a $+\pi$:

$$\omega = 2 \tan^{-1} \left(\frac{\Omega}{2f_s} \right) \quad (10)$$

Il metodo di trasformazione suddetto, è implementato in Matlab secondo due algoritmi diversi, a seconda che si tratti del passaggio di parametri alla funzione di trasformazione tramite la forma zeri – poli – guadagno o la forma nello spazio degli stadi.

4.4 Sviluppo del Direct Modulation Tools System con Matlab

Effettuiamo ora lo sviluppo principale del Direct Modulation Tools con il software GUIDE di Matlab. Per effettuare l'interazione di questo software con l'environment Nova – Mod Tools e il precedente Signal Acquisition System, sarà necessario utilizzare un file di interscambio tra le applicazioni, per passare le informazioni relative al segnale acquisito. L'utilizzo di un file esterno inoltre, permette di velocizzare le operazioni di manipolazione sul segnale, perché non se ne richiede un'ulteriore acquisizione, ma piuttosto un semplice passaggio di variabili. Il file utilizzato sarà *xchangedata.mat*, un file in formato proprietario Matlab, che consente di contenere tutte le variabili salvate precedentemente sul workspace dell'applicazione. Dalla schermata principale di Nova – Mod Tools, andiamo a settare il callback relativo al pulsante Direct Modulation Tools con il seguente codice:

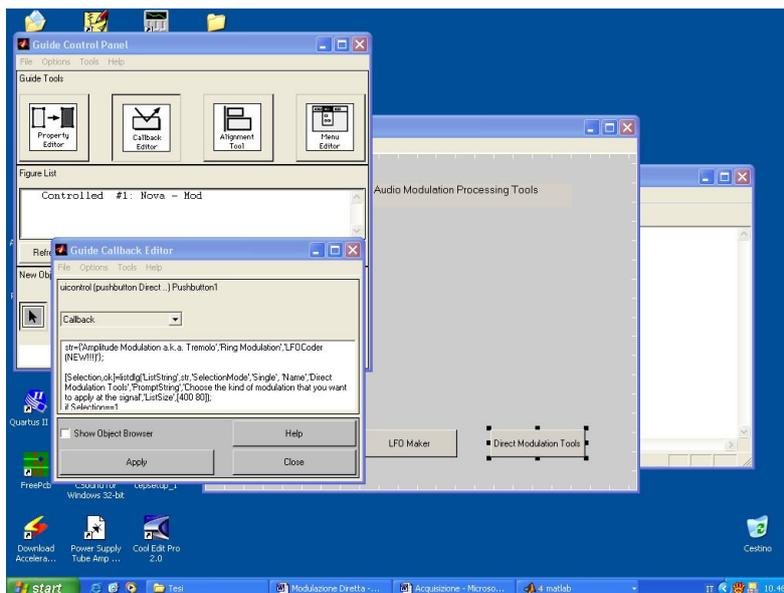


Figura 30: Inserimento del callback per il pulsante Direct Modulation Tools

```
str={'Amplitude Modulation a.k.a. Tremolo','Ring Modulation','LFOCoder (NEW!!!)'};
[Selection,ok]=listdlg('ListString',str,'SelectionMode','Single','Name','Direct
Modulation Tools','PromptString','Choose the kind of modulation that you want to
apply at the signal','ListSize',[400 80]);
if ok==1
if Selection==1
tremolo;
elseif Selection==2
ringer;
elseif Selection==3
lfocoder;
end
elseif ok==0
pause(0.1);
end
```

Il seguente listato (relativamente semplice), esegue la definizione di un vettore *str*, che avrà la funzione di contenere le frasi della list box che apparirà a video quando è premuto il pulsante associato; riceverà il valore scelto dalla listbox, dopo averla creata, ed esegue un controllo per avviare l'applicazione di modulazione diretta del segnale a seconda della scelta svolta. La listbox che appare una volta premuto il pulsante, ha più o meno la seguente forma:

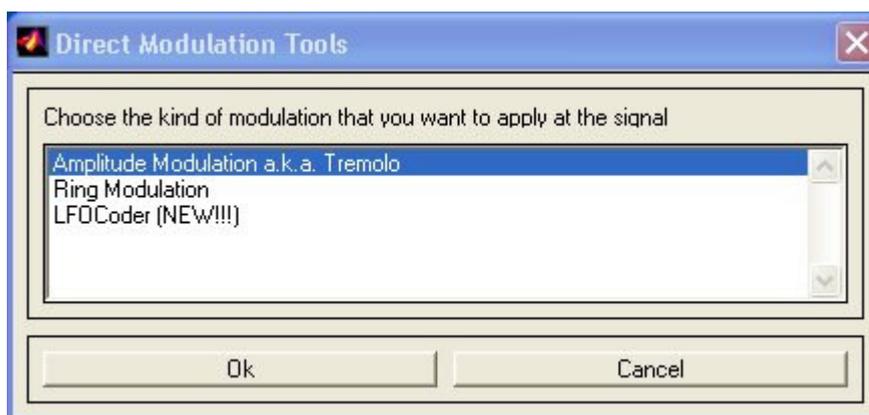


Figura 31: Listbox di selezione di applicazione effetti

Analizziamo ora i controlli messi a disposizione dalle applicazioni che consentono l'esecuzione della modulazione diretta del segnale audio:

- Tremolo e Ring Modulator (i controlli di queste due applicazioni sono praticamente identici, come identico e il loro codice a meno di qualche variazione specifica):

- Plot Dry¹⁹ Signal: è il pulsante che consente di aprire una figura esterna dal titolo "Signal Plot", la quale conterrà tutti i plot relativi all'applicazione, ed esegue il plot del segnale originale, in valori normalizzati. Separando i plot in un'altra figura, rispetto alla principale (dove ci sono i controlli), permette di snellire l'esecuzione delle operazioni
- Apply Tremolo/Ring Modulation: a seconda dei casi è il pulsante che dà l'avvio a Matlab, per l'esecuzione dell'algoritmo di modulazione diretta del segnale. Alla sua pressione, compare una waitbar che progredisce al progredire del compimento dell'algoritmo e al termine visualizza il plot del segnale in valori normalizzati, sulla figura "Signal Plot"
- Save result on a file: è il pulsante che consente di salvare il risultato dell'applicazione dell'algoritmo su file. Come per il Signal Acquisition System, esso consente di salvare in formato Microsoft PCM WAVE FORMAT con i parametri relativi al segnale originale o in formato Motion Picture Expert Group (MPEG1 – Audio) Layer 3, con un livello di bitrate a 256 Kbps
- Credits: è il pulsante che visualizza le informazioni sul creatore dell'applicazione

¹⁹ Si utilizza la notazione Dry, per un segnale senza manipolazione e Wet per il segnale trattato.

- Close: è il pulsante che invia a Matlab un comando di “close get current callback figure” e termina l’applicazione, rilasciando la memoria agli indirizzi relativi all’applicazione avviata
- Source of LFW: è il popup menù che consente di avere due scelte di sorgente della forma d’onda che dovrà andare ad eseguire il processo di modulazione del segnale: la prima scelta “LFO simulated previously and saved on a file”, consente di ricavare da un file con estensione *.lfo*, creato precedentemente con il tool LFO – Maker di Nova – Mod, i parametri di un LFO simulato correttamente e salvato su file; la seconda scelta “Build my own (this will open LFO – Maker)”, lancia un’istanza di avvio dell’applicazione LFO – Maker, la quale consentirà di creare la propria forma d’onda, per il processo di modulazione
- Actual LFW: è il controllo statico che mostra all’utente, la forma d’onda corrente, utilizzata per il processo di modulazione. I suoi valori rientrano nelle possibilità offerte da LFO – Maker
- Actual Rate: è il controllo statico che mostra all’utente, la frequenza di oscillazione dell’LFO attualmente simulato. I suoi valori rientrano nelle possibilità offerte da LFO – Maker
- Actual Depth²⁰: è il controllo statico che mostra all’utente, l’ampiezza di oscillazione dell’LFO attualmente simulato. I suoi valori rientrano nelle possibilità offerte da LFO – Maker
- Play Dry Signal: è il pulsante che consente il playback del segnale acquisito, senza l’applicazione di nessun effetto. Alla pressione del pulsante, sullo schermo appare una waitbar che progredisce al progredire dell’avanzamento del playback del segnale
- Play Wet Signal: è il pulsante che consente il playback del segnale acquisito, con l’applicazione dell’effetto e senza il mixing con il segnale Dry²¹. Alla pressione del

²⁰ Per Depth si intende profondità di modulazione dell’effetto. In questo caso, il controllo di profondità è associato a quanta modulazione vogliamo applicare al segnale e di conseguenza agisce sul parametro relativo all’ampiezza dell’LFO.

²¹ Negli effetti di modulazione diretta tipo Tremolo o Ring Modulator, non ha senso parlare di mixing, dal momento che si va a modificare l’involuppo del segnale e il segnale Dry diventa poi praticamente inutile all’ascolto, anzi, alcune volte tenderebbe a sovrastare l’effetto.

pulsante, sullo schermo appare una waitbar che progredisce al progredire dell'avanzamento del playback del segnale

- LFOCoder:

- Plot Dry Signal: è il pulsante che consente di aprire una figura esterna dal titolo “Signal Plot”, la quale conterrà tutti i plot relativi all'applicazione, ed esegue il plot del segnale originale, in valori normalizzati
- Plot Filter Response: è il pulsante che consente di aprire tre finestre che mostrano la risposta in frequenza e in fase (a frequenza normalizzata sull'asse x) per ogni singolo filtro, ricavando i valori necessari di filtraggio per l'operazione di modulazione
- Apply LFOCoder: a seconda dei casi è il pulsante che dà l'avvio a Matlab, per l'esecuzione dell'algoritmo di modulazione diretta del segnale. Alla sua pressione, compare una waitbar che progredisce al progredire del compimento dell'algoritmo e al termine visualizza il plot del segnale in valori normalizzati, sulla figura “Signal Plot”
- Save result on a file: è il pulsante che consente di salvare il risultato dell'applicazione dell'algoritmo su file. Come per il Signal Acquisition System, esso consente di salvare in formato Microsoft PCM WAVE FORMAT con i parametri relativi al segnale originale o in formato Motion Picture Expert Group (MPEG1 – Audio) Layer 3, con un livello di bitrate a 256 Kbps
- Credits: è il pulsante che visualizza le informazioni sul creatore dell'applicazione
- Close: è il pulsante che invia a Matlab un comando di “close get current callback figure” e termina l'applicazione, rilasciando la memoria agli indirizzi relativi all'applicazione avviata
- Source of LFW x: è il popup menù che consente di avere due scelte di sorgente della forma d'onda che dovrà andare ad eseguire il processo di modulazione del segnale: la prima scelta “LFO simulated previously and saved on a file”, consente di ricavare da un file con estensione *.lfo*, creato precedentemente con il tool LFO – Maker di Nova – Mod, i parametri di un LFO simulato correttamente e salvato su file; la seconda scelta “Build my own (this

will open LFO – Maker)”, lancia un’istanza di avvio dell’applicazione LFO – Maker, la quale consentirà di creare la propria forma d’onda, per il processo di modulazione. In questo caso, siccome l’applicazione utilizza tre differenti LFW, x corrisponde alla x -esima LFW considerata

- Actual LFW x : è il controllo statico che mostra all’utente, la forma d’onda corrente, utilizzata per il processo di modulazione. I suoi valori rientrano nelle possibilità offerte da LFO – Maker. In questo caso, siccome l’applicazione utilizza tre differenti LFW, x corrisponde alla x -esima LFW considerata
- Actual Rate x : è il controllo statico che mostra all’utente, la frequenza di oscillazione dell’LFO attualmente simulato. I suoi valori rientrano nelle possibilità offerte da LFO – Maker. In questo caso, siccome l’applicazione utilizza tre differenti LFW, x corrisponde alla x -esima LFW considerata
- Actual Depth x : è il controllo statico che mostra all’utente, l’ampiezza di oscillazione dell’LFO attualmente simulato. I suoi valori rientrano nelle possibilità offerte da LFO – Maker. In questo caso, siccome l’applicazione utilizza tre differenti LFW, x corrisponde alla x -esima LFW considerata
- Play Dry Signal: è il pulsante che consente il playback del segnale acquisito, senza l’applicazione di nessun effetto. Alla pressione del pulsante, sullo schermo appare una waitbar che progredisce al progredire dell’avanzamento del playback del segnale
- Play Wet Signal: è il pulsante che consente il playback del segnale acquisito, con l’applicazione dell’effetto e senza il mixing con il segnale Dry. Alla pressione del pulsante, sullo schermo appare una waitbar che progredisce al progredire dell’avanzamento del playback del segnale
- Frequency of intervention of the bandpass filter x : è il controllo editabile dall’utente, che consente di stabilire la frequenza di centro banda dell’ x -esimo filtro associato alla x -esima LFW che piloterà l’ x -esimo VCA²²

²² In questo caso simulati dagli LFO.

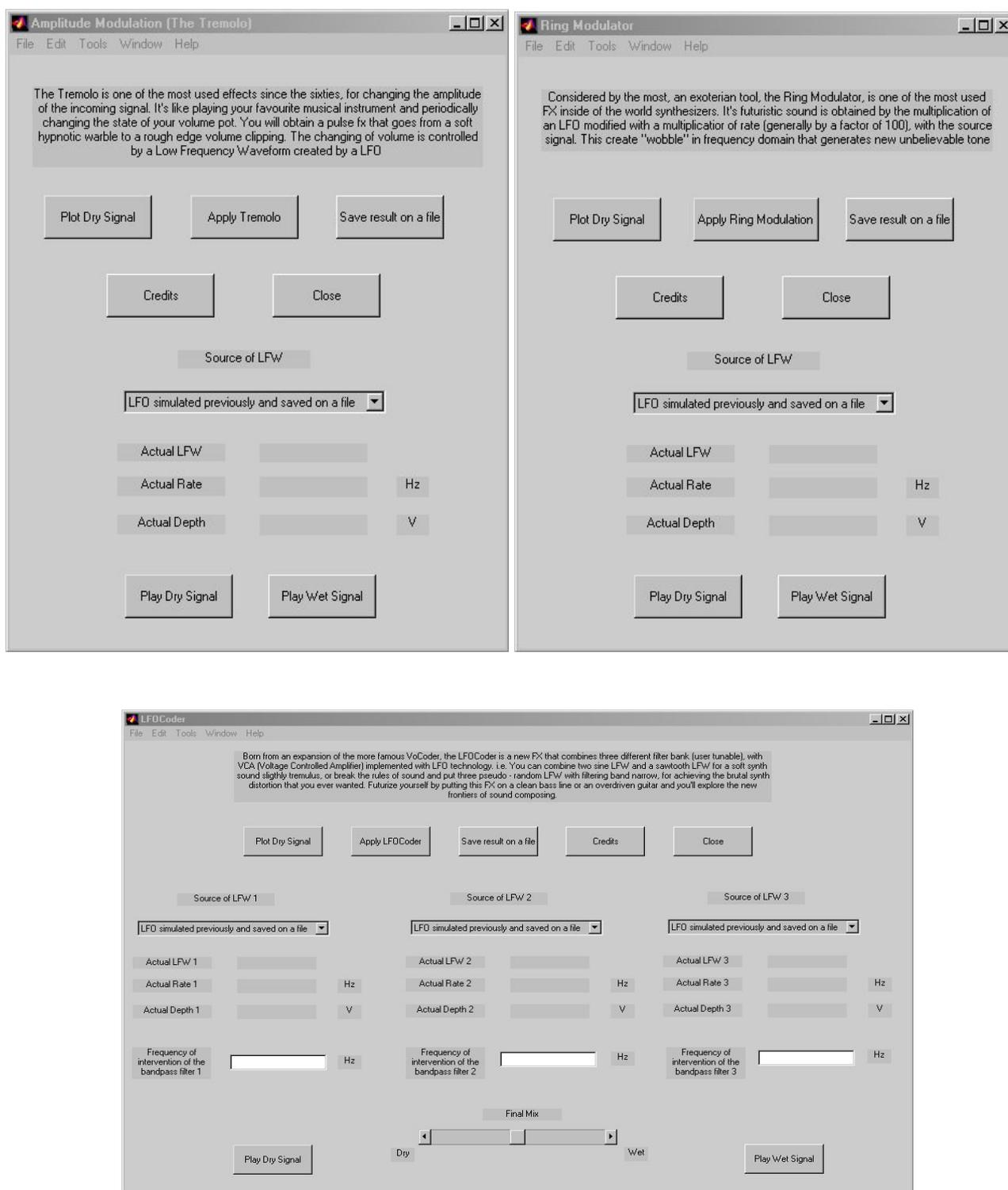


Figura 32: Forma delle applicazioni che consentono la modulazione diretta del segnale

Analizziamo ora il codice che gestisce i callback di queste applicazioni, considerando prima l'applicazione Tremolo *fcnTremolo.m* e *fcnRinger.m* (l'applicazione Ring Modulator è uguale tranne in alcuni punti che verranno poi specificati) e poi l'LFOCoder *fcnLFOCoder*. Si ricorda che il codice di queste applicazioni, è disponibile sul CD-ROM allegato alla tesi, nella directory `../Codice/Matlab` con estensione `.m`.

Questa è la parte di dichiarazione delle variabili globali da usare per l'applicazione e si ricavano i dati del segnale acquisito, tramite il file di interscambio utilizzando dall'environment *xchangedata.mat*.

```
%Function File esterno per i callback del tremolo
%
%written by Zambelli Cristian
function fcnTremolo(action)
global data x lfwtype rsst wall rate amp phase offset damping rectific h1 lfw am out dc;
data=load('xchangedata.mat');
```

Qui comincia la consueta switchyard programming. Questo case permette di ricavare il segnale audio acquisito e di plottarlo in una nuova figura con i valori normalizzati nell'intervallo [-1;+1).

```
switch(action)
case 'signal'
    t=( [1:data.count]/data.samplerate);
    x=data.y;
    h=waitbar(0,'Preparing the Dry Signal...');
    for i=1:100,
        waitbar(i/100);
    end
    close(h);
    if data.channel==1
        if data.quantize==8
            z=(x-128)/128;
        elseif data.quantize==16
            z=x/32768;
        end
    elseif data.channel==2
        if data.quantize==8
            z=(x-128)/128;
        elseif data.quantize==16
            z=x/32768;
        end
    end
    h1=figure('Name','Signal Plot','NumberTitle','off');
    subplot(311);
    plot(t,z);
    title('Dry Signal');
    ylabel('Normalized Values');
    if data.channel==1
```

```

        legend('Monaural Signal');
elseif data.channel==2
    legend('Left Channel','Right Channel');
end

```

Questo case permette di effettuare il playback del segnale senza l'applicazione dell'effetto, tramite la funzione *sound*. Così come per la funzione usata nel Signal Acquisition System, si inserisce una waitbar che progredisce al progredire del playback del file.

```

case 'playdry'
    tr=data.count/data.samplerate;
    inc=0.01;
    texe=clock;soundsc(x,data.samplerate,data.quantize);e=etime(clock,texe);
    texe2=clock;h=waitbar(0,'Playback time (Dry)...');e2=etime(clock,texe2);
    for i=0:inc:(tr-e-e2),
        waitbar(i/(tr-e-e2));
        pause(inc);
    end
    close(h);

```

Questo case permette di effettuare il playback del segnale con l'applicazione dell'effetto, tramite la funzione *sound*. Così come per la funzione usata nel Signal Acquisition System, si inserisce una waitbar che progredisce al progredire del playback del file.

```

case 'playwet'
    tr=data.count/data.samplerate;
    inc=0.01;
    texe=clock;soundsc(out,data.samplerate,data.quantize);e=etime(clock,texe);
    texe2=clock;h=waitbar(0,'Playback time (Wet)...');e2=etime(clock,texe2);
    for i=0:inc:(tr-e-e2),
        waitbar(i/(tr-e-e2));
        pause(inc);
    end
    close(h);

```

Questo case permette di ricavare le informazioni relative alla LFW da utilizzare per l'operazione di modulazione. Se il popup menù relativo alla sorgente di acquisizione della LFW è di valore pari a 1, allora apri una dialog box che permette di scegliere il file con estensione *.lfo* e ne ricava tutti i parametri per poi eseguirne il plot, altrimenti lancia un'istanza di LFO – Maker per la creazione della LFW. L'unica differenza in questo caso tra Tremolo e Ring Modulator, consiste nella moltiplicazione della frequenza di oscillazione di un valore pari a 100.

```

case 'getlfo'
    popup1handle=findobj(gcf,'Tag','PopupMenu1');
    popup1=get(popup1handle,'Value');
    axes2handle=findobj(gcf,'Tag','Axes2');
    if popup1==1
        [filename,filepath]=uigetfile('*.lfo','Get LFW from which file?');
        path=[filepath];
        name=[filename];
        fid=fopen(strcat(path,name),'r');
        lfwtype=fscanf(fid,'%s\n%');
        rsst=fscanf(fid,'%s\n%');
        wall=fscanf(fid,'%s\n%');
        rate=fscanf(fid,'%s\n%');
        amp=fscanf(fid,'%s\n%');
        phase=fscanf(fid,'%s\n%');
        offset=fscanf(fid,'%s\n%');
        damping=fscanf(fid,'%s\n%');
        rectific=fscanf(fid,'%s\n%');
        status=fclose(fid);
        h=waitbar(0,'Gathering LFO Parameters. Please Wait...');
        t=(1:data.count)/data.samplerate);
    x=data.y;
    dc=str2num(offset);
    am=str2num(amp);
    damp=str2num(damping);
    r=str2num(rate);
    p=str2num(phase);
    static=findobj(gcf,'Tag','StaticText4');
    set(static,'String',rate);
    static1=findobj(gcf,'Tag','StaticText5');
    set(static1,'String',amp);
    if strcmp(lfwtype,'sin')
        stati=findobj(gcf,'Tag','StaticText7');
        set(stati,'String','Sine');
        if strcmp(rsst,'1')
            g=dc+am*exp(-(t*damp)).*sin(2*pi*r*t+p);
            a=int32(g);
        elseif strcmp(rsst,'0')
            a=dc+am*exp(-(t*damp)).*sin(2*pi*r*t+p);
        end
    if strcmp(wall,'1')
        b=a+0.1*sin(2*pi*50*t);
    elseif strcmp(wall,'0')
        b=a;
    end
end

```

```

if strcmp(rectif,'1')
    lfw=abs(b);
elseif strcmp(rectif,'0')
    lfw=b;
end
elseif strcmp(lfwtype,'cos')
    stati=findobj(gcf,'Tag','StaticText7');
    set(stati,'String','Cosine');
    if strcmp(rsst,'1')
        g=dc+am*exp(-(t*damp)).*cos(2*pi*r*t+p);
        a=int32(g);
    elseif strcmp(rsst,'0')
        a=dc+am*exp(-(t*damp)).*cos(2*pi*r*t+p);
    end
    if strcmp(wall,'1')
        b=a+0.1*sin(2*pi*50*t);
    elseif strcmp(wall,'0')
        b=a;
    end
    if strcmp(rectif,'1')
        lfw=abs(b);
    elseif strcmp(rectif,'0')
        lfw=b;
    end
elseif strcmp(lfwtype,'tri')
    stati=findobj(gcf,'Tag','StaticText7');
    set(stati,'String','Triangular');
    if strcmp(rsst,'1')
        g=dc+am*exp(-(t*damp)).*sawtooth(2*pi*r*t,p+0.5);
        a=int32(g);
    elseif strcmp(rsst,'0')
        a=dc+am*exp(-(t*damp)).*sawtooth(2*pi*r*t,p+0.5);
    end
    if strcmp(wall,'1')
        b=a+0.1*sin(2*pi*50*t);
    elseif strcmp(wall,'0')
        b=a;
    end
    if strcmp(rectif,'1')
        lfw=abs(b);
    elseif strcmp(rectif,'0')
        lfw=b;
    end
elseif strcmp(lfwtype,'squ')
    stati=findobj(gcf,'Tag','StaticText7');
    set(stati,'String','Square');
    if strcmp(rsst,'1')

```

```

        g=dc+am*exp(-(t*damp)).*square(2*pi*r*t,p);
        a=int32(g);
elseif strcmp(rsst,'0')
        a=dc+am*exp(-(t*damp)).*square(2*pi*r*t,p);
end
if strcmp(wall,'1')
        b=a+0.1*sin(2*pi*50*t);
elseif strcmp(wall,'0')
        b=a;
end
if strcmp(rectif,'1')
        lfw=abs(b);
elseif strcmp(rectif,'0')
        lfw=b;
end
elseif strcmp(lfwtype,'exp')
        stati=findobj(gcf,'Tag','StaticText7');
        set(stati,'String','Exponential Sine');
        if strcmp(rsst,'1')
                g=dc+am*exp(-(t*damp)).*exp(-sin(2*pi*r*t+p));
                a=int32(g);
elseif strcmp(rsst,'0')
                a=dc+am*exp(-(t*damp)).*exp(-sin(2*pi*r*t+p));
end
if strcmp(wall,'1')
        b=a+0.1*sin(2*pi*50*t);
elseif strcmp(wall,'0')
        b=a;
end
if strcmp(rectif,'1')
        lfw=abs(b);
elseif strcmp(rectif,'0')
        lfw=b;
end
elseif strcmp(lfwtype,'saw')
        stati=findobj(gcf,'Tag','StaticText7');
        set(stati,'String','Sawtooth');
        if strcmp(rsst,'1')
                g=dc+am*exp(-(t*damp)).*sawtooth(2*pi*r*t,p);
                a=int32(g);
elseif strcmp(rsst,'0')
                a=dc+am*exp(-(t*damp)).*sawtooth(2*pi*r*t,p);
end
if strcmp(wall,'1')
        b=a+0.1*sin(2*pi*50*t);
elseif strcmp(wall,'0')
        b=a;

```

```

end
if strcmp(rectif,'1')
    lfw=abs(b);
elseif strcmp(rectif,'0')
    lfw=b;
end
elseif strcmp(lfwtype,'ran')
    stati=findobj(gcf,'Tag','StaticText7');
    set(stati,'String','Pseudo Random');
    if strcmp(rsst,'1')
        g=dc+am*exp(-
(t*damp)).*sin(2*pi*rand(1)*10*r*t+p)+sin(4*pi*rand(1)*10*r*t+p)+sin(8*pi*rand(1)*10*r*t+
p)+sin(16*pi*rand(1)*10*r*t+p)+sin(32*pi*rand(1)*10*r*t+p)+sin(64*pi*rand(1)*10*r*t+p);
        a=int32(g);
    elseif strcmp(rsst,'0')
        a=dc+am*exp(-
(t*damp)).*sin(2*pi*rand(1)*10*r*t+p)+sin(4*pi*rand(1)*10*r*t+p)+sin(8*pi*rand(1)*10*r*t+
p)+sin(16*pi*rand(1)*10*r*t+p)+sin(32*pi*rand(1)*10*r*t+p)+sin(64*pi*rand(1)*10*r*t+p);
    end
    if strcmp(wall,'1')
        b=a+0.1*sin(2*pi*50*t);
    elseif strcmp(wall,'0')
        b=a;
    end
    if strcmp(rectif,'1')
        lfw=abs(b);
    elseif strcmp(rectif,'0')
        lfw=b;
    end
end
for i=1:100,
    waitbar(i/100);
end
close(h);
figure(h1);
subplot(312);
plot(t,lfw);
title('LFW Modulator');
ylabel('Volts');
elseif popup1==2
    lfomaker;
end

```

Questo case permette l'applicazione del Tremolo al segnale. Si ricavano le informazioni del segnale audio acquisito, si normalizzano i valori della LFW e del segnale in intervalli compatibili per

l'esecuzione del processo di modulazione, si inizializzano i vettori che dovranno contenere i dati di uscita del segnale modulato, si esegue la modulazione seguendo il processo di allocazione degli spazi lineari e si effettua il plot del segnale di uscita in valori modulati, dopo previo filtraggio di Butterworth per l'eliminazione dei sub – armonici (non presente nel Ring Modulator).

```

case 'apply'
    t=( [1:data.count]/data.samplerate);
    l=length(t);
    n=1:l;
    if data.channel==1
        if data.quantize==8
            in=(x-128)/128;
        elseif data.quantize==16
            in=x/32768;
        end
    elseif data.channel==2
        if data.quantize==8
            in=(x-128)/128;
        elseif data.quantize==16
            in=x/32768;
        end
    end
    if strcmp(rsst,'1')
        partial=double(lfw);
        normlfw=double(partial/am+dc);
        normlfw=reshape(normlfw,l,1);
    elseif strcmp(rsst,'0')
        normlfw=lfw/am+dc;
        normlfw=reshape(normlfw,l,1);
    end
    if data.channel==1
        h=waitbar(0,'Now Tremolo is being applied (this could take a few minutes). Please
Wait...');
        for i=1:100
            waitbar(i/100);
        end
        in(n)=in(n).*normlfw(n);
    elseif data.channel==2
        h=waitbar(0,'Now Tremolo is being applied (this could take a few minutes). Please
Wait...');
        for i=1:100
            waitbar(i/100);
        end
        in(n,1)=in(n,1).*normlfw(n);
        in(n,2)=in(n,2).*normlfw(n);
    end
end

```

```

end
close(h);
if data.channel==1
    h=waitbar(0,'Killing the Sub - Harmonics generated by Tremolo...');
    [b,a]=butter(4,30/(data.samplerate),'high');
    outc=filter(b,a,in);
    out=reshape(outc,1,1);
    for i=1:100,
        waitbar(i/100);
    end
    close(h);
    figure(h1);
    subplot(313);
    plot(t,out);
    title('Wet Signal');
    ylabel('Normalized Values');
    xlabel('Seconds');
    legend('Monaural Signal');
elseif data.channel==2
    outd=[in(:,1),in(:,2)];
    h=waitbar(0,'Killing the Sub - Harmonics generated by Tremolo...');
    [b,a]=butter(4,30/(data.samplerate),'high');
    outc=filter(b,a,outd);
    out=reshape(outc,1,2);
    for i=1:100,
        waitbar(i/100);
    end
    close(h);
    figure(h1);
    subplot(313);
    plot(t,out);
    title('Wet Signal');
    ylabel('Normalized Values');
    xlabel('Seconds');
    legend('Left Channel','Right Channel');
end

```

Questo case, permette di salvare il risultato del processo di modulazione diretta del segnale su un file in formato Microsoft PCM WAVE FORMAT o Motion Picture Expert Group (MPEG1) Layer 3, conservando i parametri di samplerate e profondità di quantizzazione del segnale originale. Per l'encoding si utilizza il software LAME con una modalità di "encoding Extreme" a 256 Kbps con segnale ricampionato a 44.1 KHz.

```
case 'saveasf'
```

```

    button3=questdlg('Would you like to save this signal for further processing?','Save
file...','Microsoft PCM WAVE FORMAT','Motion Picture Expert Group (MPEG-1) Layer
3','No','No');
    if strcmp(button3,'Microsoft PCM WAVE FORMAT')
        [newfile,newpath] = uiputfile('*.wav','Save as...');
        newpath1=[newpath];
        newfile1=[newfile];
        h2=waitbar(0,'Saving in Progress... Please Wait');
        wavwrite(out,data.samplerate,data.quantize, strcat(newpath1,newfile1));
        for i=1:100,
            waitbar(i/100);
        end
        close(h2);
    elseif strcmp(button3,'Motion Picture Expert Group (MPEG-1) Layer 3')
        [newfile,newpath] = uiputfile('*.mp3','Save as...');
        newpath1=[newpath];
        newfile1=[newfile];
        wavwrite(r,data.samplerate,data.quantize,'tmpfile.wav');
        string=strcat(newpath1,newfile1,'.mp3');
        if data.channel==1
            h2=waitbar(0,'Encoding and Saving in Progress... Please Wait');
            cmd=['lame --quiet -b 256 --resample 44.1 tmpfile.wav',' ',string];
            dos(cmd);
            for i=1:100,
                waitbar(i/100);
            end
            close(h2);
            delete('tmpfile.wav');
        elseif data.channel==2
            h2=waitbar(0,'Encoding and Saving in Progress... Please Wait');
            cmd=['lame --quiet -b 256 --resample 44.1 -m s tmpfile.wav',' ',string];
            dos(cmd);
            for i=1:100,
                waitbar(i/100);
            end
            close(h2);
            delete('tmpfile.wav');
        end
    end
end

```

Questo case mostra le informazioni relative all'applicazione, utilizzando una help dialog.

```

case 'info'
    h=helpdlg('This is the Tremolo FX inside of Nova - Mod Tools enviroment built by
Zambelli Cristian. For using this feature you need to have pre-acquired the signal with
the Signal Acquisition System','Information');

```

Questo case esegue il termine dell'applicazione, rilasciando l'handle della figura dalla memoria, inviando il comando *close*.

```
case 'close'
    close(gcf);
end
```

Analizziamo ora il codice del LFOCoder:

```
%Function File esterno per i callback del LFOCoder
%
%written by Zambelli Cristian
function fcnLFOCoder(action)
global data x lfwtype rsst wall rate amp phase offset damping rectific h1 lfw1 lfw2 lfw3
am1 am2 am3 out dc1 dc2 dc3 b1 a1 b2 a2 b3 a3;
data=load('xchangedata.mat');
switch(action)
case 'signal'
    t=(1:data.count)/data.samplerate);
    x=data.y;
    h=waitbar(0,'Preparing the Dry Signal...');
    for i=1:100,
        waitbar(i/100);
    end
    close(h);
    if data.channel==1
        if data.quantize==8
            z=(x-128)/128;
        elseif data.quantize==16
            z=x/32768;
        end
    elseif data.channel==2
        if data.quantize==8
            z=(x-128)/128;
        elseif data.quantize==16
            z=x/32768;
        end
    end
    h1=figure('Name','Signal Plot','NumberTitle','off');
    subplot(511);
    plot(t,z);
    title('Dry Signal');
    ylabel('Normalized Values');
```

```

if data.channel==1
    legend('Monaural Signal');
elseif data.channel==2
    legend('Left Channel','Right Channel');
end
case 'playdry'
    tr=data.count/data.samplerate;
    inc=0.01;
    texe=clock;soundsc(x,data.samplerate,data.quantize);e=etime(clock,texe);
    texe2=clock;h=waitbar(0,'Playback time (Dry)...');e2=etime(clock,texe2);
    for i=0:inc:(tr-e-e2),
        waitbar(i/(tr-e-e2));
        pause(inc);
    end
    close(h);
case 'playwet'
    tr=data.count/data.samplerate;
    inc=0.01;
    texe=clock;soundsc(out,data.samplerate,data.quantize);e=etime(clock,texe);
    texe2=clock;h=waitbar(0,'Playback time (Wet)...');e2=etime(clock,texe2);
    for i=0:inc:(tr-e-e2),
        waitbar(i/(tr-e-e2));
        pause(inc);
    end
    close(h);
case 'getlfo1'
    popup1handle=findobj(gcf,'Tag','PopupMenu1');
    popup1=get(popup1handle,'Value');
    axes2handle=findobj(gcf,'Tag','Axes2');
    if popup1==1
        [filename,filepath]=uigetfile('*.lfo','Get LFW from which file?');
        path=[filepath];
        name=[filename];
        fid=fopen(strcat(path,name),'r');
        lfwtype=fscanf(fid,'%s\n%');
        rsst1=fscanf(fid,'%s\n%');
        wall=fscanf(fid,'%s\n%');
        rate=fscanf(fid,'%s\n%');
        amp=fscanf(fid,'%s\n%');
        phase=fscanf(fid,'%s\n%');
        offset=fscanf(fid,'%s\n%');
        damping=fscanf(fid,'%s\n%');
        rectif=fscanf(fid,'%s\n%');
        status=fclose(fid);
        h=waitbar(0,'Gathering LFO 1 Parameters. Please Wait...');
        t=(1:data.count)/data.samplerate);
    x=data.y;

```

```

dc1=str2num(offset);
dc=dc1;
aml=str2num(amp);
am=aml;
damp=str2num(damping);
r=str2num(rate);
p=str2num(phase);
static=findobj(gcf,'Tag','StaticText4');
set(static,'String',rate);
static1=findobj(gcf,'Tag','StaticText5');
set(static1,'String',amp);
if strcmp(lfwtype,'sin')
    stati=findobj(gcf,'Tag','StaticText7');
    set(stati,'String','Sine');
    if strcmp(rsst,'1')
        g=dc+am*exp(-(t*damp)).*sin(2*pi*r*t+p);
        a=int32(g);
    elseif strcmp(rsst,'0')
        a=dc+am*exp(-(t*damp)).*sin(2*pi*r*t+p);
    end
if strcmp(wall,'1')
    b=a+0.1*sin(2*pi*50*t);
elseif strcmp(wall,'0')
    b=a;
end
if strcmp(rectif,'1')
    lfw1=abs(b);
elseif strcmp(rectif,'0')
    lfw1=b;
end
elseif strcmp(lfwtype,'cos')
    stati=findobj(gcf,'Tag','StaticText7');
    set(stati,'String','Cosine');
    if strcmp(rsst,'1')
        g=dc+am*exp(-(t*damp)).*cos(2*pi*r*t+p);
        a=int32(g);
    elseif strcmp(rsst,'0')
        a=dc+am*exp(-(t*damp)).*cos(2*pi*r*t+p);
    end
if strcmp(wall,'1')
    b=a+0.1*sin(2*pi*50*t);
elseif strcmp(wall,'0')
    b=a;
end
if strcmp(rectif,'1')
    lfw1=abs(b);
elseif strcmp(rectif,'0')

```

```

    lfw1=b;
end
elseif strcmp(lfwtype, 'tri')
    stati=findobj(gcf, 'Tag', 'StaticText7');
    set(stati, 'String', 'Triangular');
    if strcmp(rsst, '1')
        g=dc+am*exp(-(t*damp)).*sawtooth(2*pi*r*t, p+0.5);
        a=int32(g);
    elseif strcmp(rsst, '0')
        a=dc+am*exp(-(t*damp)).*sawtooth(2*pi*r*t, p+0.5);
    end
    if strcmp(wall, '1')
        b=a+0.1*sin(2*pi*50*t);
    elseif strcmp(wall, '0')
        b=a;
    end
    if strcmp(rectif, '1')
        lfw1=abs(b);
    elseif strcmp(rectif, '0')
        lfw1=b;
    end
elseif strcmp(lfwtype, 'squ')
    stati=findobj(gcf, 'Tag', 'StaticText7');
    set(stati, 'String', 'Square');
    if strcmp(rsst, '1')
        g=dc+am*exp(-(t*damp)).*square(2*pi*r*t, p);
        a=int32(g);
    elseif strcmp(rsst, '0')
        a=dc+am*exp(-(t*damp)).*square(2*pi*r*t, p);
    end
    if strcmp(wall, '1')
        b=a+0.1*sin(2*pi*50*t);
    elseif strcmp(wall, '0')
        b=a;
    end
    if strcmp(rectif, '1')
        lfw1=abs(b);
    elseif strcmp(rectif, '0')
        lfw1=b;
    end
elseif strcmp(lfwtype, 'exp')
    stati=findobj(gcf, 'Tag', 'StaticText7');
    set(stati, 'String', 'Exponential Sine');
    if strcmp(rsst, '1')
        g=dc+am*exp(-(t*damp)).*exp(-sin(2*pi*r*t+p));
        a=int32(g);
    elseif strcmp(rsst, '0')

```

```

        a=dc+am*exp(-(t*damp)).*exp(-sin(2*pi*r*t+p));
    end
    if strcmp(wall,'1')
        b=a+0.1*sin(2*pi*50*t);
    elseif strcmp(wall,'0')
        b=a;
    end
    if strcmp(rectif,'1')
        lfw1=abs(b);
    elseif strcmp(rectif,'0')
        lfw1=b;
    end
    elseif strcmp(lfwtype,'saw')
        stati=findobj(gcf,'Tag','StaticText7');
        set(stati,'String','Sawtooth');
        if strcmp(rsst,'1')
            g=dc+am*exp(-(t*damp)).*sawtooth(2*pi*r*t,p);
            a=int32(g);
        elseif strcmp(rsst,'0')
            a=dc+am*exp(-(t*damp)).*sawtooth(2*pi*r*t,p);
        end
        if strcmp(wall,'1')
            b=a+0.1*sin(2*pi*50*t);
        elseif strcmp(wall,'0')
            b=a;
        end
        if strcmp(rectif,'1')
            lfw1=abs(b);
        elseif strcmp(rectif,'0')
            lfw1=b;
        end
    elseif strcmp(lfwtype,'ran')
        stati=findobj(gcf,'Tag','StaticText7');
        set(stati,'String','Pseudo Random');
        if strcmp(rsst,'1')
            g=dc+am*exp(-
(t*damp)).*sin(2*pi*rand(1)*10*r*t+p)+sin(4*pi*rand(1)*10*r*t+p)+sin(8*pi*rand(1)*10*r*t+
p)+sin(16*pi*rand(1)*10*r*t+p)+sin(32*pi*rand(1)*10*r*t+p)+sin(64*pi*rand(1)*10*r*t+p);
            a=int32(g);
        elseif strcmp(rsst,'0')
            a=dc+am*exp(-
(t*damp)).*sin(2*pi*rand(1)*10*r*t+p)+sin(4*pi*rand(1)*10*r*t+p)+sin(8*pi*rand(1)*10*r*t+
p)+sin(16*pi*rand(1)*10*r*t+p)+sin(32*pi*rand(1)*10*r*t+p)+sin(64*pi*rand(1)*10*r*t+p);
        end
        if strcmp(wall,'1')
            b=a+0.1*sin(2*pi*50*t);
        elseif strcmp(wall,'0')

```

```

        b=a;
    end
    if strcmp(rectif,'1')
        lfw1=abs(b);
    elseif strcmp(rectif,'0')
        lfw1=b;
    end
end
end
for i=1:100,
    waitbar(i/100);
end
    close(h);
    figure(h1);
subplot(512);
plot(t,lfw1);
title('LFW 1 Modulator');
ylabel('Volts');
elseif popup1==2
    lfomaker;
end
case 'getlfo2'
    popup1handle=findobj(gcf,'Tag','PopupMenu2');
    popup1=get(popup1handle,'Value');
    axes2handle=findobj(gcf,'Tag','Axes2');
    if popup1==1
        [filename,filepath]=uigetfile('*.lfo','Get LFW from which file?');
        path=[filepath];
        name=[filename];
        fid=fopen(strcat(path,name),'r');
        lfwtype=fscanf(fid,'%s\n%');
        rsst2=fscanf(fid,'%s\n%');
        wall=fscanf(fid,'%s\n%');
        rate=fscanf(fid,'%s\n%');
        amp=fscanf(fid,'%s\n%');
        phase=fscanf(fid,'%s\n%');
        offset=fscanf(fid,'%s\n%');
        damping=fscanf(fid,'%s\n%');
        rectific=fscanf(fid,'%s\n%');
        status=fclose(fid);
        h=waitbar(0,'Gathering LFO 2 Parameters. Please Wait...');
        t=( [1:data.count]/data.samplerate);
    x=data.y;
        dc2=str2num(offset);
        dc=dc2;
        am2=str2num(amp);
        am=am2;
        damp=str2num(damping);

```

```

r=str2num(rate);
p=str2num(phase);
static=findobj(gcf,'Tag','StaticText9');
set(static,'String',rate);
static1=findobj(gcf,'Tag','StaticText10');
set(static1,'String',amp);
if strcmp(lfwtype,'sin')
    stati=findobj(gcf,'Tag','StaticText8');
    set(stati,'String','Sine');
    if strcmp(rsst,'1')
        g=dc+am*exp(-(t*damp)).*sin(2*pi*r*t+p);
        a=int32(g);
    elseif strcmp(rsst,'0')
        a=dc+am*exp(-(t*damp)).*sin(2*pi*r*t+p);
    end
    if strcmp(wall,'1')
        b=a+0.1*sin(2*pi*50*t);
    elseif strcmp(wall,'0')
        b=a;
    end
    if strcmp(rectif,'1')
        lfw2=abs(b);
    elseif strcmp(rectif,'0')
        lfw2=b;
    end
elseif strcmp(lfwtype,'cos')
    stati=findobj(gcf,'Tag','StaticText8');
    set(stati,'String','Cosine');
    if strcmp(rsst,'1')
        g=dc+am*exp(-(t*damp)).*cos(2*pi*r*t+p);
        a=int32(g);
    elseif strcmp(rsst,'0')
        a=dc+am*exp(-(t*damp)).*cos(2*pi*r*t+p);
    end
    if strcmp(wall,'1')
        b=a+0.1*sin(2*pi*50*t);
    elseif strcmp(wall,'0')
        b=a;
    end
    if strcmp(rectif,'1')
        lfw2=abs(b);
    elseif strcmp(rectif,'0')
        lfw2=b;
    end
elseif strcmp(lfwtype,'tri')
    stati=findobj(gcf,'Tag','StaticText8');
    set(stati,'String','Triangular');

```

```

    if strcmp(rsst, '1')
        g=dc+am*exp(-(t*damp)).*sawtooth(2*pi*r*t,p+0.5);
        a=int32(g);
    elseif strcmp(rsst, '0')
        a=dc+am*exp(-(t*damp)).*sawtooth(2*pi*r*t,p+0.5);
    end
    if strcmp(wall, '1')
        b=a+0.1*sin(2*pi*50*t);
    elseif strcmp(wall, '0')
        b=a;
    end
    if strcmp(rectif, '1')
        lfw2=abs(b);
    elseif strcmp(rectif, '0')
        lfw2=b;
    end
    elseif strcmp(lfwtype, 'squ')
        stati=findobj(gcf, 'Tag', 'StaticText8');
        set(stati, 'String', 'Square');
        if strcmp(rsst, '1')
            g=dc+am*exp(-(t*damp)).*square(2*pi*r*t,p);
            a=int32(g);
        elseif strcmp(rsst, '0')
            a=dc+am*exp(-(t*damp)).*square(2*pi*r*t,p);
        end
        if strcmp(wall, '1')
            b=a+0.1*sin(2*pi*50*t);
        elseif strcmp(wall, '0')
            b=a;
        end
        if strcmp(rectif, '1')
            lfw2=abs(b);
        elseif strcmp(rectif, '0')
            lfw2=b;
        end
    elseif strcmp(lfwtype, 'exp')
        stati=findobj(gcf, 'Tag', 'StaticText8');
        set(stati, 'String', 'Exponential Sine');
        if strcmp(rsst, '1')
            g=dc+am*exp(-(t*damp)).*exp(-sin(2*pi*r*t+p));
            a=int32(g);
        elseif strcmp(rsst, '0')
            a=dc+am*exp(-(t*damp)).*exp(-sin(2*pi*r*t+p));
        end
        if strcmp(wall, '1')
            b=a+0.1*sin(2*pi*50*t);
        elseif strcmp(wall, '0')

```

```

    b=a;
end
if strcmp(rectif,'1')
    lfw2=abs(b);
elseif strcmp(rectif,'0')
    lfw2=b;
end
elseif strcmp(lfwtype,'saw')
    stati=findobj(gcf,'Tag','StaticText8');
    set(stati,'String','Sawtooth');
    if strcmp(rsst,'1')
        g=dc+am*exp(-(t*damp)).*sawtooth(2*pi*r*t,p);
        a=int32(g);
    elseif strcmp(rsst,'0')
        a=dc+am*exp(-(t*damp)).*sawtooth(2*pi*r*t,p);
    end
if strcmp(wall,'1')
    b=a+0.1*sin(2*pi*50*t);
elseif strcmp(wall,'0')
    b=a;
end
if strcmp(rectif,'1')
    lfw2=abs(b);
elseif strcmp(rectif,'0')
    lfw2=b;
end
elseif strcmp(lfwtype,'ran')
    stati=findobj(gcf,'Tag','StaticText8');
    set(stati,'String','Pseudo Random');
    if strcmp(rsst,'1')
        g=dc+am*exp(-
(t*damp)).*sin(2*pi*rand(1)*10*r*t+p)+sin(4*pi*rand(1)*10*r*t+p)+sin(8*pi*rand(1)*10*r*t+
p)+sin(16*pi*rand(1)*10*r*t+p)+sin(32*pi*rand(1)*10*r*t+p)+sin(64*pi*rand(1)*10*r*t+p);
        a=int32(g);
    elseif strcmp(rsst,'0')
        a=dc+am*exp(-
(t*damp)).*sin(2*pi*rand(1)*10*r*t+p)+sin(4*pi*rand(1)*10*r*t+p)+sin(8*pi*rand(1)*10*r*t+
p)+sin(16*pi*rand(1)*10*r*t+p)+sin(32*pi*rand(1)*10*r*t+p)+sin(64*pi*rand(1)*10*r*t+p);
    end
if strcmp(wall,'1')
    b=a+0.1*sin(2*pi*50*t);
elseif strcmp(wall,'0')
    b=a;
end
if strcmp(rectif,'1')
    lfw2=abs(b);
elseif strcmp(rectif,'0')

```

```

        lfw2=b;
    end
end
for i=1:100,
    waitbar(i/100);
end
    close(h);
    figure(h1);
subplot(513);
plot(t,lfw2);
title('LFW 2 Modulator');
ylabel('Volts');
elseif popup1==2
    lfomaker;
end
case 'getlfo3'
    popup1handle=findobj(gcf,'Tag','PopupMenu3');
    popup1=get(popup1handle,'Value');
    axes2handle=findobj(gcf,'Tag','Axes2');
    if popup1==1
        [filename,filepath]=uigetfile('*.lfo','Get LFW from which file?');
        path=[filepath];
        name=[filename];
        fid=fopen(strcat(path,name),'r');
        lfwtype=fscanf(fid,'%s\n%');
        rsst3=fscanf(fid,'%s\n%');
        wall=fscanf(fid,'%s\n%');
        rate=fscanf(fid,'%s\n%');
        amp=fscanf(fid,'%s\n%');
        phase=fscanf(fid,'%s\n%');
        offset=fscanf(fid,'%s\n%');
        damping=fscanf(fid,'%s\n%');
        rectific=fscanf(fid,'%s\n%');
        status=fclose(fid);
        h=waitbar(0,'Gathering LFO 3 Parameters. Please Wait...');
        t=(1:data.count)/data.samplerate);
    x=data.y;
        dc3=str2num(offset);
        dc=dc3;
        am3=str2num(amp);
        am=am3;
    damp=str2num(damping);
    r=str2num(rate);
        p=str2num(phase);
        static=findobj(gcf,'Tag','StaticText12');
        set(static,'String',rate);
        static1=findobj(gcf,'Tag','StaticText14');

```

```

set(static1, 'String', amp);
if strcmp(lfwtype, 'sin')
    stati=findobj(gcf, 'Tag', 'StaticText11');
    set(stati, 'String', 'Sine');
    if strcmp(rsst, '1')
        g=dc+am*exp(-(t*damp)).*sin(2*pi*r*t+p);
        a=int32(g);
    elseif strcmp(rsst, '0')
        a=dc+am*exp(-(t*damp)).*sin(2*pi*r*t+p);
    end
if strcmp(wall, '1')
    b=a+0.1*sin(2*pi*50*t);
elseif strcmp(wall, '0')
    b=a;
end
if strcmp(rectif, '1')
    lfw3=abs(b);
elseif strcmp(rectif, '0')
    lfw3=b;
end
elseif strcmp(lfwtype, 'cos')
    stati=findobj(gcf, 'Tag', 'StaticText11');
    set(stati, 'String', 'Cosine');
    if strcmp(rsst, '1')
        g=dc+am*exp(-(t*damp)).*cos(2*pi*r*t+p);
        a=int32(g);
    elseif strcmp(rsst, '0')
        a=dc+am*exp(-(t*damp)).*cos(2*pi*r*t+p);
    end
if strcmp(wall, '1')
    b=a+0.1*sin(2*pi*50*t);
elseif strcmp(wall, '0')
    b=a;
end
if strcmp(rectif, '1')
    lfw3=abs(b);
elseif strcmp(rectif, '0')
    lfw3=b;
end
elseif strcmp(lfwtype, 'tri')
    stati=findobj(gcf, 'Tag', 'StaticText11');
    set(stati, 'String', 'Triangular');
    if strcmp(rsst, '1')
        g=dc+am*exp(-(t*damp)).*sawtooth(2*pi*r*t, p+0.5);
        a=int32(g);
    elseif strcmp(rsst, '0')
        a=dc+am*exp(-(t*damp)).*sawtooth(2*pi*r*t, p+0.5);

```

```

end
if strcmp(wall,'1')
    b=a+0.1*sin(2*pi*50*t);
elseif strcmp(wall,'0')
    b=a;
end
if strcmp(rectif,'1')
    lfw3=abs(b);
elseif strcmp(rectif,'0')
    lfw3=b;
end
elseif strcmp(lfwtype,'squ')
    stati=findobj(gcf,'Tag','StaticText11');
    set(stati,'String','Square');
    if strcmp(rsst,'1')
        g=dc+am*exp(-(t*damp)).*square(2*pi*r*t,p);
        a=int32(g);
    elseif strcmp(rsst,'0')
        a=dc+am*exp(-(t*damp)).*square(2*pi*r*t,p);
    end
if strcmp(wall,'1')
    b=a+0.1*sin(2*pi*50*t);
elseif strcmp(wall,'0')
    b=a;
end
if strcmp(rectif,'1')
    lfw3=abs(b);
elseif strcmp(rectif,'0')
    lfw3=b;
end
elseif strcmp(lfwtype,'exp')
    stati=findobj(gcf,'Tag','StaticText11');
    set(stati,'String','Exponential Sine');
    if strcmp(rsst,'1')
        g=dc+am*exp(-(t*damp)).*exp(-sin(2*pi*r*t+p));
        a=int32(g);
    elseif strcmp(rsst,'0')
        a=dc+am*exp(-(t*damp)).*exp(-sin(2*pi*r*t+p));
    end
if strcmp(wall,'1')
    b=a+0.1*sin(2*pi*50*t);
elseif strcmp(wall,'0')
    b=a;
end
if strcmp(rectif,'1')
    lfw3=abs(b);
elseif strcmp(rectif,'0')

```

```

        lfw3=b;
    end
elseif strcmp(lfwtype, 'saw')
    stati=findobj(gcf, 'Tag', 'StaticText11');
    set(stati, 'String', 'Sawtooth');
    if strcmp(rsst, '1')
        g=dc+am*exp(-(t*damp)).*sawtooth(2*pi*r*t,p);
        a=int32(g);
    elseif strcmp(rsst, '0')
        a=dc+am*exp(-(t*damp)).*sawtooth(2*pi*r*t,p);
    end
    if strcmp(wall, '1')
        b=a+0.1*sin(2*pi*50*t);
    elseif strcmp(wall, '0')
        b=a;
    end
    if strcmp(rectif, '1')
        lfw3=abs(b);
    elseif strcmp(rectif, '0')
        lfw3=b;
    end
elseif strcmp(lfwtype, 'ran')
    stati=findobj(gcf, 'Tag', 'StaticText11');
    set(stati, 'String', 'Pseudo Random');
    if strcmp(rsst, '1')
        g=dc+am*exp(-
(t*damp)).*sin(2*pi*rand(1)*10*r*t+p)+sin(4*pi*rand(1)*10*r*t+p)+sin(8*pi*rand(1)*10*r*t+
p)+sin(16*pi*rand(1)*10*r*t+p)+sin(32*pi*rand(1)*10*r*t+p)+sin(64*pi*rand(1)*10*r*t+p);
        a=int32(g);
    elseif strcmp(rsst, '0')
        a=dc+am*exp(-
(t*damp)).*sin(2*pi*rand(1)*10*r*t+p)+sin(4*pi*rand(1)*10*r*t+p)+sin(8*pi*rand(1)*10*r*t+
p)+sin(16*pi*rand(1)*10*r*t+p)+sin(32*pi*rand(1)*10*r*t+p)+sin(64*pi*rand(1)*10*r*t+p);
    end
    if strcmp(wall, '1')
        b=a+0.1*sin(2*pi*50*t);
    elseif strcmp(wall, '0')
        b=a;
    end
    if strcmp(rectif, '1')
        lfw3=abs(b);
    elseif strcmp(rectif, '0')
        lfw3=b;
    end
end
end
for i=1:100,
    waitbar(i/100);

```

```

end
    close(h);
    figure(h1);
subplot(514);
plot(t,lfw3);
title('LFW 3 Modulator');
ylabel('Volts');
elseif popup1==2
    lfomaker;
end

```

Fino a questo punto, il codice è identico a quello delle altre applicazioni, se non per il fatto che, qui vengono acquisite tre diverse LFW anziché una sola. Questo case, permette l'applicazione dell'algoritmo che svolge l'LFOCoder: si applicano i filtri di Butterworth al segnale, in modo da averne porzioni separate per la fase di processing, con la frequenza di centro banda, stabilita dall'utente, si applica la modulazione di controllo con le tre LFW previste, si ricompone il segnale mixando le varie porzioni modulate e si salva nella variabile di uscita il plot corrispondente al segnale Wet.

```

case 'apply'
    t=(1:data.count)/data.samplerate);
l=length(t);
n=1:l;
if data.channel==1
    in1=zeros(size(t));
    in2=zeros(size(t));
    in3=zeros(size(t));
    inl=zeros(size(t));
    inr=zeros(size(t));
if data.quantize==8
    in=(x-128)/128;
elseif data.quantize==16
    in=x/32768;
end
elseif data.channel==2
    in1=zeros(size(t));
    in2=zeros(size(t));
    in3=zeros(size(t));
if data.quantize==8
    in=(x-128)/128;
elseif data.quantize==16
    in=x/32768;
end
end

```

```

end
if strcmp(rsst,'1')
    partial=double(lfw1);
    normlffc=double(partial/am1+dc1);
    normlffw1=reshape(normlffc,1,1);
elseif strcmp(rsst,'0')
    normlffc=lfw1/am1+dc1;
    normlffw1=reshape(normlffc,1,1);
end
if strcmp(rsst,'1')
    partial=double(lfw2);
    normlffc=double(partial/am2+dc2);
    normlffw2=reshape(normlffc,1,1);
elseif strcmp(rsst,'0')
    normlffc=lfw2/am2+dc2;
    normlffw2=reshape(normlffc,1,1);
end
if strcmp(rsst,'1')
    partial=double(lfw3);
    normlffc=double(partial/am3+dc3);
    normlffw3=reshape(normlffc,1,1);
elseif strcmp(rsst,'0')
    normlffc=lfw3/am3+dc3;
    normlffw3=reshape(normlffc,1,1);
end
if data.channel==1
    h=waitbar(0,'Now LFOCoder is being applied (this could take a few minutes). Please
Wait...');
    back(n)=in(n);
    in1(n)=filter(b1,a1,in).*normlffw1(n);
    in2(n)=filter(b2,a2,in).*normlffw2(n);
    in3(n)=filter(b3,a3,in).*normlffw3(n);
    in(n)=in1(n)+in2(n)+in3(n);
elseif data.channel==2
    h=waitbar(0,'Now LFOCoder is being applied (this could take a few minutes). Please
Wait...');
    back1=in(:,1);
    back2=in(:,2);
    in1(n)=filter(b1,a1,back1).*normlffw1(n);
    in2(n)=filter(b2,a2,back1).*normlffw2(n);
    in3(n)=filter(b3,a3,back1).*normlffw3(n);
    in1(n)=in1(n)+in2(n)+in3(n);
    in1(n)=filter(b1,a1,back2).*normlffw1(n);
    in2(n)=filter(b2,a2,back2).*normlffw2(n);
    in3(n)=filter(b3,a3,back2).*normlffw3(n);
    inr(n)=in1(n)+in2(n)+in3(n);
end

```

```

close(h);
slide=findobj(gcf,'Tag','Slider1');
slval=get(slide,'Value');
if data.channel==1
    h=waitbar(0,'Recomposing and Mixing. Please Wait...');
    outc=in1+in2+in3;
    outd=reshape(outc,1,1);
    backd=reshape(back,1,1);
    out=slval.*outd+(1-slval).*backd;
    for i=1:100,
        waitbar(i/100);
    end
close(h);
figure(h1);
subplot(515);
plot(t,out);
title('Wet Signal');
ylabel('Normalized Values');
xlabel('Seconds');
legend('Monaural Channel');
elseif data.channel==2
    h=waitbar(0,'Recomposing and Mixing. Please Wait...');
    outd=reshape([in1 inr],1,2);
    backd=reshape([back1 back2],1,2);
    out=slval.*outd+(1-slval).*backd;
    for i=1:100,
        waitbar(i/100);
    end
close(h);
figure(h1);
subplot(515);
plot(t,out);
title('Wet Signal');
ylabel('Normalized Values');
xlabel('Seconds');
legend('Left Channel','Right Channel');
end
case 'filterresp'
    if1h=findobj(gcf,'Tag','EditText1');
    if1=str2num(get(if1h,'String'));
    if2h=findobj(gcf,'Tag','EditText2');
    if2=str2num(get(if2h,'String'));
    if3h=findobj(gcf,'Tag','EditText3');
    if3=str2num(get(if3h,'String'));
    h5=figure('Name','Filter 1 Response','NumberTitle','off');
    [b1,a1]=butter(4,[(if1/2)/((data.samplerate)/2) ((if1*2)/((data.samplerate)/2))]);
    [b2,a2]=butter(4,[(if2/2)/((data.samplerate)/2) ((if2*2)/((data.samplerate)/2))]);

```

```

[b3,a3]=butter(4,[(if3/2)/((data.samplerate)/2)) ((if3*2)/((data.samplerate)/2)]);
freqz(b1,a1,data.samplerate/2);
h6=figure('Name','Filter 2 Response','NumberTitle','off');
freqz(b2,a2,data.samplerate/2);
h7=figure('Name','Filter 3 Response','NumberTitle','off');
freqz(b3,a3,data.samplerate/2);
case 'saveasf'
    button3=questdlg('Would you like to save this signal for further processing?','Save
file...','Microsoft PCM WAVE FORMAT','Motion Picture Expert Group (MPEG-1) Layer
3','No','No');
    if strcmp(button3,'Microsoft PCM WAVE FORMAT')
        [newfile,newpath] = uiputfile('*.wav','Save as...');
        newpath1=[newpath];
        newfile1=[newfile];
        h2=waitbar(0,'Saving in Progress... Please Wait');
        wavwrite(out,data.samplerate,data.quantize,strcat(newpath1,newfile1));
        for i=1:100,
            waitbar(i/100);
        end
        close(h2);
    elseif strcmp(button3,'Motion Picture Expert Group (MPEG-1) Layer 3')
        [newfile,newpath] = uiputfile('*.mp3','Save as...');
        newpath1=[newpath];
        newfile1=[newfile];
        wavwrite(r,data.samplerate,data.quantize,'tmpfile.wav');
        string=strcat(newpath1,newfile1,'.mp3');
        if data.channel==1
            h2=waitbar(0,'Encoding and Saving in Progress... Please Wait');
            cmd=['lame --quiet -b 256 --resample 44.1 tmpfile.wav',' ',string];
            dos(cmd);
            for i=1:100,
                waitbar(i/100);
            end
            close(h2);
            delete('tmpfile.wav');
        elseif data.channel==2
            h2=waitbar(0,'Encoding and Saving in Progress... Please Wait');
            cmd=['lame --quiet -b 256 --resample 44.1 -m s tmpfile.wav',' ',string];
            dos(cmd);
            for i=1:100,
                waitbar(i/100);
            end
            close(h2);
            delete('tmpfile.wav');
        end
    end
end
case 'info'

```

```
h=helpdlg('This is the LFOCoder FX inside of Nova - Mod Tools enviroment built by  
Zambelli Cristian. For using this feature you need to have pre-acquired the signal with  
the Signal Acquisition System','Information');  
case 'close'  
    close(gcf);  
end
```

Capitolo 5

Algoritmi di Modulazione e SmartPlayer Technique

5.1 Prefazione del sistema

Il capitolo di questa trattazione riguarda l'argomento su cui è basato il concepimento di questa tesi: gli algoritmi di modulazione. È proprio questa classe particolare di effetti per il processing dei segnali audio, che oggi rappresenta la maggior parte del mercato mondiale degli effetti utilizzati dai musicisti professionisti e non. Un algoritmo di modulazione è un oggetto che sostanzialmente, prende in ingresso un segnale audio, applica un ritardo ad esso che può essere variabile o no, effettuando una sorta di modulazione del neuma, per creare delle sonorità più "ampie", che a volte sono in grado di ricreare un ambiente acustico¹. Gli algoritmi sviluppati in questo capitolo, non sono da confondere con gli algoritmi che riguardano gli effetti di modulazione diretta. C'è infatti una differenza fondamentale tra le due categorie: gli algoritmi di modulazione effettuano una suddivisione del segnale di ingresso, in più porzioni², dove ognuna di esse è seguita da un ritardo variabile da una forma d'onda che permette di modulare il segnale; gli algoritmi di modulazione diretta del segnale invece, effettuano una modulazione dell'involuppo del segnale, in base ad una forma d'onda definita precedentemente. Non dobbiamo dimenticare però, che siamo all'interno di un environment che è interamente progettato in Matlab, quindi dovremo garantire la compatibilità degli algoritmi che svilupperemo, con il resto delle applicazioni del sistema Nova – Mod Tools. Per fare questo, sarà necessario, ancora una volta, sviluppare una GUI Application, che ricavi il segnale di ingresso dal file di interscambio *xchangedata.mat* e che possa manipolare i dati presenti in esso, in modo da poter venire incontro alle nostre esigenze. Da queste poche righe, possiamo già capire allora, quali sono gli strumenti che ci serviranno per l'implementazione di questa parte del sistema:

- Il segnale audio di ingresso dovrà essere contenuto, ovviamente insieme ai dati che riguardano la profondità di quantizzazione in bit, il samplerate, il formato del file ed altre informazioni utili, nel file di interscambio *xchangedata.mat*. Questo consentirà la compatibilità del sistema sviluppato, con tutti i tools dell'environment, compreso il sistema di Signal Acquisition, la cui importanza per l'acquisizione del segnale via file o real – time, è fondamentale.

¹ È chiaro che per ricreazione dell'ambiente acustico, si intende una sorta di riverberazione "falsa" introdotta dalla modulazione del suono, ma che non è correlata in alcun modo con la riverberazione.

² Definite anche "voci", nel linguaggio musicale.

- Lo strumento che ci servirà per effettuare una manipolazione sul ritardo da introdurre sulle varie voci del segnale audio acquisito, dovrà provenire da un LFO, proprio per il fatto che, negli algoritmi di modulazione, i tempi di ritardo, richiedono una variazione dell'ordine del millisecondo. È necessaria quindi, la compatibilità verso il tool LFO – Maker, fondamentale per la creazione di LFW proprietarie e non standardizzate come nella maggior parte dei software per il sound processing o degli effetti real – world³. Bisogna chiarire però, che non è possibile utilizzare tutti i tipi di LFO per l'applicazione degli effetti di modulazione, infatti esistono alcune classi di LFO, come ad esempio quelli esponenziali – sinusoidali, che generano forme d'onda al di fuori di intervalli utilizzabili con i sistemi di modulazione.
- I ritardi sulle voci del segnale di ingresso, dovranno essere implementati seguendo una metodologia che si avvicini il più possibile a quella utilizzata nella progettazione degli effetti di modulazione real – world, garantendo una minimizzazione del rumore e una qualità superiore per il trattamento del segnale.

Gli effetti che andremo a realizzare saranno quindi: il Real Polyphonic Chorus, che implementa un multi – voice Chorus (fino a 12 voci), in cui è possibile variare ogni tipo di parametro per ogni voce relativa al segnale di ingresso, con l'aggiunta di un sistema di ricreazione dell'ambiente stereofonico a 12 voci; il Classical Flanger – Phaser, che implementa il più classico degli effetti Flanger degli anni '70, con l'aggiunta di un Phasing Stage che permette di ottenere un Flanger Stereofonico, in cui è possibile scegliere il grado di sfasamento delle voci, oltre ad ogni parametro che riguarda la manipolazione del segnale tramite l'effetto.

5.2 Chorus

5.2.1 Come funziona un Effetto Chorus

Proprio come un coro di cantanti, l'effetto chorus permette di ottenere delle sonorità che simulano il suono di un segnale audio, come se fosse eseguito da più persone. Il suo effetto è quello di aggiungere spazialità e dimensione⁴ al suono, ed è spesso descritto da un suono ricco e arioso. L'algoritmo che sta alla base del Chorus non è affatto complicato o perlomeno così spettacolare da

³ Per real – world si intende la classe di effetti stompboxes, rack – mounted e floor – board multifx.

⁴ Da non confondere con la spazialità introdotta dagli effetti di ritardo e riverberazione.

renderlo inaccessibile. Cosa succede a due persone che tentano di suonare uno strumento musicale all'unisono? Per tanto impegno e bravura riproposti dai due musicisti, accadrà sempre che ci sarà una imprecisione nella sincronizzazione del suono prodotto, quindi in sostanza c'è un ritardo tra i due segnali audio. Inoltre, il pitch⁵ dei due strumenti, nonostante una perfetta accordatura⁶, potrebbe portare ad una “deviazione” di tono. Queste sono le funzionalità che l'effetto Chorus riproduce. Il leggero ritardo tra i due segnali audio, può essere facilmente implementato con una “linea di ritardo” o “delay line”. Per creare l'effetto di “scordatura”, si può utilizzare una trasformazione della linea di ritardo, in una linea di ritardo a lunghezza variabile o “Variable Length Delay Line”. La dicitura Variable Length, mi indica che il ritardo cambia in funzione del tempo, anche se ancora non è stato chiarito quale sarà l'effetto sul pitch del segnale audio. Per capire come avviene questo fenomeno, si immagini di visualizzare la linea di ritardo come un dispositivo di registrazione. La sua funzione è quella di memorizzare una copia esatta del segnale di ingresso in arrivo, come un normale Cassette Recorder, e di mandarla in uscita con un leggero ritardo, alla stessa frequenza. Se si desidera incrementare la quantità di ritardo, si vorrà avere a disposizione un segmento che dovrà contenere il segnale prima del playback, di dimensioni maggiori. Per fare questo, si effettua una lettura, al di fuori della linea di ritardo, ad un rate minore di ciò che viene effettivamente registrato (il rate di registrazione non cambia, quindi più segnale viene immagazzinato). Questa procedura è del tutto simile al compimento della pressione delle dita, sul nastro trasportatore della cassetta, che si traduce in un abbassamento del pitch. Allo stesso modo, per ridurre il tempo di ritardo, dovremmo leggere più velocemente, che è analogo a velocizzare il playback di una cassetta, con il risultato di aumentare il pitch⁷. Quindi, se noi sommiamo il segnale ritardato e modificato nel pitch ottenuto con questo procedimento, con il segnale originale, abbiamo l'effetto Chorus e lo possiamo vedere nella figura successiva:

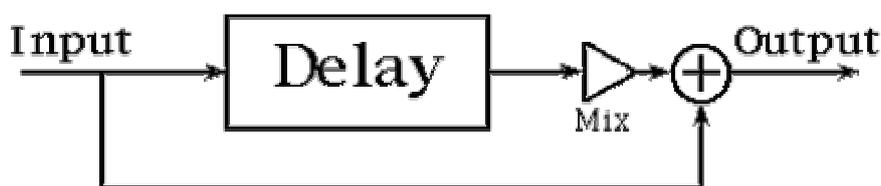


Figura 33: Schema base di un effetto Chorus

Questa struttura è molto familiare, perché usata anche per un altro tipo di effetto che vedremo successivamente: il Flanger. Il Chorus differisce da esso per alcuni parametri, ed una differenza

⁵ Nei testi latini prende il nome di neuma o altezza della nota.

⁶ Procedimento adottato dai musicisti, per sincronizzare una sorta di livello di riferimento con altri strumenti musicali per evitare sgradevoli effetti di “battimento d’onda”, qualora si propagano nell’ambiente, due suoni a frequenza leggermente diversa.

⁷ Questo effetto prende anche il nome di effetto Munchkin’, che deriva da un famoso cartone animato in cui i protagonisti cantavano le canzoni con un pitch volutamente elevato.

chiave, è la quantità di delay usato. I tempi di ritardo in un Chorus sono tipicamente maggiori rispetto ad un Flanger e variano tra i 20 e i 30 ms (il Flanger utilizza ritardi da 1 a 10 ms). Questo maggior ritardo non produce la tipica caratteristica di sweeping introdotta dal suono del Flanger. Un'altra differenza, è che molto spesso il Chorus non ammette un percorso di retroazione o feedback, del segnale già processato. L'unico punto che resta da discutere, è il modo con cui il tempo di ritardo deve cambiare. In generale, si utilizzano delle forme d'onda periodiche che noi conosciamo molto bene: le LFW. È quindi fondamentale l'apporto dei LFO per questi sistemi. Si può quindi controllare il suono del Chorus, cambiando la forma d'onda, la sua frequenza, l'ampiezza, ecc.

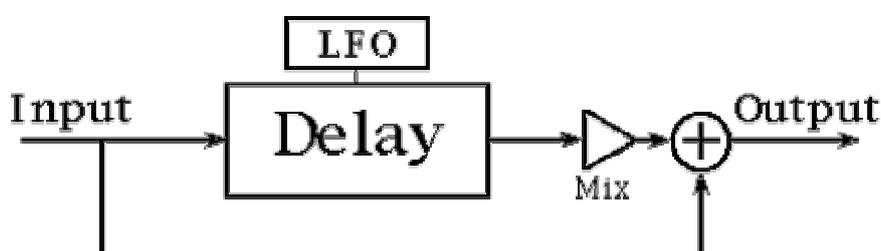


Figura 34: Effetto Chorus con l'aggiunta di un LFO per la modifica del tempo di ritardo

Algoritmicamente possiamo descrivere l'effetto Chorus con questa equazione:

$$out(t) = in(t) + in(t - d * f(t)) \quad (1)$$

dove d è la lunghezza del ritardo, ad una frequenza di campionamento non modificata, ed $f(t)$ rappresenta la funzione che modificherà la frequenza di campionamento. La deviazione di pitch, in ogni istante temporale è proporzionale a:

$$\log \left[1 - d * \frac{df(t)}{dt} \right] \quad (2)$$

L'implementazione di questo tipo di effetto, nel dominio analogico, si basa sull'utilizzo di una circuiteria particolare, che permette di ovviare alle problematiche di dover generare dei tempi ritardo molto contenuti. Questi circuiti prendono il nome di BBD⁸, molto in voga negli anni '70 ed ora reperibili a prezzi elevati. I dispositivi di questa classe si comportano fondamentalmente come

⁸ Acronimo di Bucket Brigade Delay. Si tratta di circuiti integrati a più stadi che permettevano di ottenere ritardi dell'ordine di qualche centinaio di ms. Oggi si trovano sui cataloghi di elettronica a prezzi elevati e la qualità non è più la stessa di allora. Il miglior BBD è stato il Panasonic MN3002, che si trova nella maggior parte delle unità di Boss ed EHX degli anni '70, oggi sostituito con il TDA2107.

un Sample and Hold a più stadi in cascata. Nel dominio digitale invece, i ritardi sono piuttosto semplici da implementare, se si utilizzano dei buffer circolari. Molto semplicemente, si prende una porzione di memoria e si legge e scrive su di essa sequenzialmente per ogni sample period, incrementando alla prossima locazione di memoria per ogni campione. Tuttavia una linea di ritardo variabile richiede un po' più di lavoro. Il ritardo variabile, richiede l'utilizzo di tempi di ritardo che non sono multipli interi del periodo di campionamento (mentre il segnale di ingresso è campionato a multipli del periodo di campionamento). Bisogna quindi effettuare una stima del valore di due campioni vicini: questo processo prende il nome di interpolazione. Un metodo molto semplice è dato dall'interpolazione lineare, la quale connette i valori dei due campioni tramite una segmento di retta, ma è un metodo che può introdurre rumore, quindi si preferisce usare altre metodologie come l'interpolazione passa – tutto o l'interpolazione tramite Spline cubica. Una miglioria del sistema di Chorus, a cui si è potuto assistere con il trascorrere del tempo, è stata l'introduzione del Chorus Stereofonico, generalmente costituito da due Chorus Monofonici in quadratura⁹. Questa tecnica crea un suono più ampio, perché il suono che arriva ad ognuna delle orecchie è differente. In un Chorus Multi – Voiced, per ricreare la stereofonia, è sufficiente effettuare il panning delle voci al di fuori del centro della zona di missaggio (lo vedremo nel paragrafo successivo).

5.2.2 Parametri di un Chorus

Analizziamo ora i principali parametri che vengono utilizzati nella maggior parte delle unità effetto presenti sul mercato e che ci serviranno per l'implementazione successiva del sistema di Chorus.

- **Delay:** questo parametro controlla semplicemente la quantità di ritardo da utilizzare. Più specificatamente, esso controlla il minimo tempo di ritardo utilizzato. Più il delay risulta piccolo, più il Chorus tenderà ad agire come un Flanger. Tempi tipici di ritardo variano tra i 20 e i 30 ms.
- **Sweep Depth/Width:** questo parametro controlla quanto cambiamento sul ritardo è ammissibile durante l'applicazione dell'effetto. È abitualmente espresso in millisecondi, e la somma dello Sweep Depth con il parametro di Delay, mi da il ritardo massimo utilizzato, per il processing del segnale. In alternativa, si può pensare allo sweep come l'ampiezza del LFO pilota.

⁹ Cioè con l'uscita sfasata di 90° l'una rispetto all'altra.

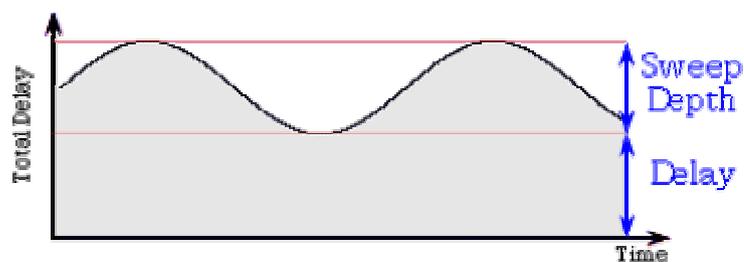


Figura 35: Il delay complessivo è la somma dei due precedenti parametri

Lo Sweep Depth incrementa anche la modulazione di pitch introdotta dalla linea di ritardo variabile. Sweep elevati causeranno degli “warble¹⁰” nel suono finale.

- LFO Waveform: la forma d’onda in uscita dal LFO mostra come il ritardo cambia nel tempo. Quando la forma d’onda raggiunge il massimo, il ritardo sarà al suo picco massimo. Quando la forma d’onda e il total delay time aumentano, il pitch si abbassa. Ecco alcune LFO usate tipicamente:

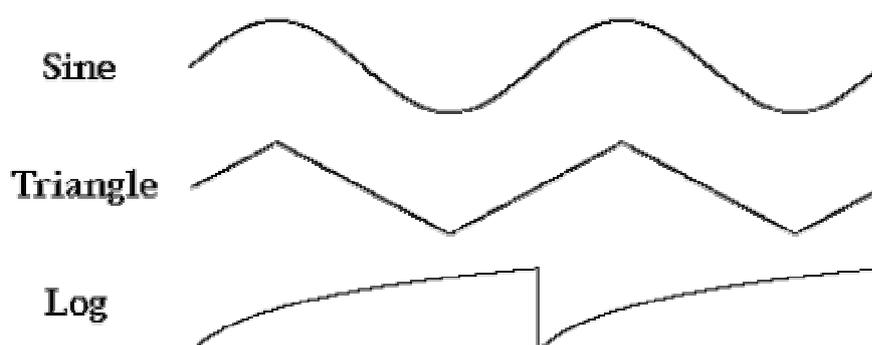


Figura 36: LFW utilizzate tipicamente negli effetti Chorus

La quantità di pitch modulation introdotta dal Chorus è legata alla velocità di variazione del LFO, le porzioni ripide della forma d’onda produrranno un largo cambiamento di pitch, mentre le porzioni piatte non hanno effetto di pitch modulation. Aumentare lo Sweep Depth si traduce in uno stretching della LFW verticalmente, che permette alterazioni maggiori di pitch nel segnale di uscita.

- Speed/Rate: questo controllo è piuttosto semplice da capire. Questo parametro si riferisce alla velocità di ripetizione di se stessa ed è anche il fattore che stabilisce la pitch modulation. Aumentare il rate della LFW consiste nel comprimere la forma d’onda nel tempo, che corrisponde in una maggior pitch modulation.

¹⁰ Segnale audio percepito come una vibrazione continua del suono.

- **Number of Voices:** fino a questo punto si è trattato di Chorus che si riferiscono ad una singola voce e questo significa che c'è una sola copia del segnale di ingresso, su cui lavorare. Ma non c'è ragione che ci impedisce di avere copie multiple del suono, modellizzando una situazione con più di due strumenti suonati all'unisono. Alcuni Chorus permettono di scegliere il numero di voci. Tipicamente un Multi – Voiced Chorus usa una singola LFW per tutte le voci, ma ogni voce ha una fase differente. Questo significa che in ogni punto dell'asse temporale, ogni voce si trova ad essere in un punto diverso e quindi ad un diverso ritardo. Se tutte le voci fossero in fase, si avrebbe un effetto di singola voce, ma a livello incrementato.

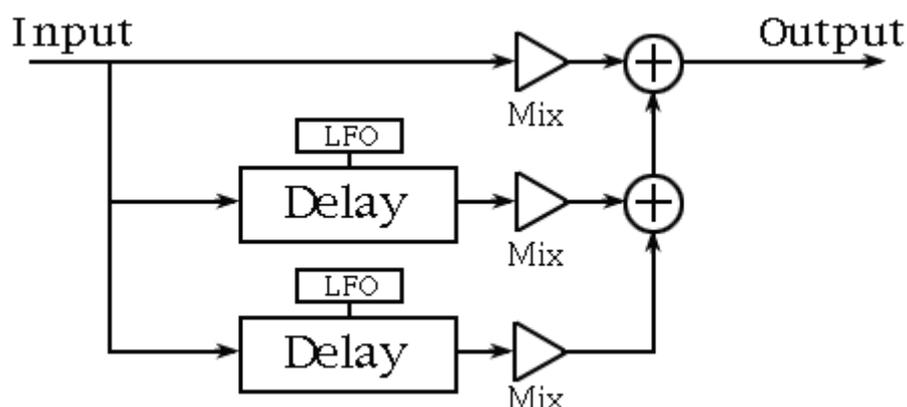


Figura 37: Struttura di un Multi - Voiced Chorus

5.2.3 Speed Test

Effettuiamo ora uno Speed Test, per verificare se il tempo di esecuzione del nostro algoritmo è sufficientemente contenuto e competitivo rispetto ad altri software. Inoltre, con questo test, saremo anche in grado di valutare le capacità computazionali di Matlab su grandi volumi di dati:

	Real Polyphonic Chorus implementato nel Nova – Mod Environment Tools	Real – Time Chorus implementato nel software Cool Edit Pro 2.0 di Syntrillium
File da 15 sec. Monaurale processato con un Chorus a singola voce con rate 0.6 Hz	11.72 sec compreso il plot time con le unità di misura scalate	4.16 sec senza il plot time
File da 15 sec. Monaurale processato con un Chorus a sei voci con rate 0.6 Hz	24.35 compreso il plot time con le unità di misura scalate	12.37 sec compreso il plot time
File da 15 sec. Monaurale processato con un Chorus a dodici voci con rate 0.6 Hz	46.47 compreso il plot time con le unità di misura scalate	24.41 sec compreso il plot time
File da 15 sec. Monaurale processato con un Chorus a dodici voci con rate 0.6 Hz e costruzione del campo stereofonico	39.87 sec compreso il plot time con le unità di misura scalate e il campo stereofonico completamente costruito	24.66 sec + il tempo per creare un segnale stereofonico, dal momento che non esiste la funzione di costruzione

Come si evince da questa tabella, nonostante l'utilizzo della riallocazione degli spazi lineari e del built – in indexing¹¹ di Matlab, non si riesce ad eliminare il gap temporale (di almeno il doppio) che esiste tra un linguaggio interpretato come Matlab ed uno compilato come il C/C++. Nonostante tutto però, il nostro sistema offre una funzionalità di costruzione del campo stereofonico a minor dispendio temporale rispetto al software Cool Edit Pro.

5.3 Flanger

5.3.1 Come funziona un Effetto Flanger

L'effetto Flanger ha un suono caratteristico che viene associato ad una sensazione di “avvolgimento” dell'ascoltatore da parte del suono stesso o associato al rumore di un jet che decolla sopra la testa. Il Flanger è generalmente considerato un tipo particolare di Phasing (un altro popolare effetto di modulazione che non usa però LFO). Come noteremo, il Flanger crea un set di notches equi – spaziate nello spettro audio. Il Phasing invece, usa sempre un set di notches che però, vengono arbitrariamente modificati, mediante un filtraggio di tipo passa – tutto. Il Flanger è creato missando un segnale, con una sua copia leggermente ritardata, dove il ritardo varia in maniera continua. Questo procedimento, già analizzato con l'effetto Chorus, non è difficile da ottenere con un equipaggiamento audio amatoriale, e ci sono leggende piuttosto contrastanti su come sia nato questo effetto. Alcuni affermano che fu scoperto dai Beatles, altri affermano che sia stato Phil Spector... Resta comunque il fatto, che il Flanger è stato scoperto per errore. Quando i ritardi nei segnali audio, venivano ancora creati facendo “girare” il segnale su due differenti bobine di Tape Recorder a velocità diversa, qualcuno toccò la testina del registratore, producendo una alterazione del pitch del suono. Con il mix finale dei segnali, il suono caratteristico del Flanger vide la luce. Siccome la testina sui cui poggia il nastro, è anche conosciuta con il nome di flangia, ecco nato il nome di una nuova leggenda. I Flanger dei giorni d'oggi, permettono di modellare il suono, controllando quanto segnale ritardato deve essere aggiunto all'originale tramite un controllo definito “depth¹²”.

¹¹ Si tratta di una sorta di in – place computation che Matlab permette di utilizzare con le variabili vettoriali o matriciali.

¹² In alcuni Flanger questo prende anche il nome del controllo di Mix.

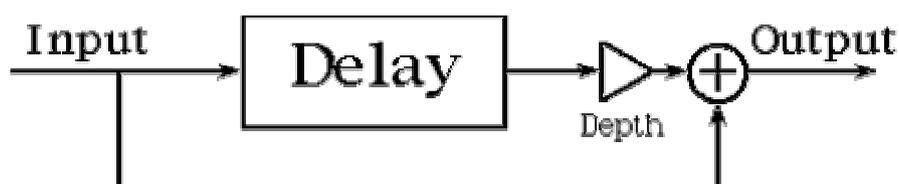


Figura 38: Esempio di struttura di Flanger con controllo di Depth

Quando ascoltiamo un segnale audio processato con un Flanger, non siamo in grado di percepire l'eco, a causa del ritardo molto corto. In un Flanger, i tempi di ritardo tipici variano da 1 a 10 ms¹³. Invece di creare un eco, il ritardo ha l'azione di "filtrare" il segnale, e questo effetto causa una serie di notches nella risposta in frequenza, come possiamo vedere nella figura successiva. I punti in cui la risposta in frequenza tende a zero, indicano che il suono a quelle frequenze, viene eliminato, mentre le altre frequenze subiscono una modulazione di ampiezza. Questo tipo di risposta in frequenza prende il nome di filtro comb¹⁴.

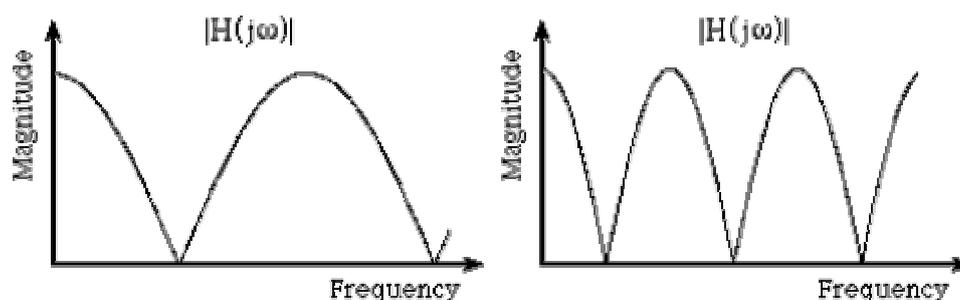


Figura 39: Risposta in frequenza di un Flanger con Depth=1. La figura di sinistra rappresenta un Flanger con un ritardo minore.

Questi notches nella risposta in frequenza sono creati per interferenza distruttiva. Si immagina un tono puro costituito da una sinusoide. Se si prova a ritardare questo segnale ed aggiungerlo all'originale, la somma dei due segnali può apparire leggermente differente. Ad un estremo, dove il ritardo è tale che il segnale sia perfettamente fuori fase, come un segnale aumenta, l'altro diminuisce della stessa quantità in modo da causare interferenza distruttiva, in modo che l'intero segnale sparisca completamente. Ovviamente, i due segnali possono rimanere in fase dopo il ritardo, raddoppiando l'ampiezza del segnale a quella particolare frequenza, causando interferenza costruttiva. Per ogni ritardo dato, alcune frequenze saranno eliminate e alcune no. Nel Flanger, si

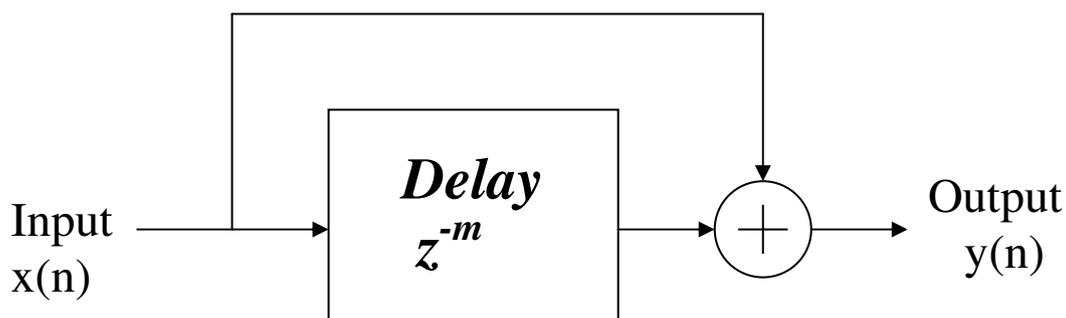
¹³ Si consideri che l'orecchio umano, è in grado di percepire dei ritardi a partire da 50 – 70 ms.

¹⁴ Comb è il termine inglese che indica il pettine. Infatti la risposta in frequenza di questo filtro, rassomiglia ai denti che appoggiano su un pettine. L'equazione che descrive un filtro comb è:

$$out(t) = in(t + d) + in(t + d - b)$$

dove d è il ritardo e b è la variazione continua dello stesso.

può controllare la profondità¹⁵ dei notches, proprio usando il controllo di depth. Quando il depth è a zero, la risposta in frequenza è piatta, ma come si comincia a incrementare il depth, i notches cominciano ad apparire e ad estendersi verso il raggiungimento dello zero nella risposta. Comunque anche se i notches non si estendono completamente a zero, è possibile percepirne l'effetto. Il fatto che il ritardo esegua un filtraggio di tipo comb, lo possiamo dimostrare così: si prenda la struttura del flanger



L'equazione del sistema con le trasformate z è:

$$Y(z) = X(z) + z^{-m} X(z) \quad (3)$$

$$H(z) = \frac{Y(z)}{X(z)} = 1 + z^{-m} \quad (4)$$

Sostituendo z con $e^{j\hat{\omega}}$ si ottiene la risposta in frequenza del sistema, che è proprio quella di un filtro comb, in regime sinusoidale:

$$H(e^{j\hat{\omega}}) = 1 + e^{-j\hat{\omega}m} = e^{-j\hat{\omega}m/2} [2 \cos(\hat{\omega}m/2)] \quad (5)$$

$$|H(e^{j\hat{\omega}})| = |2 \cos(\hat{\omega}m/2)| \quad (6)$$

L'azione di sweep dei notches è ottenuta cambiando continuamente la quantità di ritardo usato. Come il ritardo aumenta, i notches shiftano verso le basse frequenze. Il modo in cui questo ritardo è variato, è determinato dalle LFW applicate tramite LFO. Il cambiamento di ritardo nel Flanger, crea una sorta di pitch modulation, percepita come dei "warbles". Questo accade perché si deve leggere più velocemente o lentamente dal segnale ritardato. Se si pensa al Flanger come a due Tape Recorder che processano lo stesso segnale audio. Per incrementare il ritardo tra i due segnali, è

¹⁵ O meglio la presenza di essi nell'intero spettro relativo al segnale audio.

necessario rallentare la flangia, mentre per decrementarlo bisogna velocizzare la flangia, causando un sorta di effetto 'munchkin'. Ovviamente solo la copia ritardata del suono ha dei cambiamenti di pitch, quando mixato con il segnale inalterato. La circuiteria di Chorus e Flanger sono simili, tranne alcune differenze che fanno chiamare il Chorus come il figlio del Flanger. Quando si usa il Flanger con strumenti musicali, si possono esplorare delle possibilità interessanti. Per strumenti non percussivi che generano un pitch e toni sovrarmonici, i notches che il Flanger produce in teoria dovrebbero coincidere esattamente con la fondamentale e con gli overtones¹⁶, eliminando il suono dello strumento in parte. In pratica, uno strumento non scompare interamente, ma subisce una modulazione di ampiezza severa. Per questo motivo, si usa spesso il Flanger con strumenti percussivi o noise – like. Infatti, il Flanger è spesso usato nel mix complessivo, piuttosto che uno specifico strumento. Così come per il Chorus, anche per il Flanger è possibile creare un sistema di Stereofonia, che permette di ampliare il suono, utilizzando due Flanger monofonici in quadratura tra loro, in modo che l'orecchio umano percepisca differenti effetti di Flanger.

5.3.2 Parametri di un Flanger

Vediamo in questo paragrafo, quali sono i parametri più importanti per la maggior parte delle unità Flanger presenti sul mercato:

- **Depth (Mix):** questo parametro permette di avere notches più pronunciati. Nelle unità multieffetto, il parametro di depth può essere controllato solo nella fase di mixing finale, e non è disponibile nella processazione principale. I termini Depth e Mix sono intercambiabili.
- **Delay:** questo parametro specifica il minimo ritardo usato per la copia del segnale di ingresso. Guardando la risposta in frequenza, questo valore determina l'altezza del primo notch. Se il Delay aumenta, il primo notch tende a diminuire. In alcuni casi, questo parametro viene settato a zero, in questo caso i notches effettueranno uno sweep nella parte superiore del range di frequenze audio ed essenzialmente scomparirà momentaneamente. Nella maggior parte dei casi, questo parametro non è controllabile.
- **Sweep Depth (Mix):** questo parametro stabilisce l'ampiezza della variazione del tempo di ritardo, ed essenzialmente corrisponde all'ampiezza della LFW. Lo sweep depth rappresenta

¹⁶ Strumenti come il pianoforte generano tipicamente dei suoni che includono una fondamentale e multipli interi dell'armonica fondamentale. Al contrario, la batteria, include moltissimi armonici e non genera una fondamentale precisa.

il ritardo massimo sommabile al ritardo fissato dal Delay Parameter. Esso determina di quanto in basso, nella risposta in frequenza, devono andare i notches. Un valore basso di Depth, manterrà la variazione del ritardo piuttosto contenuta, mentre un valore più elevato, causerà notches più spalmati su tutto lo spettro di frequenza audio. La figura successiva mostra il legame dei parametri con la LFW

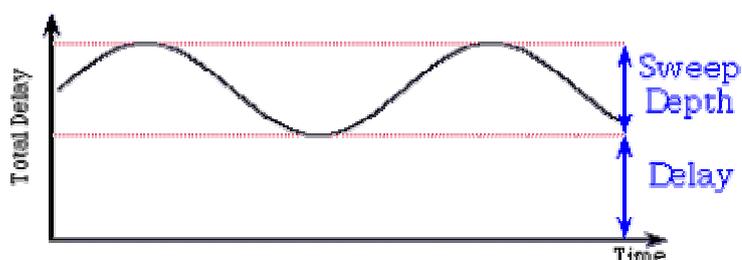


Figura 40: Struttura del tempo di ritardo per il Flanger

All'aumentare di questo parametro, la pitch modulation comincia a farsi più percepibile. Da notare che al variare del parametro di ritardo, sia i limiti superiori che inferiori del primo notch saranno cambiati, ma se si modifica il valore di Depth, la modifica avviene solo al valore del limite inferiore. Quando si setta il Flanger per effettuare lo sweep¹⁷ su un particolare range di frequenze, prima è meglio settare il delay time, in modo da stabilire il punto massimo dello sweep, poi per aggiustare il punto minimo, si agisce sullo sweep depth.

- LFO Waveform: alcuni Flanger permettono di scegliere la LFW che permette il cambiamento del tempo di ritardo nel tempo.

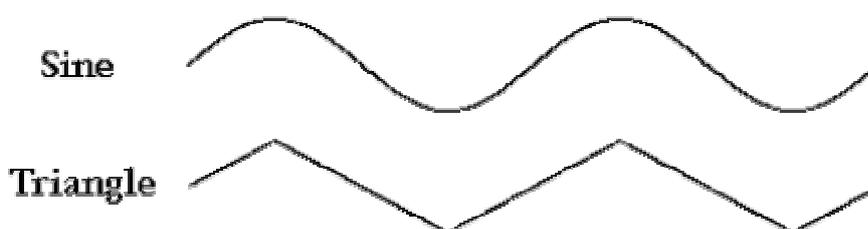


Figura 41: LFW più comuni nelle unità Flanger del mercato

- Feedback/Regeneration: alcune unità danno la possibilità di instradare l'uscita del Flanger verso l'ingresso. In alcuni casi, si può anche specificare se bisogna sommare o sottrarre il segnale di feedback. Un valore elevato di feedback può creare un suono estremamente

¹⁷ In questo caso ci si riferisce agli sweep dei notches e non allo Sweep Depth.

“metallico” e “intenso”. Ovviamente, quando il guadagno del feedback tende a uno¹⁸, il sistema tende ad essere instabile, risultando in un possibile clipping o overflow.

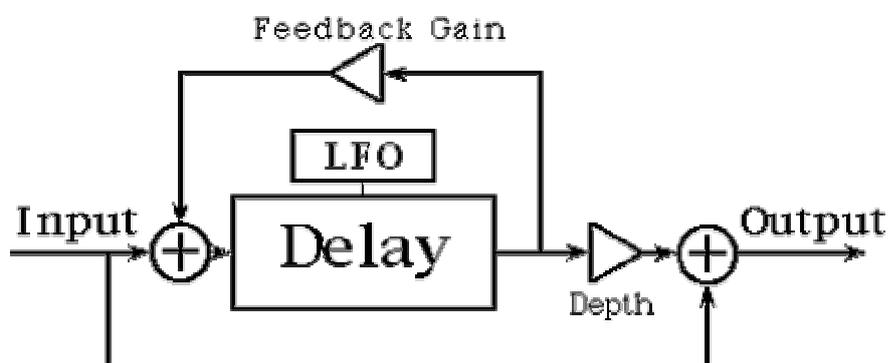


Figura 42: Struttura di un sistema Flanger con l'introduzione del controllo di Feedback Gain

- **Speed/Rate:** questo controllo stabilisce la velocità di ripetizione della LFW nel tempo o equivalentemente quante volte per secondo i notches devono andare in alto e in basso nella risposta in frequenza. Questo parametro aumenta anche la quantità di pitch modulation.

5.3.3 Speed Test

Così come abbiamo fatto per il sistema di Chorusing, vogliamo testare la velocità dell'algorithmo di Flanger – Phaser, per valutare la sua capacità di trattare i dati in modo veloce e corretto, nei confronti di grandi quantità di operazioni da compiere.

	Flanger - Phaser implementato nel Nova – Mod Environment Tools	Flanger implementato nel software Cool Edit Pro 2.0 di Syntrillium
File monaurale da 15 sec. processato con un Flanger a singola LFW	4.41 sec compreso il plot time con le unità di misura scalate	2.48 sec compreso il plot time
File monaurale da 15 sec. processato con un Flanger a doppia LFW	6.01 sec compreso il plot time con le unità di misura scalate	3.01 sec + il tempo per costruire un segnale stereofonico, perché Cool Edit non supporta lo Stereo – Flanger da un segnale monaurale
File monaurale da 15 sec. processato con un Flanger a doppia LFW + feedback	7.01 sec compreso il plot time con le unità di misura scalate	3.13 sec + il tempo per costruire un segnale stereofonico, perché Cool Edit non supporta lo Stereo – Flanger da un segnale monaurale

¹⁸ Dalla Teoria dei Sistemi, quando il guadagno di un sistema in retroazione tende all'unità, il sistema tende a spostarsi dal suo punto di equilibrio, per non tornarci più. Tale condizione viene detta di instabilità.

Anche in questo caso, si può notare che, pur essendo i tempi di calcolo piuttosto contenuti, non si può competere con un linguaggio compilato.

5.4 La Tecnica SmartPlayer e lo Human Brain Behaviour

5.4.1 Approccio e sviluppo dell'idea

Analizzando attentamente gli algoritmi di modulazione, abbiamo capito che questi oggetti, lavorano sul ritardo introdotto su una particolare porzione del segnale audio (la voce appunto), in modo da variare questa caratteristica nel tempo, in maniera periodica. Ecco quindi che gli LFO, rappresentano un valido aiuto per compiere questa operazione e nonostante la loro impraticità di utilizzo nei software di alta fascia per il sound processing come ad esempio Cool Edit Pro o Cubase¹⁹, trovano ancora oggi la maggior parte di applicazione negli effetti di modulazione real – world. È stato però anche detto, che questi algoritmi di modulazione che noi applichiamo ad un segnale audio arbitrario, ci dovrebbero restituire delle sonorità che ci diano l'idea di più persone in grado di riprodurre lo stesso segnale quasi all'unisono. La domanda è quindi: può un LFO rappresentare il comportamento di un essere umano? O meglio: può il funzionamento di un essere umano, stimato da una forma d'onda periodica standardizzata²⁰? La risposta alla domanda è molto semplice: NO! Se si valuta attentamente il comportamento di due esseri umani differenti, che siano ad esempio in grado di suonare lo stesso brano con il pianoforte, noteremo che esistono delle differenze sostanziali nell'esecuzione, dovute tipicamente: ad una conoscenza del brano, ad una sicurezza imputabile alla bravura nel suonare lo strumento, ecc. Differenze che non sono ben rappresentabili da una singola sinusoide, ma che sono stimabili, ovviamente con un elevato grado di imprecisione, dal momento che non siamo ancora a conoscenza del funzionamento preciso del cervello umano, da semplici equazioni che coinvolgono parametri diversi. L'idea quindi, è di costruire un sistema che permetta di essere integrato con gli algoritmi di modulazione svolti, che però dia la possibilità di raggiungere un livello maggiore di astrazione, dovuto all'utilizzo di un algoritmo in grado di poter rappresentare a grandi linee, il comportamento del cervello umano (da qui è nato il nome di Human Brain Behaviour) di un musicista: il sistema di SmartPlayer. Grazie a questa funzionalità, saremo in grado di trasformare un Chorus in un Chorus "intelligente", basato su una modulazione dei ritardi, che non tiene conto dell'andamento di una sinusoide, ma che tiene

¹⁹ Questi software, utilizzano per praticità e per velocizzare le operazioni di calcolo, dei ritardi che vengono modificati da uno sweep lineare di valori. Ciò significa che non è una forma d'onda che modula il ritardo, ma è un ciclo che varia il moltiplicatore del tempo di ritardo con un andamento lineare o a volte pseudo – randomico.

²⁰ Per standardizzata si intende che appartiene alle classiche onde sinusoidali, triangolari, ecc. Tutte forme d'onda riconducibili a forme d'onda matematiche predefinite.

conto ad esempio, della timidezza di chi sta suonando. Si vuole associare, ad ogni singola componente del comportamento di un musicista, una variabile. È chiaro che se dovessimo considerare realmente, il funzionamento del cervello umano, bisognerebbe valutare un sistema di equazioni differenziali fortemente accoppiate in N incognite, dove N rappresenta il numero di neuroni presenti all'interno del tessuto cerebrale, ma è chiaro che qui stiamo andando verso la biochimica, la medicina e chissà cos'altro... Rimanendo con i piedi per terra, vogliamo utilizzare un sistema piuttosto intuitivo, basato sui caratteri principali del comportamento di un musicista e che permetta un livello di calcolo molto semplice e affrontabile da qualsiasi macchina. Ovviamente, perché sia resa possibile l'integrazione con l'intero set di algoritmi di modulazione, si dovrà costruire un file in Matlab basato ancora una volta su una GUI Application che dovrà gestire i dati, ma questo lo vedremo nei paragrafi successivi.

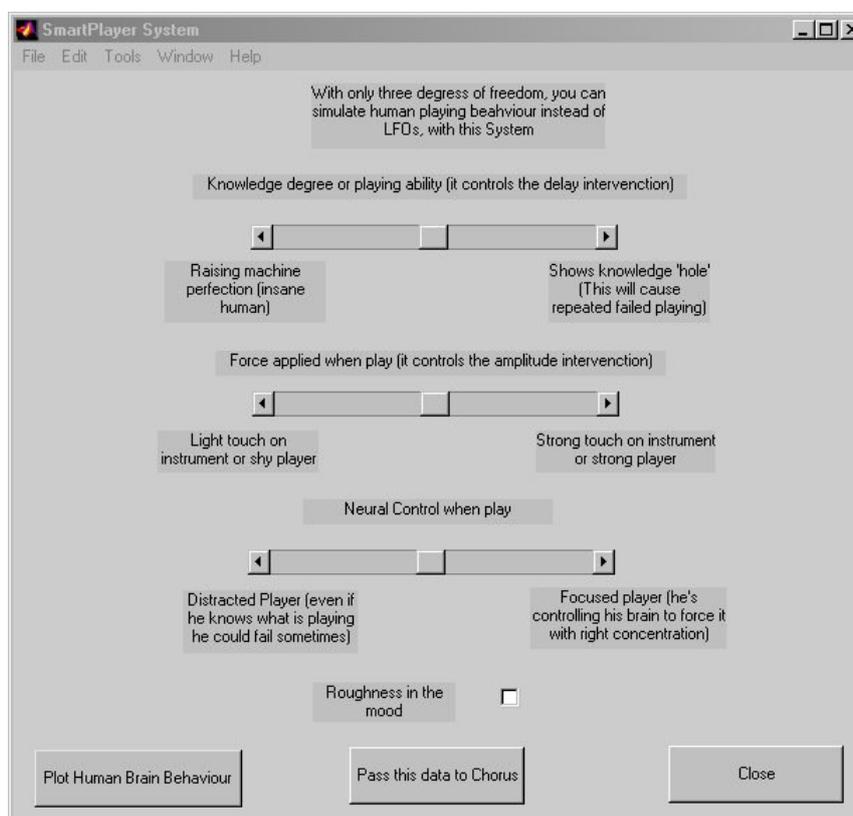


Figura 43: Struttura dell'applicazione GUI del sistema SmartPlayer

5.4.2 Parametri del sistema di SmartPlayer

Stimiamo ora quali sono i parametri che ci consentiranno di poter utilizzare una simulazione del comportamento del cervello di un musicista e riportiamo questi parametri sotto forma di variabile che poi sarà inserita in una equazione ben definita. Da alcuni discorsi con musicisti di diverso

stampo, è emerso che per il 75% dei casi la cosa più importante per chi suona uno strumento musicale, è la conoscenza dello stesso: quindi si fa riferimento alla tecnica esecutiva. Un'altra dose di importanza viene attribuita al carattere del musicista, ovvero se il suo tocco sullo strumento è particolarmente lieve o deciso. Allo stesso modo è importante la concentrazione, infatti, se il più bravo violinista del mondo viene distratto continuamente durante l'esecuzione di un brano, è logico che il risultato della stessa, non sarà uno dei migliori. In poche righe siamo riusciti già a sviscerare i parametri più importanti che costituiscono il sistema SmartPlayer:

- Knowledge degree or playing ability: questa variabile, vista sottoforma di slider che l'utente può liberamente manipolare nell'applicazione, stabilisce il grado di conoscenza di cui l'esecutore è dotato, per l'esecuzione del brano (o in questo caso per la riproduzione del segnale audio). L'importanza di questo parametro, è quella di stabilire la quantità di ritardo introdotta sul segnale e di conseguenza, se si ricorda come funziona un effetto Chorus, la quantità di pitch modulation sullo stesso. Se un musicista ha un grado di conoscenza elevato del proprio strumento e del brano che sta suonando, abbassa notevolmente la probabilità di errore nell'esecuzione (in questo caso vista come cambiamenti di neuma del segnale o come introduzione di ritardi dovute ad incertezze), mentre se una persona non conosce lo strumento o il brano, tenderà ad introdurre delle incertezze nell'esecuzione, che nel 98% dei casi si tramutano in ritardi o note sbagliate.
- Force applied when play: questa variabile, stabilisce il "tocco" dell'esecutore. Se infatti, un musicista possiede un tocco lieve oppure possiede una timidezza che gli impedisce un'espressione corretta del brano, l'effetto introdotto dalla conoscenza o meno del brano, sarà piuttosto basso. È come, in un certo modo, avere un valore di Depth molto basso in un effetto Chorus. Viceversa, se si possiede un tocco deciso o l'esecutore è forte²¹, la conoscenza o meno del brano o dello strumento musicale, si farà maggiormente sentire.
- Neural Control when play: questa variabile, valuta il grado di concentrazione o meno di un esecutore. Ric conducendoci all'esempio di poche righe fa, anche il più bravo esecutore del mondo può fallire, se continuamente distratto e questo parametro tiene conto della distrazione stessa, sia essa interna, cioè dell'esecutore o esterna, cioè del resto del mondo.

²¹ Per forte si intende l'intensità del segnale e non la forza intrinseca dell'esecutore.

- **Roughness in the mood:** questa più che una variabile è la rappresentazione di uno stato di fatto. Se si guarda l'applicazione principale infatti, è l'unico controllo rappresentato con un checkbox. Il suddetto parametro, ci definisce l'andamento del carattere dell'esecutore. Se un esecutore è in generale una persona calma, pacata o comunque sa controllarsi, mostrerà, nel diagramma temporale dell'andamento del suo HBB²², un andamento "morbido". Viceversa, se un esecutore è agitato o aggressivo, avrà un andamento nello HBB, molto spigoloso.

Se potessimo utilizzare un'equazione per stimare la relazione tra questi parametri, scriveremo:

$$HBB(t) = Force_Applied(t) \cdot (Roughness_Mood(t) \cdot Knowledge(t) \cdot (1 - Neural_Focus(t)))$$

Cioè, l'andamento nel tempo del comportamento del cervello umano è dato dalla iterazione dei parametri definiti precedentemente, anch'essi in dipendenza dal tempo. Vediamo, a titolo di esempio, alcuni grafici che ci possono stabilire, alcuni tra i comportamenti più diffusi negli esecutori:

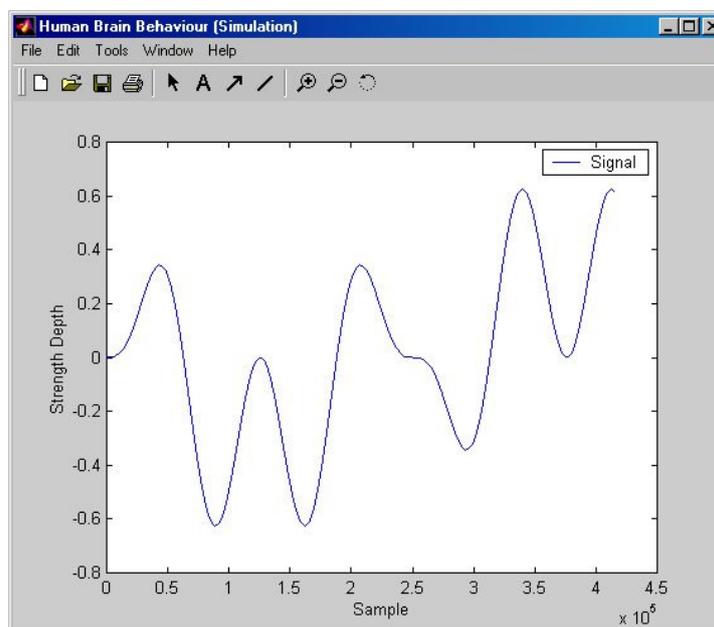


Figura 44: Questo è l'andamento dello HBB di un esecutore concentrato e con un grado di conoscenza medio dello strumento, calmo e con un tocco deciso

²² Acronimo di Human Brain Behaviour. D'ora in poi utilizzeremo la dicitura sotto forma di acronimo.

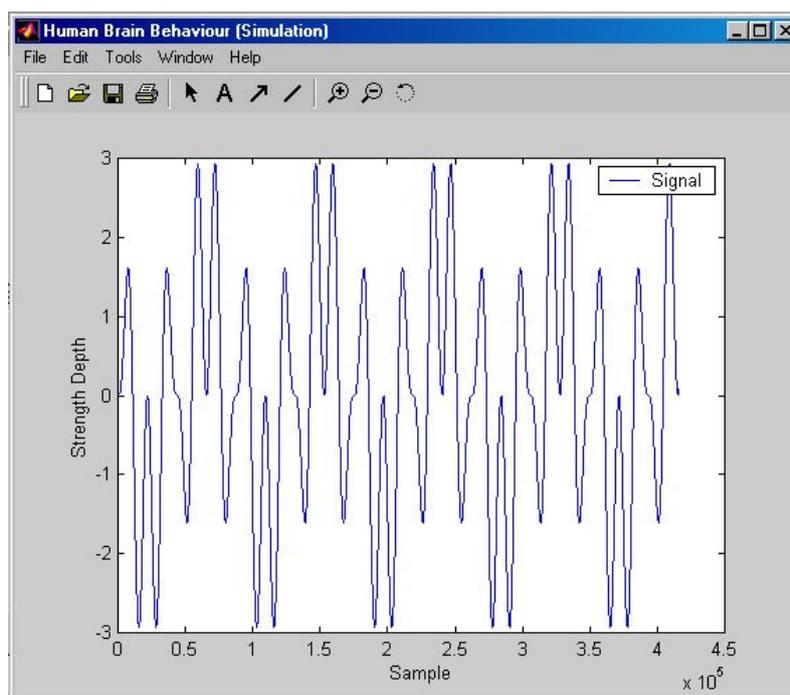


Figura 45: Cosa accade allo HBB dell'esecutore precedente, se viene distratto

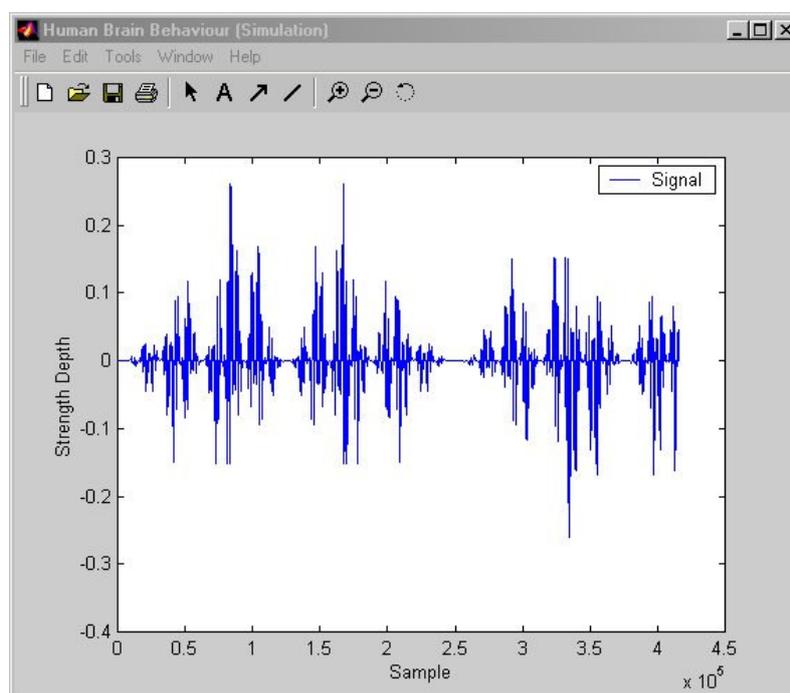


Figura 46: Stesso esecutore, ma con un comportamento agitato e aggressivo (Si notino i picchi nello HBB)

5.4.3 Implementazione in Matlab del sistema SmartPlayer

Riportiamo ora il codice del sistema di SmartPlayer, presente nel file *fcnSmart.m*, cercando di chiarire alcuni punti fondamentali. Il seguente algoritmo, deve essere compatibile con il resto dell'environment Nova – Mod, ed in particolare deve poter comunicare con gli effetti Chorus e Flanger – Phaser, i propri dati. Si utilizzerà quindi un file di interscambio *xchangesmart.mat* che passerà all'applicazione principale, i dati relativi allo HBB, che verranno interpretati dagli algoritmi di modulazione, come una normale forma d'onda. Il calcolo dello HBB, avviene però, in base ai parametri di ritardo massimo e di mixing, stabilito negli effetti di modulazione, quindi il file di interscambio sarà bidirezionale. Il file è diviso in base allo switchyard programming. Dopo avere ricavato i dati dagli effetti di modulazione relativi al ritardo massimo e alla lunghezza del file da processare, si calcola la forma dello HBB in base all'equazione precedente, dove si ha che: se il roughness in the mood parameter è nullo, allora il comportamento è stimato da tre stadi sinusoidali in sovrapposizione, in modo da garantire un comportamento pseudo – randomico sufficientemente smooth; viceversa, se il roughness in the mood è a uno, gli stadi sinusoidali in sovrapposizione sono nove, il che mi stabilisce dei picchi nel comportamento piuttosto elevati. Per passare i dati all'applicazione principale si scrive la forma d'onda nel file di interscambio, molto semplice...

```
%Function File esterno per i callback del sistema di simulazione umana
%
%written by Zambelli Cristian
function fcnSmart(action)
global lfw;
switch(action)
case 'plothbb'
    got=load('xchangesmart.mat');
    md=got.md;
    len=got.len;
    n=1:len+md;
    slidelh=findobj(gcf,'Tag','Slider1');
    slide1=get(slidelh,'Value')/10;
    slide2h=findobj(gcf,'Tag','Slider2');
    slide2=get(slide2h,'Value');
    slide3h=findobj(gcf,'Tag','Slider3');
    slide3=get(slide3h,'Value');
    checkh=findobj(gcf,'Tag','Checkbox1');
    check=get(checkh,'Value');
    rinv=1/slide1;
    data.samplerate=44100;
    ww=2*pi/round(rinv*data.samplerate);
```

```

lfw=zeros(size(n));
if check==0
    lfw(n)=slide2.*(sin(2*pi*ww.*n).*sin(4*pi*ww.*n).*sin(8*pi*ww.*n).*(slide1+(1-
slide3)));
elseif check==1

lfw(n)=slide2.*(sin(2*pi*ww.*n).*sin(4*pi*ww.*n).*sin(8*pi*ww.*n).*sin(16*pi*ww.*n).*sin(
32*pi*ww.*n).*sin(64*pi*ww.*n).*sin(128*pi*ww.*n).*sin(256*pi*ww.*n).*sin(512*pi*ww.*n).*(
slide1+(1-slide3)));
    end
    h1=figure('Name','Human Brain Behaviour (Simulation)','NumberTitle','off');
    plot(lfw);
    ylabel('Strength Depth');
    xlabel('Sample');
    legend('Signal');
case 'passtochorus'
    assignin('base','lfw',lfw);
    h=waitbar(0,'Passing data to Main Application. Please Wait...');
    for i=1:100,
        waitbar(i/100);
    end
    save('xchangesmart');
    close(h);
case 'close'
    close(gcf);
end

```

Da notare, che questa applicazione deve agire in background rispetto agli effetti di modulazione. Siccome Matlab non prevede una soluzione di applicazioni concorrenti in background, utilizziamo una warning dialog, che rimane in attesa del passaggio dei dati da un sistema all'altro, alla pressione del pulsante di OK.

5.5 Sviluppo degli algoritmi di modulazione con Matlab

Costruiamo il set di algoritmi di modulazione, sempre tramite il software GUIDE di Matlab, utilizzando la funzionalità di stesura degli script in linguaggio Matlab. Ancora una volta si dovrà utilizzare un'applicazione che interagisca con il resto del sistema e che garantisca una facile iterazione con l'utente tramite una semplice interfaccia GUI. Come abbiamo più volte ribadito in questo capitolo, sarà necessario un file di interscambio *xchangedata.mat*, che permetta il passaggio dei dati provenienti dal sistema di Signal Acquisition all'applicazione, in modo che i dati del segnale audio da processare, siano valutati una sola volta all'inizio dell'algoritmo, in modo da non

perdere tempo durante l'esecuzione. È necessario predisporre ora, l'environment Nova – Mod Tools, per l'inglobamento del nuovo set di algoritmi per il sound processing. Costruiremo quindi un pulsante sull'estrema destra del sistema che chiameremo “Modulation Algorithm”, che alla sua pressione, visualizza una listbox che ci permetta di scegliere quale algoritmo utilizzare, ricordando che il codice di questa funzionalità è uguale a quello della listbox di scelta degli algoritmi di modulazione diretta del segnale e che quindi omettiamo per brevità. Una volta che le applicazioni relative agli algoritmi da implementare sono state create, valutiamo i controlli e le operazioni che essi ci offrono:

- Real Polyphonic Chorus:

- Voice nr. X²³: è il controllo stabilito con una checkbox, che permette di definire all'algoritmo, se la X – esima voce, deve essere valutata nel calcolo del processing del segnale audio. La pressione o meno di questo controllo, influisce anche sulla visualizzazione delle LFW utilizzate per l'applicazione dell'effetto Chorus.
- Modulating LFW: si tratta di un popup menù a tre scelte. La prima scelta (LFO previously simulated and saved on a file) consente di ricavare la LFW da un LFO già simulato con il tool LFO – Maker e salvato su file; la seconda scelta (Build my own), consente di lanciare un'istanza del tool LFO – Maker, in modo che l'utente sia in grado di creare una propria forma d'onda, qualora essa non sia già presente su disco; la terza scelta (Use SmartPlayer Technique), consente di utilizzare il sistema di simulazione del comportamento umano SmartPlayer, ma richiede l'utilizzo del file di interscambio *xchangesmart.mat* e il backgrounding dell'applicazione principale.
- Gathered LFW Type: è il controllo statico che visualizza la tipologia di forma d'onda utilizzata per l'applicazione dell'effetto Chorus al segnale audio. Qualora sia stata scelta la funzionalità di SmartPlayer, questo controllo visualizza la dicitura “Smart”.
- Gathered LFW Rate: è il controllo statico che visualizza la frequenza della forma d'onda utilizzata per l'applicazione dell'effetto Chorus. Qualora sia stata scelta la funzionalità di SmartPlayer, questo controllo visualizza la dicitura “Smart”.

²³ Dove X corrisponde al numero di voce attualmente selezionata.

- **Gathered LFW Depth:** è il controllo statico che visualizza l'ampiezza di picco della forma d'onda utilizzata per l'applicazione dell'effetto Chorus, ricordando che, maggiore è l'ampiezza di essa, maggiore sarà l'effetto sul ritardo e sulla pitch modulation introdotta. Qualora sia stata scelta la funzionalità di SmartPlayer, questo controllo visualizza la dicitura "Smart".
- **Vibrato Rate:** è il controllo dinamico che sotto forma di editbox consente di implementare una funzionalità molto importante per il sistema di Chorusing: il Vibrato. L'applicazione di una forma d'onda normale, permette di variare in modo periodico il tempo di ritardo della voce del segnale audio, ma bisogna anche tenere conto delle possibili variazioni sull'ampiezza di questa forma d'onda, dovuta tipicamente a modulazione. Il Vibrato è quindi una sorta di modulazione di ampiezza sinusoidale della forma d'onda utilizzata per il Chorusing, in cui la frequenza di modulazione è controllata da questo parametro. Con valori di frequenza di Vibrato molto bassi, l'effetto sarà quasi impercettibile, mentre con valori molto elevati, le voci saranno caratterizzate da un andamento tremolante.
- **Base Delay in msec:** è il controllo dinamico visto sotto forma di editbox, in cui si va ad inserire il ritardo base in millisecondi, per il calcolo delle forme d'onde di modulazione del ritardo e per l'applicazione del processo di Chorusing. Valori consigliati per un effetto Chorus ottimo, variano tra i 20 e i 40 millisecondi.
- **Enable Stereophonia (only for Monaural Sources):** quando una sorgente sonora a cui si applica l'effetto Chorus è di tipo Monaurale, il suono in uscita dall'effetto, risulterà particolarmente "denso" e scordato. Questo controllo, consente di applicare la stereofonia ad un segnale monaurale, per trasformarlo in un segnale stereofonico. Quando la checkbox relativa a questo controllo è premuta, i controlli Voice nr. X, si trasformano in definitori di punti dello spazio stereofonico. Pensiamo allo spazio stereofonico come ad un semicerchio in cui gli estremi rappresentano il segnale all'estrema sinistra e all'estrema destra dell'ascoltatore, che si troverà nel centro esatto del semicerchio. Questo semicerchio sarà diviso in un numero equispaziato di settori, definito dal numero di voci utilizzate per l'applicazione dell'effetto. Ogni voce quindi, uscirà da un punto dello spazio, che verrà identificato dalla posizione nel semicerchio. Se il numero di voci è dispari, allora ci sarà sempre una voce diretta verso il centro del semicerchio, cioè diretta verso l'ascoltatore; viceversa, se le voci saranno pari, il semicerchio risulterà essere correttamente equispaziato.

- Channel to process when the source signal is stereophonic: questo popup menù, consente di ricavare da un segnale stereofonico, il canale da processare. Ciò ammette quindi, un processing separato sia per il canale sinistro che per quello destro, di un segnale stereofonico.
- Retrieve Signal: è il pulsante che consente di ricavare la forma del segnale audio non processato e lo visualizza in un subplot in funzione del tempo ed in valori dei campioni normalizzati nell'intervallo $[-1;1)$.
- Plot LFW's: è il pulsante che consente di effettuare il plot delle forme d'onde utilizzate per il processing del segnale audio. Il plot complessivo è diviso in 12 subplot che vengono assegnati, ognuno ad una diversa forma d'onda, in modo che ad es. alla posizione 4 del subplot, corrisponde la LFW nr.4. Se essa non viene utilizzata nel processing, allora non viene visualizzata.
- Apply Chorus: è il pulsante che avvia l'algoritmo di processing tramite l'effetto Chorus. Al termine dello svolgimento dell'algoritmo, segnalato dalla presenza di una waitbar che ne visualizza la progressione computazionale, appare il subplot al di sotto del segnale non processato, con il nuovo segnale affetto da Chorus. Questo subplot contiene la forma d'onda del segnale in funzione del tempo e con valori normalizzati nell'intervallo $[-1;1)$ per i campioni.
- Save result on a file: è il pulsante che consente di salvare il risultato del processing tramite Chorus su un file audio in formato Microsoft WAVE PCM FORMAT o Motion Picture Expert Group (MPEG-1) Layer 3 a 256 Kbps.
- Play Original Signal: è il pulsante che esegue il playback del segnale originale non processato. L'avanzamento del playback è segnalato dall'avanzamento di una waitbar.
- Play Chorused Signal: è il pulsante che esegue il playback del segnale processato con effetto Chorus. L'avanzamento del playback è segnalato dall'avanzamento di una waitbar.
- Credits: è il pulsante che visualizza le informazioni relative al creatore dell'applicazione

- Final Mix: è lo slider che permette di stabilire la proporzione di segnale non processato (Dry) o processato (Wet), nel risultato finale. Se lo slider è completamente a destra, allora si ottiene solamente il segnale Wet, viceversa si ottiene solo il segnale Dry. Per un migliore effetto Chorus, si consigliano valori tendenti al Dry, in modo da enfatizzare lievemente la pitch modulation, ed ottenere un effetto molto spazioso, ma poco “scordato”.
- Close: è il pulsante che consente la chiusura dell’applicazione, liberando la memoria dagli handle dei controlli.

- Classical Flanger – Phaser:

- Source of unique LFW: è il popup menù a tre scelte che consente di definire l’unica forma d’onda necessaria al Flanger – Phaser, per compiere il processing del segnale. La prima scelta (LFO previously simulated and saved on a file) consente di ricavare la LFW da un LFO già simulato con il tool LFO – Maker e salvato su file; la seconda scelta (Build my own), consente di lanciare un’istanza del tool LFO – Maker, in modo che l’utente sia in grado di creare una propria forma d’onda, qualora essa non sia già presente su disco; la terza scelta (Use SmartPlayer Technique), consente di utilizzare il sistema di simulazione del comportamento umano SmartPlayer, ma richiede l’utilizzo del file di interscambio *xchangesmart.mat* e il backgrounding dell’applicazione principale.
- Gathered LFW Type: è il controllo statico che visualizza la tipologia di forma d’onda utilizzata per l’applicazione dell’effetto Flanger – Phaser al segnale audio. Qualora sia stata scelta la funzionalità di SmartPlayer, questo controllo visualizza la dicitura “Smart”.
- Gathered LFW Rate: è il controllo statico che visualizza la frequenza della forma d’onda utilizzata per l’applicazione dell’effetto Flanger – Phaser. Qualora sia stata scelta la funzionalità di SmartPlayer, questo controllo visualizza la dicitura “Smart”.
- Gathered LFW Depth: è il controllo statico che visualizza l’ampiezza di picco della forma d’onda utilizzata per l’applicazione dell’effetto Flanger – Phaser, ricordando che, maggiore è l’ampiezza di essa, maggiore sarà l’effetto sul ritardo e sui notches introdotti nella risposta

in frequenza. Qualora sia stata scelta la funzionalità di SmartPlayer, questo controllo visualizza la dicitura “Smart”.

- **Vibrato Rate:** è il controllo dinamico che sotto forma di editbox consente di implementare una funzionalità molto importante per il sistema di Chorusing, che poi muterà per il Flanging: il Vibrato. Con valori di frequenza di Vibrato molto bassi, l'effetto sarà quasi impercettibile, mentre con valori molto elevati, le voci saranno caratterizzate da un andamento tremolante.
- **Base Delay in msec:** è il controllo dinamico visto sotto forma di editbox, in cui si va ad inserire il ritardo base in millisecondi, per il calcolo delle forme d'onde di modulazione del ritardo e per l'applicazione del processo di Flanging. Valori consigliati per un effetto Flanger – Phaser ottimo, variano tra 1 e 10 millisecondi.
- **Enable Stereo Flanging (only for monaural sources):** questo controllo sotto forma di checkbox, consente di implementare la funzionalità di stereofonia. Tutto ciò è ottenuto, interponendo una seconda voce al segnale originale e al segnale processato con la prima LFW, con la seconda voce sfasata di una quantità definita dall'utente, in modo da ottenere una sonorità di Flanger più ampia, perché il suono che arriva alle orecchie è differente per ogni lato.
- **Channel to process when the source signal is stereophonic:** questo popup menù, consente di ricavare da un segnale stereofonico, il canale da processare. Ciò ammette quindi, un processing separato sia per il canale sinistro che per quello destro, di un segnale stereofonico.
- **Flanging Depth:** è il controllo dinamico che stabilisce la percentuale di segnale processato nel segnale audio finale. Valori tendenti a 1, daranno un segnale di uscita che comprende completamente il segnale processato con effetto Flanger – Phaser; viceversa, valori tendenti a 0, daranno in uscita solo il segnale originale.
- **Stereo Phasing in degrees (only when Stereo Flanger is enabled):** quando il sistema di stereofonia del segnale processato è abilitato, è necessario stabilire di quanti gradi la seconda voce deve essere sfasata rispetto alla prima. Questo garantisce spaziosità al suono

finale e per un effetto ottimo si consiglia di utilizzare uno sfasamento tra le voci di 90 gradi, in modo da ottenere il suono del classico Quadrature – Flanger.

- **Feedback Gain:** il sistema di Flanging è caratterizzato dalla possibilità di riportare l'uscita della linea di ritardo in ingresso alla linea stessa, formando un percorso di retroazione. Quando si usa il feedback, si possono ottenere delle sonorità particolarmente intense, dovute all'aumento del guadagno. Questo controllo dinamico, stabilisce il guadagno da imporre al percorso di retroazione (si consiglia valori <1).
- **Feedback Subtraction:** questa checkbox permette di definire se il segnale retroazionato deve essere sottratto o sommato al segnale di ingresso della linea di ritardo. Quando la checkbox è attiva, il valore di Feedback Gain imposto, sarà negativo.
- **Retrieve Signal:** è il pulsante che consente di ricavare la forma del segnale audio non processato e lo visualizza in un subplot in funzione del tempo ed in valori dei campioni normalizzati nell'intervallo $[-1;1)$.
- **Plot LFW's:** è il pulsante che consente di effettuare il plot delle forme d'onde utilizzate per il processing del segnale audio. Il plot complessivo è diviso in 2 subplot che vengono assegnati, ognuno ad una diversa forma d'onda, in modo che ad es. alla posizione 2 del subplot, corrisponde la LFW nr.2. Se essa non viene utilizzata nel processing, allora non viene visualizzata e il plot risulta unico.
- **Apply Flanger:** è il pulsante che avvia l'algoritmo di processing tramite l'effetto Flanger. Al termine dello svolgimento dell'algoritmo, segnalato dalla presenza di una waitbar che ne visualizza la progressione computazionale, appare il subplot al di sotto del segnale non processato, con il nuovo segnale affetto da Flanger. Questo subplot contiene la forma d'onda del segnale in funzione del tempo e con valori normalizzati nell'intervallo $[-1;1)$ per i campioni.
- **Save result on a file:** è il pulsante che consente di salvare il risultato del processing tramite Chorus su un file audio in formato Microsoft WAVE PCM FORMAT o Motion Picture Expert Group (MPEG-1) Layer 3 a 256 Kbps.

- Play Original Signal: è il pulsante che esegue il playback del segnale originale non processato. L'avanzamento del playback è segnalato dall'avanzamento di una waitbar.
- Play Flanged Signal: è il pulsante che esegue il playback del segnale processato con effetto Flanger – Phaser. L'avanzamento del playback è segnalato dall'avanzamento di una waitbar.
- Credits: è il pulsante che visualizza le informazioni relative al creatore dell'applicazione
- Close: è il pulsante che consente la chiusura dell'applicazione, liberando la memoria dagli handle dei controlli.

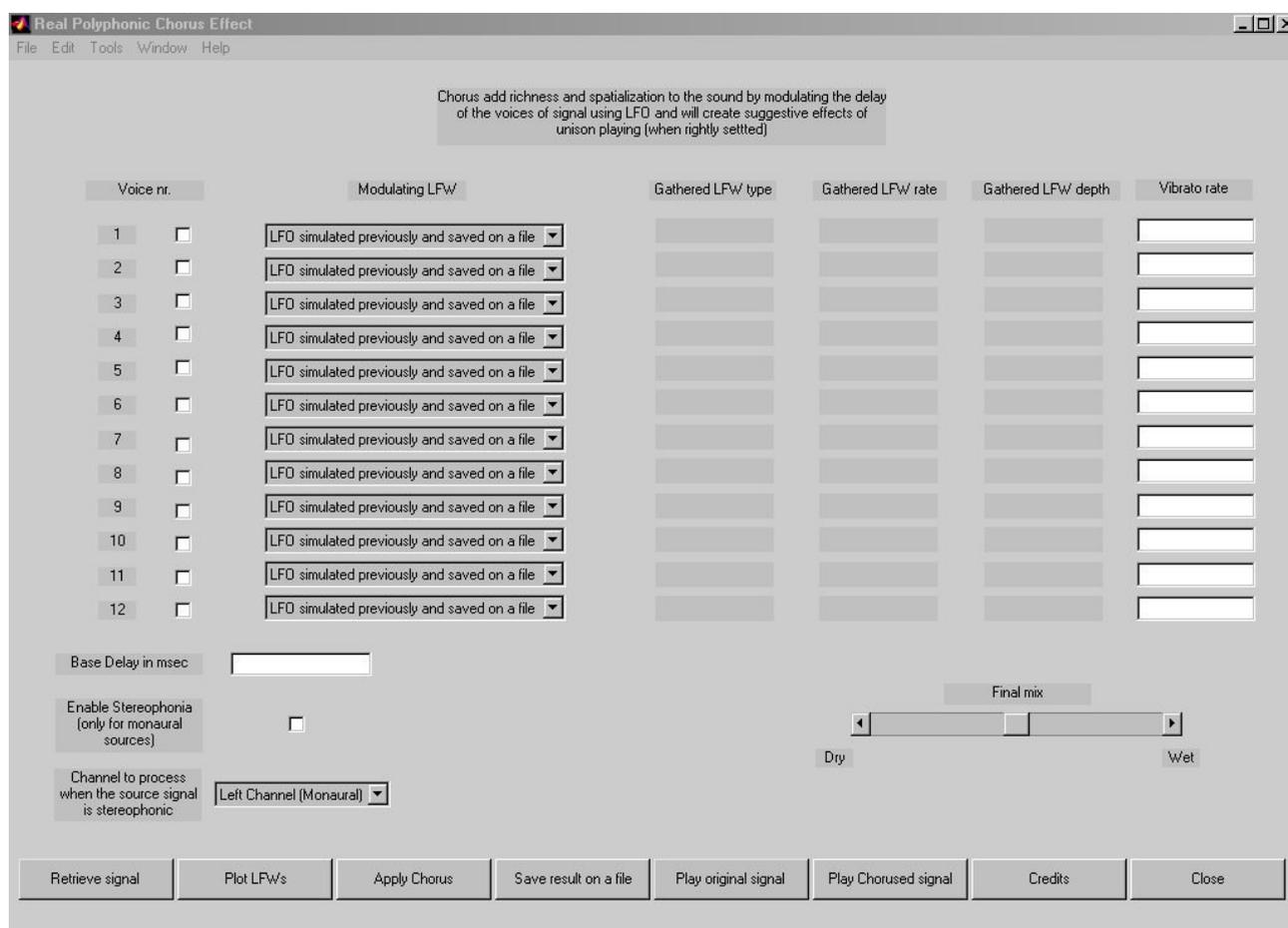


Figura 47: Come appare la struttura dell'applicazione GUI per il Chorus

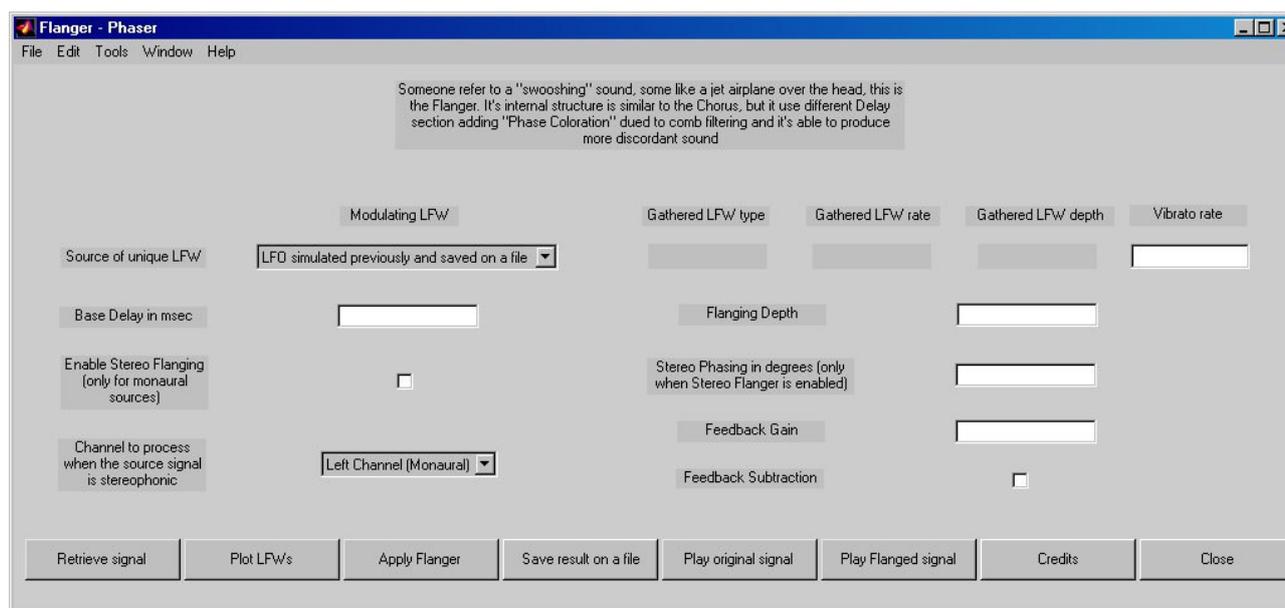


Figura 48: Come appare la struttura dell'applicazione GUI per il Flanger – Phaser

Partiamo ora con l'analisi del lunghissimo codice che costituirà le applicazioni degli algoritmi di Chorus e di Flanger – Phaser al segnale audio in ingresso ad essi, ricordando che i listati appartengono rispettivamente ai file *fcnChorus.m* e *fcnPFlanger.m* e che sono disponibili nel CD-ROM allegato alla tesi, nella directory *../Codice/Matlab* con estensione *.m*. Analizziamo l'algoritmo del sistema di Chorus.

Questa è la parte di dichiarazione delle variabili globali da usare per l'applicazione e si ricavano i dati del segnale acquisito, tramite il file di interscambio utilizzando dall'environment *xchangedata.mat*.

```
%Function File esterno per i callback del chorus
%
%written by Zambelli Cristian
function fcnChorus(action)
global x data t md lfw1 lfw2 lfw3 lfw4 lfw5 lfw6 lfw7 lfw8 lfw9 lfw10 lfw11 lfw12 am1 am2
am3 am4 am5 am6 am7 am8 am9 am10 am11 am12 popup1 popup2 popup3 popup4 popup5 popup6
popup7 popup8 popup9 popup10 popup11 popup12 y h1;
data=load('xchangedata.mat');
```

Qui comincia la consueta switchyard programming. Questo case permette di ricavare il segnale audio acquisito e di plottarlo in una nuova figura con i valori normalizzati nell'intervallo [-1;+1).

```
switch(action)
```

```

case 'retrieve'
    t=( [1:data.count]/data.samplerate);
    x=data.y;
    h=waitbar(0, 'Retrieving the Original Signal...');
    for i=1:100,
        waitbar(i/100);
    end
    close(h);
    if data.channel==1
        if data.quantize==8
            z=(x-128)/128;
        elseif data.quantize==16
            z=x/32768;
        end
    elseif data.channel==2
        chanh=findobj(gcf, 'Tag', 'PopupMenu20');
        chan=get(chanh, 'Value');
        if chan==1
            x=data.lefty;
        elseif chan==2
            x=data.righty;
        end
        if data.quantize==8
            z=(x-128)/128;
        elseif data.quantize==16
            z=x/32768;
        end
    end
    end
    h1=figure('Name', 'Signal Plot', 'NumberTitle', 'off');
    subplot(211);
    plot(t, z);
    title('Dry Signal');
    ylabel('Normalized Values');
    if data.channel==1
        legend('Monaural Signal');
    elseif data.channel==2
        if chan==1
            legend('Left Channel');
        elseif chan==2
            legend('Right Channel');
        end
    end
end

```

Questo case permette di effettuare il playback del segnale senza l'applicazione dell'effetto, tramite la funzione *sound*. Così come per la funzione usata nel Signal Acquisition System, si inserisce una waitbar che progredisce al progredire del playback del file.

```
case 'playdry'
    tr=data.count/data.samplerate;
    inc=0.01;
    texe=clock;soundsc(x,data.samplerate,data.quantize);e=etime(clock,texe);
    texe2=clock;h=waitbar(0,'Playback time (Dry)...');e2=etime(clock,texe2);
    for i=0:inc:(tr-e-e2),
        waitbar(i/(tr-e-e2));
        pause(inc);
    end
    close(h);
```

Questo case permette di effettuare il playback del segnale con l'applicazione dell'effetto, tramite la funzione *sound*. Così come per la funzione usata nel Signal Acquisition System, si inserisce una waitbar che progredisce al progredire del playback del file.

```
case 'playwet'
    tr=data.count/data.samplerate;
    inc=0.01;
    texe=clock;sound(y,data.samplerate,data.quantize);e=etime(clock,texe);
    texe2=clock;h=waitbar(0,'Playback time (Wet)...');e2=etime(clock,texe2);
    for i=0:inc:(tr-e-e2),
        waitbar(i/(tr-e-e2));
        pause(inc);
    end
    close(h);
```

Questo case permette di ricavare le informazioni relative alla LFW da utilizzare per l'operazione di modulazione. Se il popup menù relativo alla sorgente di acquisizione della LFW è di valore pari a 1, allora apri una dialog box che permette di scegliere il file con estensione *.lfo* e ne ricava tutti i parametri per poi eseguirne il plot, altrimenti lancia un'istanza di LFO – Maker per la creazione della LFW. La creazione della LFW avviene a seconda del valore di Base Delay utilizzato e va detto che l'unica LFW che non può essere utilizzata per questo algoritmo (a causa della sua natura matematica) è l'esponenziale – sinusoidale. Se invece il valore del popup menù è pari a 3, allora si lancia un'istanza del sistema di SmartPlayer, passando i parametri del segnale audio da valutare e il valore del Base Delay nel file *xchangesmart.m*; manda in background l'applicazione e attende la

pressione del pulsante OK, per riprendere i dati della LFW modificati a seconda dello HBB scelto con il sistema di SmartPlayer. Questo codice viene replicato per tutte le 12 voci.

```

case 'getvoicel'
    popuplhandle=findobj(gcf,'Tag','PopupMenu1');
    popupl=get(popuplhandle,'Value');
    mdh=findobj(gcf,'Tag','EditText20');
    md=ceil((str2num(get(mdh,'String'))/1000)*data.samplerate);
    if popupl==1
        [filename,filepath]=uigetfile('*.lfo','Get LFW from which file?');
        path=[filepath];
        name=[filename];
        fid=fopen(strcat(path,name),'r');
        lfwtype=fscanf(fid,'%s\n%');
        rsst=fscanf(fid,'%s\n%');
        wall=fscanf(fid,'%s\n%');
        rate=fscanf(fid,'%s\n%');
        amp=fscanf(fid,'%s\n%');
        phase=fscanf(fid,'%s\n%');
        offset=fscanf(fid,'%s\n%');
        damping=fscanf(fid,'%s\n%');
        rectif=fscanf(fid,'%s\n%');
        status=fclose(fid);
        h=waitbar(0,'Gathering LFO Parameters. Please Wait...');
        x=data.y;
        n=1:length(x)+md;
    am1=str2num(amp);
    damp=str2num(damping);
    r=str2num(rate);
    rinv=1/r;
    ww=2*pi/round(rinv*data.samplerate);
    p=str2num(phase);
    static=findobj(gcf,'Tag','StaticText21');
    set(static,'String',rate);
    static1=findobj(gcf,'Tag','StaticText31');
    set(static1,'String',amp);
    if strcmp(lfwtype,'sin')
        stati=findobj(gcf,'Tag','StaticText11');
        set(stati,'String','Sine');
        if strcmp(rsst,'1')
            p=(p*2*pi)/360;
            g=(1-exp(-(n*damp)).*sin(ww.*n+p));
            a=int32(g);
        elseif strcmp(rsst,'0')
            p=(p*2*pi)/360;
            a=(1-exp(-(n*damp)).*sin(ww.*n+p));

```

```

end
if strcmp(wall, '1')
    b=a+0.1*sin(2*pi*50*t);
elseif strcmp(wall, '0')
    b=a;
end
if strcmp(rectif, '1')
    lfw1=double(abs(b));
elseif strcmp(rectif, '0')
    lfw1=double(b);
end
elseif strcmp(lfwtype, 'cos')
    stati=findobj(gcf, 'Tag', 'StaticText11');
    set(stati, 'String', 'Cosine');
    if strcmp(rsst, '1')
        p=(p*2*pi)/360;
        g=(1-exp(-(n*damp)).*cos(ww.*n+p));
        a=int32(g);
    elseif strcmp(rsst, '0')
        p=(p*2*pi)/360;
        a=(1-exp(-(n*damp)).*cos(ww.*n+p));
    end
if strcmp(wall, '1')
    b=a+0.1*sin(2*pi*50*t);
elseif strcmp(wall, '0')
    b=a;
end
if strcmp(rectif, '1')
    lfw1=double(abs(b));
elseif strcmp(rectif, '0')
    lfw1=double(b);
end
elseif strcmp(lfwtype, 'tri')
    stati=findobj(gcf, 'Tag', 'StaticText11');
    set(stati, 'String', 'Triangular');
    if strcmp(rsst, '1')
        g=(1-exp(-(n*damp)).*sawtooth(ww.*n,p+0.5));
        a=int32(g);
    elseif strcmp(rsst, '0')
        a=(1-exp(-(n*damp)).*sawtooth(ww.*n,p+0.5));
    end
if strcmp(wall, '1')
    b=a+0.1*sin(2*pi*50*t);
elseif strcmp(wall, '0')
    b=a;
end
if strcmp(rectif, '1')

```

```

        lfw1=double(abs(b));
elseif strcmp(rectif,'0')
        lfw1=double(b);
end
elseif strcmp(lfwtype,'squ')
        stati=findobj(gcf,'Tag','StaticText11');
        set(stati,'String','Square');
        if strcmp(rsst,'1')
                g=(1-exp(-(n*damp)).*square(ww.*n,p));
                a=int32(g);
        elseif strcmp(rsst,'0')
                a=(1-exp(-(n*damp)).*square(ww.*n,p));
        end
if strcmp(wall,'1')
        b=a+0.1*sin(2*pi*50*t);
elseif strcmp(wall,'0')
        b=a;
end
if strcmp(rectif,'1')
        lfw1=double(abs(b));
elseif strcmp(rectif,'0')
        lfw1=double(b);
end
elseif strcmp(lfwtype,'exp')
        h1=warndlg('Sorry but this is not a type of LFW usable with this kind of
algorithm','Warning!!!');
        uiwait(h1);
elseif strcmp(lfwtype,'saw')
        stati=findobj(gcf,'Tag','StaticText11');
        set(stati,'String','Sawtooth');
        if strcmp(rsst,'1')
                g=(1-exp(-(n*damp)).*sawtooth(ww.*n,p));
                a=int32(g);
        elseif strcmp(rsst,'0')
                a=(1-exp(-(n*damp)).*sawtooth(ww.*n,p));
        end
if strcmp(wall,'1')
        b=a+0.1*sin(2*pi*50*t);
elseif strcmp(wall,'0')
        b=a;
end
if strcmp(rectif,'1')
        lfw1=double(abs(b));
elseif strcmp(rectif,'0')
        lfw1=double(b);
end
elseif strcmp(lfwtype,'ran')

```

```

stati=findobj(gcf,'Tag','StaticText11');
set(stati,'String','Pseudo Random');
if strcmp(rsst,'1')
    g=(1-exp(-(n*damp)).*sin(rand(1)*ww.*n+p));
    a=int32(g);
elseif strcmp(rsst,'0')
    a=(1-exp(-(n*damp)).*sin(rand(1)*ww.*n+p));
end
if strcmp(wall,'1')
    b=a+0.1*sin(2*pi*50*t);
elseif strcmp(wall,'0')
    b=a;
end
if strcmp(rectif,'1')
    lfw1=double(abs(b));
elseif strcmp(rectif,'0')
    lfw1=double(b);
end
end
for i=1:100,
    waitbar(i/100);
end
close(h);
elseif popup1==2
    lfomaker;
elseif popup1==3
    static=findobj(gcf,'Tag','StaticText21');
    set(static,'String','Smart');
    static1=findobj(gcf,'Tag','StaticText31');
    set(static1,'String','Smart');
    stati=findobj(gcf,'Tag','StaticText11');
    set(stati,'String','Smart');
    if data.channel==1
        len=data.count;
    elseif data.channel==2
        len=(data.count)/2;
    end
    assignin('base','maxdelays',md);
    assignin('base','lengthfile',len);
    assignin('base','samp',data.samplerate);
    h=waitbar(0,'Passing data to SmartPlayer System');
    save('xchangesmart');
    for i=1:100,
        waitbar(i/100);
    end
    close(h);
    smartplayer;

```

```

    h3=warndlg('Please maintain this Dialog Opened (dont push OK) until you have
finished to use the SmartPlayer System (he has the control of the entire system
now!!!)', 'WARNING!!!');
    uiwait(h3);
    passed=load('xchangesmart.mat');
    lfw1=passed.lfw;
    lfw1=abs(lfw1)';
end
case 'getvoice2'
    popup2handle=findobj(gcf,'Tag','PopupMenu2');
    popup2=get(popup2handle,'Value');
    mdh=findobj(gcf,'Tag','EditText20');
    md=ceil((str2num(get(mdh,'String'))/1000)*data.samplerate);
    if popup2==1
        [filename,filepath]=uigetfile('*.lfo','Get LFW from which file?');
        path=[filepath];
        name=[filename];
        fid=fopen(strcat(path,name),'r');
        lfwtype=fscanf(fid,'%s\n%');
        rsst=fscanf(fid,'%s\n%');
        wall=fscanf(fid,'%s\n%');
        rate=fscanf(fid,'%s\n%');
        amp=fscanf(fid,'%s\n%');
        phase=fscanf(fid,'%s\n%');
        offset=fscanf(fid,'%s\n%');
        damping=fscanf(fid,'%s\n%');
        rectif=fscanf(fid,'%s\n%');
        status=fclose(fid);
        h=waitbar(0,'Gathering LFO Parameters. Please Wait...');
        x=data.y;
        n=1:length(x)+md;
    am2=str2num(amp);
    damp=str2num(damping);
    r=str2num(rate);
    rinv=1/r;
    ww=2*pi/round(rinv*data.samplerate);
    p=str2num(phase);
    static=findobj(gcf,'Tag','StaticText22');
    set(static,'String',rate);
    static1=findobj(gcf,'Tag','StaticText32');
    set(static1,'String',amp);
    if strcmp(lfwtype,'sin')
        stati=findobj(gcf,'Tag','StaticText12');
        set(stati,'String','Sine');
        if strcmp(rsst,'1')
            g=(1-exp(-(n*damp)).*sin(ww.*n+p));
            a=int32(g);

```

```

elseif strcmp(rsst, '0')
    a=(1-exp(-(n*damp)).*sin(ww.*n+p));
end
if strcmp(wall, '1')
    b=a+0.1*sin(2*pi*50*t);
elseif strcmp(wall, '0')
    b=a;
end
if strcmp(rectif, '1')
    lfw2=double(abs(b));
elseif strcmp(rectif, '0')
    lfw2=double(b);
end
elseif strcmp(lfwtype, 'cos')
    stati=findobj(gcf, 'Tag', 'StaticText12');
    set(stati, 'String', 'Cosine');
    if strcmp(rsst, '1')
        g=(1-exp(-(n*damp)).*cos(ww.*n+p));
        a=int32(g);
    elseif strcmp(rsst, '0')
        a=(1-exp(-(n*damp)).*cos(ww.*n+p));
    end
    if strcmp(wall, '1')
        b=a+0.1*sin(2*pi*50*t);
    elseif strcmp(wall, '0')
        b=a;
    end
    if strcmp(rectif, '1')
        lfw2=double(abs(b));
    elseif strcmp(rectif, '0')
        lfw2=double(b);
    end
elseif strcmp(lfwtype, 'tri')
    stati=findobj(gcf, 'Tag', 'StaticText12');
    set(stati, 'String', 'Triangular');
    if strcmp(rsst, '1')
        g=(1-exp(-(n*damp)).*sawtooth(ww.*n, p+0.5));
        a=int32(g);
    elseif strcmp(rsst, '0')
        a=(1-exp(-(n*damp)).*sawtooth(ww.*n, p+0.5));
    end
    if strcmp(wall, '1')
        b=a+0.1*sin(2*pi*50*t);
    elseif strcmp(wall, '0')
        b=a;
    end
    if strcmp(rectif, '1')

```

```

        lfw2=double(abs(b));
elseif strcmp(rectif,'0')
    lfw2=double(b);
end
elseif strcmp(lfwtype,'squ')
    stati=findobj(gcf,'Tag','StaticText12');
    set(stati,'String','Square');
    if strcmp(rsst,'1')
        g=(1-exp(-(n*damp)).*square(ww.*n,p));
        a=int32(g);
elseif strcmp(rsst,'0')
    a=(1-exp(-(n*damp)).*square(ww.*n,p));
end
if strcmp(wall,'1')
    b=a+0.1*sin(2*pi*50*t);
elseif strcmp(wall,'0')
    b=a;
end
if strcmp(rectif,'1')
    lfw2=double(abs(b));
elseif strcmp(rectif,'0')
    lfw2=double(b);
end
elseif strcmp(lfwtype,'exp')
    h1=warndlg('Sorry but this is not a type of LFW usable with this kind of
algorithm','Warning!!!');
    uiwait(h1);
elseif strcmp(lfwtype,'saw')
    stati=findobj(gcf,'Tag','StaticText12');
    set(stati,'String','Sawtooth');
    if strcmp(rsst,'1')
        g=(1-exp(-(n*damp)).*sawtooth(ww.*n,p));
        a=int32(g);
elseif strcmp(rsst,'0')
    a=(1-exp(-(n*damp)).*sawtooth(ww.*n,p));
end
if strcmp(wall,'1')
    b=a+0.1*sin(2*pi*50*t);
elseif strcmp(wall,'0')
    b=a;
end
if strcmp(rectif,'1')
    lfw2=double(abs(b));
elseif strcmp(rectif,'0')
    lfw2=double(b);
end
elseif strcmp(lfwtype,'ran')

```

```

stati=findobj(gcf,'Tag','StaticText12');
set(stati,'String','Pseudo Random');
if strcmp(rsst,'1')
    g=(1-exp(-(n*damp)).*sin(rand(1)*ww.*n+p));
    a=int32(g);
elseif strcmp(rsst,'0')
    a=(1-exp(-(n*damp)).*sin(rand(1)*ww.*n+p));
end
if strcmp(wall,'1')
    b=a+0.1*sin(2*pi*50*t);
elseif strcmp(wall,'0')
    b=a;
end
if strcmp(rectif,'1')
    lfw2=double(abs(b));
elseif strcmp(rectif,'0')
    lfw2=double(b);
end
end
for i=1:100,
    waitbar(i/100);
end
close(h);
elseif popup2==2
    lfomaker;
elseif popup2==3
    static=findobj(gcf,'Tag','StaticText22');
    set(static,'String','Smart');
    static1=findobj(gcf,'Tag','StaticText32');
    set(static1,'String','Smart');
    stati=findobj(gcf,'Tag','StaticText12');
    set(stati,'String','Smart');
    if data.channel==1
        len=data.count;
    elseif data.channel==2
        len=(data.count)/2;
    end
    assignin('base','maxdelays',md);
    assignin('base','lengthfile',len);
    assignin('base','samp',data.samplerate);
    h=waitbar(0,'Passing data to SmartPlayer System');
    save('xchangesmart');
    for i=1:100,
        waitbar(i/100);
    end
    close(h);
    smartplayer;

```

```

h3=warndlg('Please maintain this Dialog Opened (dont push OK) until you have
finished to use the SmartPlayer System (he has the control of the entire system
now!!!)', 'WARNING!!!');
    uiwait(h3);
    passed=load('xchangesmart.mat');
    lfw2=passed.lfw;
    lfw2=abs(lfw2)';
end
case 'getvoice3'
    popup3handle=findobj(gcf, 'Tag', 'PopupMenu3');
    popup3=get(popup3handle, 'Value');
    mdh=findobj(gcf, 'Tag', 'EditText20');
    md=ceil((str2num(get(mdh, 'String'))/1000)*data.samplerate);
    if popup3==1
        [filename,filepath]=uigetfile('*.lfo', 'Get LFW from which file?');
        path=[filepath];
        name=[filename];
        fid=fopen(strcat(path,name), 'r');
        lfwtype=fscanf(fid, '%s\n%');
        rsst=fscanf(fid, '%s\n%');
        wall=fscanf(fid, '%s\n%');
        rate=fscanf(fid, '%s\n%');
        amp=fscanf(fid, '%s\n%');
        phase=fscanf(fid, '%s\n%');
        offset=fscanf(fid, '%s\n%');
        damping=fscanf(fid, '%s\n%');
        rectif=fscanf(fid, '%s\n%');
        status=fclose(fid);
        h=waitbar(0, 'Gathering LFO Parameters. Please Wait...');
        x=data.y;
        n=1:length(x)+md;
    am3=str2num(amp);
    damp=str2num(damping);
        r=str2num(rate);
        rinv=1/r;
        ww=2*pi/round(rinv*data.samplerate);
        p=str2num(phase);
        static=findobj(gcf, 'Tag', 'StaticText23');
        set(static, 'String', rate);
        static1=findobj(gcf, 'Tag', 'StaticText33');
        set(static1, 'String', amp);
        if strcmp(lfwtype, 'sin')
            stati=findobj(gcf, 'Tag', 'StaticText13');
            set(stati, 'String', 'Sine');
            if strcmp(rsst, '1')
                g=(1-exp(-(n*damp)).*sin(ww.*n+p));
                a=int32(g);
            end
        end
    end
end

```

```

elseif strcmp(rsst, '0')
    a=(1-exp(-(n*damp)).*sin(ww.*n+p));
end
if strcmp(wall, '1')
    b=a+0.1*sin(2*pi*50*t);
elseif strcmp(wall, '0')
    b=a;
end
if strcmp(rectif, '1')
    lfw3=double(abs(b));
elseif strcmp(rectif, '0')
    lfw3=double(b);
end
elseif strcmp(lfwtype, 'cos')
    stati=findobj(gcf, 'Tag', 'StaticText13');
    set(stati, 'String', 'Cosine');
    if strcmp(rsst, '1')
        g=(1-exp(-(n*damp)).*cos(ww.*n+p));
        a=int32(g);
    elseif strcmp(rsst, '0')
        a=(1-exp(-(n*damp)).*cos(ww.*n+p));
    end
    if strcmp(wall, '1')
        b=a+0.1*sin(2*pi*50*t);
    elseif strcmp(wall, '0')
        b=a;
    end
    if strcmp(rectif, '1')
        lfw3=double(abs(b));
    elseif strcmp(rectif, '0')
        lfw3=double(b);
    end
elseif strcmp(lfwtype, 'tri')
    stati=findobj(gcf, 'Tag', 'StaticText13');
    set(stati, 'String', 'Triangular');
    if strcmp(rsst, '1')
        g=(1-exp(-(n*damp)).*sawtooth(ww.*n, p+0.5));
        a=int32(g);
    elseif strcmp(rsst, '0')
        a=(1-exp(-(n*damp)).*sawtooth(ww.*n, p+0.5));
    end
    if strcmp(wall, '1')
        b=a+0.1*sin(2*pi*50*t);
    elseif strcmp(wall, '0')
        b=a;
    end
    if strcmp(rectif, '1')

```

```

        lfw3=double(abs(b));
elseif strcmp(rectif,'0')
    lfw3=double(b);
end
elseif strcmp(lfwtype,'squ')
    stati=findobj(gcf,'Tag','StaticText13');
    set(stati,'String','Square');
    if strcmp(rsst,'1')
        g=(1-exp(-(n*damp)).*square(ww.*n,p));
        a=int32(g);
elseif strcmp(rsst,'0')
    a=(1-exp(-(n*damp)).*square(ww.*n,p));
end
if strcmp(wall,'1')
    b=a+0.1*sin(2*pi*50*t);
elseif strcmp(wall,'0')
    b=a;
end
if strcmp(rectif,'1')
    lfw3=double(abs(b));
elseif strcmp(rectif,'0')
    lfw3=double(b);
end
elseif strcmp(lfwtype,'exp')
    h1=warndlg('Sorry but this is not a type of LFW usable with this kind of
algorithm','Warning!!!');
    uiwait(h1);
elseif strcmp(lfwtype,'saw')
    stati=findobj(gcf,'Tag','StaticText13');
    set(stati,'String','Sawtooth');
    if strcmp(rsst,'1')
        g=(1-exp(-(n*damp)).*sawtooth(ww.*n,p));
        a=int32(g);
elseif strcmp(rsst,'0')
    a=(1-exp(-(n*damp)).*sawtooth(ww.*n,p));
end
if strcmp(wall,'1')
    b=a+0.1*sin(2*pi*50*t);
elseif strcmp(wall,'0')
    b=a;
end
if strcmp(rectif,'1')
    lfw3=double(abs(b));
elseif strcmp(rectif,'0')
    lfw3=double(b);
end
elseif strcmp(lfwtype,'ran')

```

```

stati=findobj(gcf,'Tag','StaticText13');
set(stati,'String','Pseudo Random');
if strcmp(rsst,'1')
    g=(1-exp(-(n*damp)).*sin(rand(1)*ww.*n+p));
    a=int32(g);
elseif strcmp(rsst,'0')
    a=(1-exp(-(n*damp)).*sin(rand(1)*ww.*n+p));
end
if strcmp(wall,'1')
    b=a+0.1*sin(2*pi*50*t);
elseif strcmp(wall,'0')
    b=a;
end
if strcmp(rectif,'1')
    lfw3=double(abs(b));
elseif strcmp(rectif,'0')
    lfw3=double(b);
end
end
for i=1:100,
    waitbar(i/100);
end
close(h);
elseif popup3==2
    lfomaker;
elseif popup3==3
    static=findobj(gcf,'Tag','StaticText23');
    set(static,'String','Smart');
    static1=findobj(gcf,'Tag','StaticText33');
    set(static1,'String','Smart');
    stati=findobj(gcf,'Tag','StaticText13');
    set(stati,'String','Smart');
    if data.channel==1
        len=data.count;
    elseif data.channel==2
        len=(data.count)/2;
    end
    assignin('base','maxdelays',md);
    assignin('base','lengthfile',len);
    assignin('base','samp',data.samplerate);
    h=waitbar(0,'Passing data to SmartPlayer System');
    save('xchangesmart');
    for i=1:100,
        waitbar(i/100);
    end
    close(h);
    smartplayer;

```

```

h3=warndlg('Please maintain this Dialog Opened (dont push OK) until you have
finished to use the SmartPlayer System (he has the control of the entire system
now!!!)', 'WARNING!!!');
    uiwait(h3);
    passed=load('xchangesmart.mat');
    lfw3=passed.lfw;
    lfw3=abs(lfw3)';
end
case 'getvoice4'
    popup4handle=findobj(gcf, 'Tag', 'PopupMenu4');
    popup4=get(popup4handle, 'Value');
    mdh=findobj(gcf, 'Tag', 'EditText20');
    md=ceil((str2num(get(mdh, 'String'))/1000)*data.samplerate);
    if popup4==1
        [filename,filepath]=uigetfile('*.lfo', 'Get LFW from which file?');
        path=[filepath];
        name=[filename];
        fid=fopen(strcat(path,name), 'r');
        lfwtype=fscanf(fid, '%s\n%');
        rsst=fscanf(fid, '%s\n%');
        wall=fscanf(fid, '%s\n%');
        rate=fscanf(fid, '%s\n%');
        amp=fscanf(fid, '%s\n%');
        phase=fscanf(fid, '%s\n%');
        offset=fscanf(fid, '%s\n%');
        damping=fscanf(fid, '%s\n%');
        rectif=fscanf(fid, '%s\n%');
        status=fclose(fid);
        h=waitbar(0, 'Gathering LFO Parameters. Please Wait...');
        x=data.y;
        n=1:length(x)+md;
    am4=str2num(amp);
    damp=str2num(damping);
        r=str2num(rate);
        rinv=1/r;
        ww=2*pi/round(rinv*data.samplerate);
        p=str2num(phase);
        static=findobj(gcf, 'Tag', 'StaticText24');
        set(static, 'String', rate);
        static1=findobj(gcf, 'Tag', 'StaticText34');
        set(static1, 'String', amp);
        if strcmp(lfwtype, 'sin')
            stati=findobj(gcf, 'Tag', 'StaticText14');
            set(stati, 'String', 'Sine');
            if strcmp(rsst, '1')
                g=(1-exp(-(n*damp)).*sin(ww.*n+p));
                a=int32(g);
            end
        end
    end
end

```

```

elseif strcmp(rsst, '0')
    a=(1-exp(-(n*damp)).*sin(ww.*n+p));
end
if strcmp(wall, '1')
    b=a+0.1*sin(2*pi*50*t);
elseif strcmp(wall, '0')
    b=a;
end
if strcmp(rectif, '1')
    lfw4=double(abs(b));
elseif strcmp(rectif, '0')
    lfw4=double(b);
end
elseif strcmp(lfwtype, 'cos')
    stati=findobj(gcf, 'Tag', 'StaticText14');
    set(stati, 'String', 'Cosine');
    if strcmp(rsst, '1')
        g=(1-exp(-(n*damp)).*cos(ww.*n+p));
        a=int32(g);
    elseif strcmp(rsst, '0')
        a=(1-exp(-(n*damp)).*cos(ww.*n+p));
    end
    if strcmp(wall, '1')
        b=a+0.1*sin(2*pi*50*t);
    elseif strcmp(wall, '0')
        b=a;
    end
    if strcmp(rectif, '1')
        lfw4=double(abs(b));
    elseif strcmp(rectif, '0')
        lfw4=double(b);
    end
elseif strcmp(lfwtype, 'tri')
    stati=findobj(gcf, 'Tag', 'StaticText14');
    set(stati, 'String', 'Triangular');
    if strcmp(rsst, '1')
        g=(1-exp(-(n*damp)).*sawtooth(ww.*n, p+0.5));
        a=int32(g);
    elseif strcmp(rsst, '0')
        a=(1-exp(-(n*damp)).*sawtooth(ww.*n, p+0.5));
    end
    if strcmp(wall, '1')
        b=a+0.1*sin(2*pi*50*t);
    elseif strcmp(wall, '0')
        b=a;
    end
    if strcmp(rectif, '1')

```

```

        lfw4=double(abs(b));
elseif strcmp(rectif,'0')
    lfw4=double(b);
end
elseif strcmp(lfwtype,'squ')
    stati=findobj(gcf,'Tag','StaticText14');
    set(stati,'String','Square');
    if strcmp(rsst,'1')
        g=(1-exp(-(n*damp)).*square(ww.*n,p));
        a=int32(g);
elseif strcmp(rsst,'0')
    a=(1-exp(-(n*damp)).*square(ww.*n,p));
end
if strcmp(wall,'1')
    b=a+0.1*sin(2*pi*50*t);
elseif strcmp(wall,'0')
    b=a;
end
if strcmp(rectif,'1')
    lfw4=double(abs(b));
elseif strcmp(rectif,'0')
    lfw4=double(b);
end
elseif strcmp(lfwtype,'exp')
    h1=warndlg('Sorry but this is not a type of LFW usable with this kind of
algorithm','Warning!!!');
    uiwait(h1);
elseif strcmp(lfwtype,'saw')
    stati=findobj(gcf,'Tag','StaticText14');
    set(stati,'String','Sawtooth');
    if strcmp(rsst,'1')
        g=(1-exp(-(n*damp)).*sawtooth(ww.*n,p));
        a=int32(g);
elseif strcmp(rsst,'0')
    a=(1-exp(-(n*damp)).*sawtooth(ww.*n,p));
end
if strcmp(wall,'1')
    b=a+0.1*sin(2*pi*50*t);
elseif strcmp(wall,'0')
    b=a;
end
if strcmp(rectif,'1')
    lfw4=double(abs(b));
elseif strcmp(rectif,'0')
    lfw4=double(b);
end
elseif strcmp(lfwtype,'ran')

```

```

stati=findobj(gcf,'Tag','StaticText14');
set(stati,'String','Pseudo Random');
if strcmp(rsst,'1')
    g=(1-exp(-(n*damp)).*sin(rand(1)*ww.*n+p));
    a=int32(g);
elseif strcmp(rsst,'0')
    a=(1-exp(-(n*damp)).*sin(rand(1)*ww.*n+p));
end
if strcmp(wall,'1')
    b=a+0.1*sin(2*pi*50*t);
elseif strcmp(wall,'0')
    b=a;
end
if strcmp(rectif,'1')
    lfw4=double(abs(b));
elseif strcmp(rectif,'0')
    lfw4=double(b);
end
end
for i=1:100,
    waitbar(i/100);
end
close(h);
elseif popup4==2
    lfomaker;
elseif popup4==3
    static=findobj(gcf,'Tag','StaticText24');
    set(static,'String','Smart');
    static1=findobj(gcf,'Tag','StaticText34');
    set(static1,'String','Smart');
    stati=findobj(gcf,'Tag','StaticText14');
    set(stati,'String','Smart');
    if data.channel==1
        len=data.count;
    elseif data.channel==2
        len=(data.count)/2;
    end
    assignin('base','maxdelays',md);
    assignin('base','lengthfile',len);
    assignin('base','samp',data.samplerate);
    h=waitbar(0,'Passing data to SmartPlayer System');
    save('xchangesmart');
    for i=1:100,
        waitbar(i/100);
    end
    close(h);
    smartplayer;

```

```

h3=warndlg('Please maintain this Dialog Opened (dont push OK) until you have
finished to use the SmartPlayer System (he has the control of the entire system
now!!!)', 'WARNING!!!');
    uiwait(h3);
    passed=load('xchangesmart.mat');
    lfw4=passed.lfw;
    lfw4=abs(lfw4)';
end
case 'getvoice5'
    popup5handle=findobj(gcf,'Tag','PopupMenu5');
    popup5=get(popup5handle,'Value');
    mdh=findobj(gcf,'Tag','EditText20');
    md=ceil((str2num(get(mdh,'String'))/1000)*data.samplerate);
    if popup5==1
        [filename,filepath]=uigetfile('*.lfo','Get LFW from which file?');
        path=[filepath];
        name=[filename];
        fid=fopen(strcat(path,name),'r');
        lfwtype=fscanf(fid,'%s\n%');
        rsst=fscanf(fid,'%s\n%');
        wall=fscanf(fid,'%s\n%');
        rate=fscanf(fid,'%s\n%');
        amp=fscanf(fid,'%s\n%');
        phase=fscanf(fid,'%s\n%');
        offset=fscanf(fid,'%s\n%');
        damping=fscanf(fid,'%s\n%');
        rectif=fscanf(fid,'%s\n%');
        status=fclose(fid);
        h=waitbar(0,'Gathering LFO Parameters. Please Wait...');
        x=data.y;
        n=1:length(x)+md;
    am5=str2num(amp);
    damp=str2num(damping);
        r=str2num(rate);
        rinv=1/r;
        ww=2*pi/round(rinv*data.samplerate);
        p=str2num(phase);
        static=findobj(gcf,'Tag','StaticText25');
        set(static,'String',rate);
        static1=findobj(gcf,'Tag','StaticText35');
        set(static1,'String',amp);
        if strcmp(lfwtype,'sin')
            stati=findobj(gcf,'Tag','StaticText15');
            set(stati,'String','Sine');
            if strcmp(rsst,'1')
                g=(1-exp(-(n*damp)).*sin(ww.*n+p));
                a=int32(g);
            end
        end
    end
end

```

```

elseif strcmp(rsst, '0')
    a=(1-exp(-(n*damp)).*sin(ww.*n+p));
end
if strcmp(wall, '1')
    b=a+0.1*sin(2*pi*50*t);
elseif strcmp(wall, '0')
    b=a;
end
if strcmp(rectif, '1')
    lfw5=double(abs(b));
elseif strcmp(rectif, '0')
    lfw5=double(b);
end
elseif strcmp(lfwtype, 'cos')
    stati=findobj(gcf, 'Tag', 'StaticText15');
    set(stati, 'String', 'Cosine');
    if strcmp(rsst, '1')
        g=(1-exp(-(n*damp)).*cos(ww.*n+p));
        a=int32(g);
    elseif strcmp(rsst, '0')
        a=(1-exp(-(n*damp)).*cos(ww.*n+p));
    end
    if strcmp(wall, '1')
        b=a+0.1*sin(2*pi*50*t);
    elseif strcmp(wall, '0')
        b=a;
    end
    if strcmp(rectif, '1')
        lfw5=double(abs(b));
    elseif strcmp(rectif, '0')
        lfw5=double(b);
    end
elseif strcmp(lfwtype, 'tri')
    stati=findobj(gcf, 'Tag', 'StaticText15');
    set(stati, 'String', 'Triangular');
    if strcmp(rsst, '1')
        g=(1-exp(-(n*damp)).*sawtooth(ww.*n, p+0.5));
        a=int32(g);
    elseif strcmp(rsst, '0')
        a=(1-exp(-(n*damp)).*sawtooth(ww.*n, p+0.5));
    end
    if strcmp(wall, '1')
        b=a+0.1*sin(2*pi*50*t);
    elseif strcmp(wall, '0')
        b=a;
    end
    if strcmp(rectif, '1')

```

```

        lfw5=double(abs(b));
elseif strcmp(rectif,'0')
    lfw5=double(b);
end
elseif strcmp(lfwtype,'squ')
    stati=findobj(gcf,'Tag','StaticText15');
    set(stati,'String','Square');
    if strcmp(rsst,'1')
        g=(1-exp(-(n*damp)).*square(ww.*n,p));
        a=int32(g);
elseif strcmp(rsst,'0')
    a=(1-exp(-(n*damp)).*square(ww.*n,p));
end
if strcmp(wall,'1')
    b=a+0.1*sin(2*pi*50*t);
elseif strcmp(wall,'0')
    b=a;
end
if strcmp(rectif,'1')
    lfw5=double(abs(b));
elseif strcmp(rectif,'0')
    lfw5=double(b);
end
elseif strcmp(lfwtype,'exp')
    h1=warndlg('Sorry but this is not a type of LFW usable with this kind of
algorithm','Warning!!!');
    uiwait(h1);
elseif strcmp(lfwtype,'saw')
    stati=findobj(gcf,'Tag','StaticText15');
    set(stati,'String','Sawtooth');
    if strcmp(rsst,'1')
        g=(1-exp(-(n*damp)).*sawtooth(ww.*n,p));
        a=int32(g);
elseif strcmp(rsst,'0')
    a=(1-exp(-(n*damp)).*sawtooth(ww.*n,p));
end
if strcmp(wall,'1')
    b=a+0.1*sin(2*pi*50*t);
elseif strcmp(wall,'0')
    b=a;
end
if strcmp(rectif,'1')
    lfw5=double(abs(b));
elseif strcmp(rectif,'0')
    lfw5=double(b);
end
elseif strcmp(lfwtype,'ran')

```

```

stati=findobj(gcf,'Tag','StaticText15');
set(stati,'String','Pseudo Random');
if strcmp(rsst,'1')
    g=(1-exp(-(n*damp)).*sin(rand(1)*ww.*n+p));
    a=int32(g);
elseif strcmp(rsst,'0')
    a=(1-exp(-(n*damp)).*sin(rand(1)*ww.*n+p));
end
if strcmp(wall,'1')
    b=a+0.1*sin(2*pi*50*t);
elseif strcmp(wall,'0')
    b=a;
end
if strcmp(rectif,'1')
    lfw5=double(abs(b));
elseif strcmp(rectif,'0')
    lfw5=double(b);
end
end
for i=1:100,
    waitbar(i/100);
end
close(h);
elseif popup5==2
    lfomaker;
elseif popup5==3
    static=findobj(gcf,'Tag','StaticText25');
    set(static,'String','Smart');
    static1=findobj(gcf,'Tag','StaticText35');
    set(static1,'String','Smart');
    stati=findobj(gcf,'Tag','StaticText15');
    set(stati,'String','Smart');
    if data.channel==1
        len=data.count;
    elseif data.channel==2
        len=(data.count)/2;
    end
    assignin('base','maxdelays',md);
    assignin('base','lengthfile',len);
    assignin('base','samp',data.samplerate);
    h=waitbar(0,'Passing data to SmartPlayer System');
    save('xchangesmart');
    for i=1:100,
        waitbar(i/100);
    end
    close(h);
    smartplayer;

```

```

    h3=warndlg('Please maintain this Dialog Opened (dont push OK) until you have
finished to use the SmartPlayer System (he has the control of the entire system
now!!!)', 'WARNING!!!');
    uiwait(h3);
    passed=load('xchangesmart.mat');
    lfw5=passed.lfw;
    lfw5=abs(lfw5)';
end
case 'getvoice6'
    popup6handle=findobj(gcf, 'Tag', 'PopupMenu6');
    popup6=get(popup6handle, 'Value');
    mdh=findobj(gcf, 'Tag', 'EditText20');
    md=ceil((str2num(get(mdh, 'String'))/1000)*data.samplerate);
    if popup6==1
        [filename, filepath]=uigetfile('*.lfo', 'Get LFW from which file?');
        path=[filepath];
        name=[filename];
        fid=fopen(strcat(path, name), 'r');
        lfwtype=fscanf(fid, '%s\n%');
        rsst=fscanf(fid, '%s\n%');
        wall=fscanf(fid, '%s\n%');
        rate=fscanf(fid, '%s\n%');
        amp=fscanf(fid, '%s\n%');
        phase=fscanf(fid, '%s\n%');
        offset=fscanf(fid, '%s\n%');
        damping=fscanf(fid, '%s\n%');
        rectif=fscanf(fid, '%s\n%');
        status=fclose(fid);
        h=waitbar(0, 'Gathering LFO Parameters. Please Wait...');
        x=data.y;
        n=1:length(x)+md;
    am6=str2num(amp);
    damp=str2num(damping);
    r=str2num(rate);
    rinv=1/r;
    ww=2*pi/round(rinv*data.samplerate);
    p=str2num(phase);
    static=findobj(gcf, 'Tag', 'StaticText26');
    set(static, 'String', rate);
    static1=findobj(gcf, 'Tag', 'StaticText36');
    set(static1, 'String', amp);
    if strcmp(lfwtype, 'sin')
        stati=findobj(gcf, 'Tag', 'StaticText16');
        set(stati, 'String', 'Sine');
        if strcmp(rsst, '1')
            g=(1-exp(-(n*damp)).*sin(ww.*n+p));
            a=int32(g);

```

```

elseif strcmp(rsst, '0')
    a=(1-exp(-(n*damp)).*sin(ww.*n+p));
end
if strcmp(wall, '1')
    b=a+0.1*sin(2*pi*50*t);
elseif strcmp(wall, '0')
    b=a;
end
if strcmp(rectif, '1')
    lfw6=double(abs(b));
elseif strcmp(rectif, '0')
    lfw6=double(b);
end
elseif strcmp(lfwtype, 'cos')
    stati=findobj(gcf, 'Tag', 'StaticText16');
    set(stati, 'String', 'Cosine');
    if strcmp(rsst, '1')
        g=(1-exp(-(n*damp)).*cos(ww.*n+p));
        a=int32(g);
    elseif strcmp(rsst, '0')
        a=(1-exp(-(n*damp)).*cos(ww.*n+p));
    end
    if strcmp(wall, '1')
        b=a+0.1*sin(2*pi*50*t);
    elseif strcmp(wall, '0')
        b=a;
    end
    if strcmp(rectif, '1')
        lfw6=double(abs(b));
    elseif strcmp(rectif, '0')
        lfw6=double(b);
    end
elseif strcmp(lfwtype, 'tri')
    stati=findobj(gcf, 'Tag', 'StaticText16');
    set(stati, 'String', 'Triangular');
    if strcmp(rsst, '1')
        g=(1-exp(-(n*damp)).*sawtooth(ww.*n, p+0.5));
        a=int32(g);
    elseif strcmp(rsst, '0')
        a=(1-exp(-(n*damp)).*sawtooth(ww.*n, p+0.5));
    end
    if strcmp(wall, '1')
        b=a+0.1*sin(2*pi*50*t);
    elseif strcmp(wall, '0')
        b=a;
    end
    if strcmp(rectif, '1')

```

```

        lfw6=double(abs(b));
elseif strcmp(rectif,'0')
    lfw6=double(b);
end
elseif strcmp(lfwtype,'squ')
    stati=findobj(gcf,'Tag','StaticText16');
    set(stati,'String','Square');
    if strcmp(rsst,'1')
        g=(1-exp(-(n*damp)).*square(ww.*n,p));
        a=int32(g);
elseif strcmp(rsst,'0')
    a=(1-exp(-(n*damp)).*square(ww.*n,p));
end
if strcmp(wall,'1')
    b=a+0.1*sin(2*pi*50*t);
elseif strcmp(wall,'0')
    b=a;
end
if strcmp(rectif,'1')
    lfw6=double(abs(b));
elseif strcmp(rectif,'0')
    lfw6=double(b);
end
elseif strcmp(lfwtype,'exp')
    h1=warndlg('Sorry but this is not a type of LFW usable with this kind of
algorithm','Warning!!!');
    uiwait(h1);
elseif strcmp(lfwtype,'saw')
    stati=findobj(gcf,'Tag','StaticText16');
    set(stati,'String','Sawtooth');
    if strcmp(rsst,'1')
        g=(1-exp(-(n*damp)).*sawtooth(ww.*n,p));
        a=int32(g);
elseif strcmp(rsst,'0')
    a=(1-exp(-(n*damp)).*sawtooth(ww.*n,p));
end
if strcmp(wall,'1')
    b=a+0.1*sin(2*pi*50*t);
elseif strcmp(wall,'0')
    b=a;
end
if strcmp(rectif,'1')
    lfw6=double(abs(b));
elseif strcmp(rectif,'0')
    lfw6=double(b);
end
elseif strcmp(lfwtype,'ran')

```

```

stati=findobj(gcf,'Tag','StaticText16');
set(stati,'String','Pseudo Random');
if strcmp(rsst,'1')
    g=(1-exp(-(n*damp)).*sin(rand(1)*ww.*n+p));
    a=int32(g);
elseif strcmp(rsst,'0')
    a=(1-exp(-(n*damp)).*sin(rand(1)*ww.*n+p));
end
if strcmp(wall,'1')
    b=a+0.1*sin(2*pi*50*t);
elseif strcmp(wall,'0')
    b=a;
end
if strcmp(rectif,'1')
    lfw6=double(abs(b));
elseif strcmp(rectif,'0')
    lfw6=double(b);
end
end
for i=1:100,
    waitbar(i/100);
end
close(h);
elseif popup6==2
    lfomaker;
elseif popup6==3
    static=findobj(gcf,'Tag','StaticText26');
    set(static,'String','Smart');
    static1=findobj(gcf,'Tag','StaticText36');
    set(static1,'String','Smart');
    stati=findobj(gcf,'Tag','StaticText16');
    set(stati,'String','Smart');
    if data.channel==1
        len=data.count;
    elseif data.channel==2
        len=(data.count)/2;
    end
    assignin('base','maxdelays',md);
    assignin('base','lengthfile',len);
    assignin('base','samp',data.samplerate);
    h=waitbar(0,'Passing data to SmartPlayer System');
    save('xchangesmart');
    for i=1:100,
        waitbar(i/100);
    end
    close(h);
    smartplayer;

```

```

h3=warndlg('Please maintain this Dialog Opened (dont push OK) until you have
finished to use the SmartPlayer System (he has the control of the entire system
now!!!)', 'WARNING!!!');
    uiwait(h3);
    passed=load('xchangesmart.mat');
    lfw6=passed.lfw;
    lfw6=abs(lfw6)';
end
case 'getvoice7'
    popup7handle=findobj(gcf,'Tag','PopupMenu7');
    popup7=get(popup7handle,'Value');
    mdh=findobj(gcf,'Tag','EditText20');
    md=ceil((str2num(get(mdh,'String'))/1000)*data.samplerate);
    if popup7==1
        [filename,filepath]=uigetfile('*.lfo','Get LFW from which file?');
        path=[filepath];
        name=[filename];
        fid=fopen(strcat(path,name),'r');
        lfwtype=fscanf(fid,'%s\n%');
        rsst=fscanf(fid,'%s\n%');
        wall=fscanf(fid,'%s\n%');
        rate=fscanf(fid,'%s\n%');
        amp=fscanf(fid,'%s\n%');
        phase=fscanf(fid,'%s\n%');
        offset=fscanf(fid,'%s\n%');
        damping=fscanf(fid,'%s\n%');
        rectif=fscanf(fid,'%s\n%');
        status=fclose(fid);
        h=waitbar(0,'Gathering LFO Parameters. Please Wait...');
        x=data.y;
        n=1:length(x)+md;
    am7=str2num(amp);
    damp=str2num(damping);
        r=str2num(rate);
        rinv=1/r;
        ww=2*pi/round(rinv*data.samplerate);
        p=str2num(phase);
        static=findobj(gcf,'Tag','StaticText27');
        set(static,'String',rate);
        static1=findobj(gcf,'Tag','StaticText37');
        set(static1,'String',amp);
        if strcmp(lfwtype,'sin')
            stati=findobj(gcf,'Tag','StaticText17');
            set(stati,'String','Sine');
            if strcmp(rsst,'1')
                g=(1-exp(-(n*damp)).*sin(ww.*n+p));
                a=int32(g);
            end
        end
    end
end

```

```

elseif strcmp(rsst, '0')
    a=(1-exp(-(n*damp)).*sin(ww.*n+p));
end
if strcmp(wall, '1')
    b=a+0.1*sin(2*pi*50*t);
elseif strcmp(wall, '0')
    b=a;
end
if strcmp(rectif, '1')
    lfw7=double(abs(b));
elseif strcmp(rectif, '0')
    lfw7=double(b);
end
elseif strcmp(lfwtype, 'cos')
    stati=findobj(gcf, 'Tag', 'StaticText17');
    set(stati, 'String', 'Cosine');
    if strcmp(rsst, '1')
        g=(1-exp(-(n*damp)).*cos(ww.*n+p));
        a=int32(g);
    elseif strcmp(rsst, '0')
        a=(1-exp(-(n*damp)).*cos(ww.*n+p));
    end
    if strcmp(wall, '1')
        b=a+0.1*sin(2*pi*50*t);
    elseif strcmp(wall, '0')
        b=a;
    end
    if strcmp(rectif, '1')
        lfw7=double(abs(b));
    elseif strcmp(rectif, '0')
        lfw7=double(b);
    end
elseif strcmp(lfwtype, 'tri')
    stati=findobj(gcf, 'Tag', 'StaticText17');
    set(stati, 'String', 'Triangular');
    if strcmp(rsst, '1')
        g=(1-exp(-(n*damp)).*sawtooth(ww.*n, p+0.5));
        a=int32(g);
    elseif strcmp(rsst, '0')
        a=(1-exp(-(n*damp)).*sawtooth(ww.*n, p+0.5));
    end
    if strcmp(wall, '1')
        b=a+0.1*sin(2*pi*50*t);
    elseif strcmp(wall, '0')
        b=a;
    end
    if strcmp(rectif, '1')

```

```

        lfw7=double(abs(b));
elseif strcmp(rectif,'0')
    lfw7=double(b);
end
elseif strcmp(lfwtype,'squ')
    stati=findobj(gcf,'Tag','StaticText17');
    set(stati,'String','Square');
    if strcmp(rsst,'1')
        g=(1-exp(-(n*damp)).*square(ww.*n,p));
        a=int32(g);
elseif strcmp(rsst,'0')
    a=(1-exp(-(n*damp)).*square(ww.*n,p));
end
if strcmp(wall,'1')
    b=a+0.1*sin(2*pi*50*t);
elseif strcmp(wall,'0')
    b=a;
end
if strcmp(rectif,'1')
    lfw7=double(abs(b));
elseif strcmp(rectif,'0')
    lfw7=double(b);
end
elseif strcmp(lfwtype,'exp')
    h1=warndlg('Sorry but this is not a type of LFW usable with this kind of
algorithm','Warning!!!');
    uiwait(h1);
elseif strcmp(lfwtype,'saw')
    stati=findobj(gcf,'Tag','StaticText17');
    set(stati,'String','Sawtooth');
    if strcmp(rsst,'1')
        g=(1-exp(-(n*damp)).*sawtooth(ww.*n,p));
        a=int32(g);
elseif strcmp(rsst,'0')
    a=(1-exp(-(n*damp)).*sawtooth(ww.*n,p));
end
if strcmp(wall,'1')
    b=a+0.1*sin(2*pi*50*t);
elseif strcmp(wall,'0')
    b=a;
end
if strcmp(rectif,'1')
    lfw7=double(abs(b));
elseif strcmp(rectif,'0')
    lfw7=double(b);
end
elseif strcmp(lfwtype,'ran')

```

```

    stati=findobj(gcf,'Tag','StaticText17');
    set(stati,'String','Pseudo Random');
    if strcmp(rsst,'1')
        g=(1-exp(-(n*damp)).*sin(rand(1)*ww.*n+p));
        a=int32(g);
    elseif strcmp(rsst,'0')
        a=(1-exp(-(n*damp)).*sin(rand(1)*ww.*n+p));
    end
    if strcmp(wall,'1')
        b=a+0.1*sin(2*pi*50*t);
    elseif strcmp(wall,'0')
        b=a;
    end
    if strcmp(rectif,'1')
        lfw7=double(abs(b));
    elseif strcmp(rectif,'0')
        lfw7=double(b);
    end
end
for i=1:100,
    waitbar(i/100);
end
close(h);
elseif popup7==2
    lfomaker;
elseif popup7==3
    static=findobj(gcf,'Tag','StaticText27');
    set(static,'String','Smart');
    static1=findobj(gcf,'Tag','StaticText37');
    set(static1,'String','Smart');
    stati=findobj(gcf,'Tag','StaticText17');
    set(stati,'String','Smart');
    if data.channel==1
        len=data.count;
    elseif data.channel==2
        len=(data.count)/2;
    end
    assignin('base','maxdelays',md);
    assignin('base','lengthfile',len);
    assignin('base','samp',data.samplerate);
    h=waitbar(0,'Passing data to SmartPlayer System');
    save('xchangesmart');
    for i=1:100,
        waitbar(i/100);
    end
    close(h);
    smartplayer;

```

```

    h3=warndlg('Please maintain this Dialog Opened (dont push OK) until you have
finished to use the SmartPlayer System (he has the control of the entire system
now!!!)', 'WARNING!!!');
    uiwait(h3);
    passed=load('xchangesmart.mat');
    lfw7=passed.lfw;
    lfw7=abs(lfw7)';
end
case 'getvoice8'
    popup8handle=findobj(gcf, 'Tag', 'PopupMenu8');
    popup8=get(popup8handle, 'Value');
    mdh=findobj(gcf, 'Tag', 'EditText20');
    md=ceil((str2num(get(mdh, 'String'))/1000)*data.samplerate);
    if popup8==1
        [filename, filepath]=uigetfile('*.lfo', 'Get LFW from which file?');
        path=[filepath];
        name=[filename];
        fid=fopen(strcat(path, name), 'r');
        lfwtype=fscanf(fid, '%s\n%');
        rsst=fscanf(fid, '%s\n%');
        wall=fscanf(fid, '%s\n%');
        rate=fscanf(fid, '%s\n%');
        amp=fscanf(fid, '%s\n%');
        phase=fscanf(fid, '%s\n%');
        offset=fscanf(fid, '%s\n%');
        damping=fscanf(fid, '%s\n%');
        rectif=fscanf(fid, '%s\n%');
        status=fclose(fid);
        h=waitbar(0, 'Gathering LFO Parameters. Please Wait...');
        x=data.y;
        n=1:length(x)+md;
    am8=str2num(amp);
    damp=str2num(damping);
    r=str2num(rate);
    rinv=1/r;
    ww=2*pi/round(rinv*data.samplerate);
    p=str2num(phase);
    static=findobj(gcf, 'Tag', 'StaticText28');
    set(static, 'String', rate);
    static1=findobj(gcf, 'Tag', 'StaticText38');
    set(static1, 'String', amp);
    if strcmp(lfwtype, 'sin')
        stati=findobj(gcf, 'Tag', 'StaticText18');
        set(stati, 'String', 'Sine');
        if strcmp(rsst, '1')
            g=(1-exp(-(n*damp)).*sin(ww.*n+p));
            a=int32(g);

```

```

elseif strcmp(rsst, '0')
    a=(1-exp(-(n*damp)).*sin(ww.*n+p));
end
if strcmp(wall, '1')
    b=a+0.1*sin(2*pi*50*t);
elseif strcmp(wall, '0')
    b=a;
end
if strcmp(rectif, '1')
    lfw8=double(abs(b));
elseif strcmp(rectif, '0')
    lfw8=double(b);
end
elseif strcmp(lfwtype, 'cos')
    stati=findobj(gcf, 'Tag', 'StaticText18');
    set(stati, 'String', 'Cosine');
    if strcmp(rsst, '1')
        g=(1-exp(-(n*damp)).*cos(ww.*n+p));
        a=int32(g);
    elseif strcmp(rsst, '0')
        a=(1-exp(-(n*damp)).*cos(ww.*n+p));
    end
    if strcmp(wall, '1')
        b=a+0.1*sin(2*pi*50*t);
    elseif strcmp(wall, '0')
        b=a;
    end
    if strcmp(rectif, '1')
        lfw8=double(abs(b));
    elseif strcmp(rectif, '0')
        lfw8=double(b);
    end
elseif strcmp(lfwtype, 'tri')
    stati=findobj(gcf, 'Tag', 'StaticText18');
    set(stati, 'String', 'Triangular');
    if strcmp(rsst, '1')
        g=(1-exp(-(n*damp)).*sawtooth(ww.*n, p+0.5));
        a=int32(g);
    elseif strcmp(rsst, '0')
        a=(1-exp(-(n*damp)).*sawtooth(ww.*n, p+0.5));
    end
    if strcmp(wall, '1')
        b=a+0.1*sin(2*pi*50*t);
    elseif strcmp(wall, '0')
        b=a;
    end
    if strcmp(rectif, '1')

```

```

        lfw8=double(abs(b));
elseif strcmp(rectif,'0')
    lfw8=double(b);
end
elseif strcmp(lfwtype,'squ')
    stati=findobj(gcf,'Tag','StaticText18');
    set(stati,'String','Square');
    if strcmp(rsst,'1')
        g=(1-exp(-(n*damp)).*square(ww.*n,p));
        a=int32(g);
elseif strcmp(rsst,'0')
    a=(1-exp(-(n*damp)).*square(ww.*n,p));
end
if strcmp(wall,'1')
    b=a+0.1*sin(2*pi*50*t);
elseif strcmp(wall,'0')
    b=a;
end
if strcmp(rectif,'1')
    lfw8=double(abs(b));
elseif strcmp(rectif,'0')
    lfw8=double(b);
end
elseif strcmp(lfwtype,'exp')
    h1=warndlg('Sorry but this is not a type of LFW usable with this kind of
algorithm','Warning!!!');
    uiwait(h1);
elseif strcmp(lfwtype,'saw')
    stati=findobj(gcf,'Tag','StaticText18');
    set(stati,'String','Sawtooth');
    if strcmp(rsst,'1')
        g=(1-exp(-(n*damp)).*sawtooth(ww.*n,p));
        a=int32(g);
elseif strcmp(rsst,'0')
    a=(1-exp(-(n*damp)).*sawtooth(ww.*n,p));
end
if strcmp(wall,'1')
    b=a+0.1*sin(2*pi*50*t);
elseif strcmp(wall,'0')
    b=a;
end
if strcmp(rectif,'1')
    lfw8=double(abs(b));
elseif strcmp(rectif,'0')
    lfw8=double(b);
end
elseif strcmp(lfwtype,'ran')

```

```

stati=findobj(gcf,'Tag','StaticText18');
set(stati,'String','Pseudo Random');
if strcmp(rsst,'1')
    g=(1-exp(-(n*damp)).*sin(rand(1)*ww.*n+p));
    a=int32(g);
elseif strcmp(rsst,'0')
    a=(1-exp(-(n*damp)).*sin(rand(1)*ww.*n+p));
end
if strcmp(wall,'1')
    b=a+0.1*sin(2*pi*50*t);
elseif strcmp(wall,'0')
    b=a;
end
if strcmp(rectif,'1')
    lfw8=double(abs(b));
elseif strcmp(rectif,'0')
    lfw8=double(b);
end
end
for i=1:100,
    waitbar(i/100);
end
close(h);
elseif popup8==2
    lfomaker;
elseif popup8==3
    static=findobj(gcf,'Tag','StaticText28');
    set(static,'String','Smart');
    static1=findobj(gcf,'Tag','StaticText38');
    set(static1,'String','Smart');
    stati=findobj(gcf,'Tag','StaticText18');
    set(stati,'String','Smart');
    if data.channel==1
        len=data.count;
    elseif data.channel==2
        len=(data.count)/2;
    end
    assignin('base','maxdelays',md);
    assignin('base','lengthfile',len);
    assignin('base','samp',data.samplerate);
    h=waitbar(0,'Passing data to SmartPlayer System');
    save('xchangesmart');
    for i=1:100,
        waitbar(i/100);
    end
    close(h);
    smartplayer;

```

```

h3=warndlg('Please maintain this Dialog Opened (dont push OK) until you have
finished to use the SmartPlayer System (he has the control of the entire system
now!!!)', 'WARNING!!!');
    uiwait(h3);
    passed=load('xchangesmart.mat');
    lfw8=passed.lfw;
    lfw8=abs(lfw8)';
end
case 'getvoice9'
    popup9handle=findobj(gcf, 'Tag', 'PopupMenu9');
    popup9=get(popup9handle, 'Value');
    mdh=findobj(gcf, 'Tag', 'EditText20');
    md=ceil((str2num(get(mdh, 'String'))/1000)*data.samplerate);
    if popup9==1
        [filename,filepath]=uigetfile('*.lfo', 'Get LFW from which file?');
        path=[filepath];
        name=[filename];
        fid=fopen(strcat(path,name), 'r');
        lfwtype=fscanf(fid, '%s\n%');
        rsst=fscanf(fid, '%s\n%');
        wall=fscanf(fid, '%s\n%');
        rate=fscanf(fid, '%s\n%');
        amp=fscanf(fid, '%s\n%');
        phase=fscanf(fid, '%s\n%');
        offset=fscanf(fid, '%s\n%');
        damping=fscanf(fid, '%s\n%');
        rectif=fscanf(fid, '%s\n%');
        status=fclose(fid);
        h=waitbar(0, 'Gathering LFO Parameters. Please Wait...');
        x=data.y;
        n=1:length(x)+md;
    am9=str2num(amp);
    damp=str2num(damping);
    r=str2num(rate);
    rin=1/r;
    ww=2*pi/round(rin*data.samplerate);
    p=str2num(phase);
    static=findobj(gcf, 'Tag', 'StaticText29');
    set(static, 'String', rate);
    static1=findobj(gcf, 'Tag', 'StaticText39');
    set(static1, 'String', amp);
    if strcmp(lfwtype, 'sin')
        stati=findobj(gcf, 'Tag', 'StaticText19');
        set(stati, 'String', 'Sine');
        if strcmp(rsst, '1')
            g=(1-exp(-(n*damp)).*sin(ww.*n+p));
            a=int32(g);

```

```

elseif strcmp(rsst, '0')
    a=(1-exp(-(n*damp)).*sin(ww.*n+p));
end
if strcmp(wall, '1')
    b=a+0.1*sin(2*pi*50*t);
elseif strcmp(wall, '0')
    b=a;
end
if strcmp(rectif, '1')
    lfw9=double(abs(b));
elseif strcmp(rectif, '0')
    lfw9=double(b);
end
elseif strcmp(lfwtype, 'cos')
    stati=findobj(gcf, 'Tag', 'StaticText19');
    set(stati, 'String', 'Cosine');
    if strcmp(rsst, '1')
        g=(1-exp(-(n*damp)).*cos(ww.*n+p));
        a=int32(g);
    elseif strcmp(rsst, '0')
        a=(1-exp(-(n*damp)).*cos(ww.*n+p));
    end
    if strcmp(wall, '1')
        b=a+0.1*sin(2*pi*50*t);
    elseif strcmp(wall, '0')
        b=a;
    end
    if strcmp(rectif, '1')
        lfw9=double(abs(b));
    elseif strcmp(rectif, '0')
        lfw9=double(b);
    end
elseif strcmp(lfwtype, 'tri')
    stati=findobj(gcf, 'Tag', 'StaticText19');
    set(stati, 'String', 'Triangular');
    if strcmp(rsst, '1')
        g=(1-exp(-(n*damp)).*sawtooth(ww.*n, p+0.5));
        a=int32(g);
    elseif strcmp(rsst, '0')
        a=(1-exp(-(n*damp)).*sawtooth(ww.*n, p+0.5));
    end
    if strcmp(wall, '1')
        b=a+0.1*sin(2*pi*50*t);
    elseif strcmp(wall, '0')
        b=a;
    end
    if strcmp(rectif, '1')

```

```

        lfw9=double(abs(b));
elseif strcmp(rectif,'0')
        lfw9=double(b);
end
elseif strcmp(lfwtype,'squ')
        stati=findobj(gcf,'Tag','StaticText19');
        set(stati,'String','Square');
        if strcmp(rsst,'1')
                g=(1-exp(-(n*damp)).*square(ww.*n,p));
                a=int32(g);
elseif strcmp(rsst,'0')
                a=(1-exp(-(n*damp)).*square(ww.*n,p));
end
if strcmp(wall,'1')
        b=a+0.1*sin(2*pi*50*t);
elseif strcmp(wall,'0')
        b=a;
end
if strcmp(rectif,'1')
        lfw9=double(abs(b));
elseif strcmp(rectif,'0')
        lfw9=double(b);
end
elseif strcmp(lfwtype,'exp')
        h1=warndlg('Sorry but this is not a type of LFW usable with this kind of
algorithm','Warning!!!');
        uiwait(h1);
elseif strcmp(lfwtype,'saw')
        stati=findobj(gcf,'Tag','StaticText19');
        set(stati,'String','Sawtooth');
        if strcmp(rsst,'1')
                g=(1-exp(-(n*damp)).*sawtooth(ww.*n,p));
                a=int32(g);
elseif strcmp(rsst,'0')
                a=(1-exp(-(n*damp)).*sawtooth(ww.*n,p));
end
if strcmp(wall,'1')
        b=a+0.1*sin(2*pi*50*t);
elseif strcmp(wall,'0')
        b=a;
end
if strcmp(rectif,'1')
        lfw9=double(abs(b));
elseif strcmp(rectif,'0')
        lfw9=double(b);
end
elseif strcmp(lfwtype,'ran')

```

```

stati=findobj(gcf,'Tag','StaticText19');
set(stati,'String','Pseudo Random');
if strcmp(rsst,'1')
    g=(1-exp(-(n*damp)).*sin(rand(1)*ww.*n+p));
    a=int32(g);
elseif strcmp(rsst,'0')
    a=(1-exp(-(n*damp)).*sin(rand(1)*ww.*n+p));
end
if strcmp(wall,'1')
    b=a+0.1*sin(2*pi*50*t);
elseif strcmp(wall,'0')
    b=a;
end
if strcmp(rectif,'1')
    lfw9=double(abs(b));
elseif strcmp(rectif,'0')
    lfw9=double(b);
end
end
for i=1:100,
    waitbar(i/100);
end
close(h);
elseif popup9==2
    lfomaker;
elseif popup9==3
    static=findobj(gcf,'Tag','StaticText29');
    set(static,'String','Smart');
    static1=findobj(gcf,'Tag','StaticText39');
    set(static1,'String','Smart');
    stati=findobj(gcf,'Tag','StaticText19');
    set(stati,'String','Smart');
    if data.channel==1
        len=data.count;
    elseif data.channel==2
        len=(data.count)/2;
    end
    assignin('base','maxdelays',md);
    assignin('base','lengthfile',len);
    assignin('base','samp',data.samplerate);
    h=waitbar(0,'Passing data to SmartPlayer System');
    save('xchangesmart');
    for i=1:100,
        waitbar(i/100);
    end
    close(h);
    smartplayer;

```

```

h3=warndlg('Please maintain this Dialog Opened (dont push OK) until you have
finished to use the SmartPlayer System (he has the control of the entire system
now!!!)', 'WARNING!!!');
    uiwait(h3);
    passed=load('xchangesmart.mat');
    lfw9=passed.lfw;
    lfw9=abs(lfw9)';
end
case 'getvoicel0'
    popup10handle=findobj(gcf,'Tag','PopupMenu10');
    popup10=get(popup10handle,'Value');
    mdh=findobj(gcf,'Tag','EditText20');
    md=ceil((str2num(get(mdh,'String'))/1000)*data.samplerate);
    if popup10==1
        [filename,filepath]=uigetfile('*.lfo','Get LFW from which file?');
        path=[filepath];
        name=[filename];
        fid=fopen(strcat(path,name),'r');
        lfwtype=fscanf(fid,'%s\n%');
        rsst=fscanf(fid,'%s\n%');
        wall=fscanf(fid,'%s\n%');
        rate=fscanf(fid,'%s\n%');
        amp=fscanf(fid,'%s\n%');
        phase=fscanf(fid,'%s\n%');
        offset=fscanf(fid,'%s\n%');
        damping=fscanf(fid,'%s\n%');
        rectif=fscanf(fid,'%s\n%');
        status=fclose(fid);
        h=waitbar(0,'Gathering LFO Parameters. Please Wait...');
        x=data.y;
        n=1:length(x)+md;
    am10=str2num(amp);
    damp=str2num(damping);
        r=str2num(rate);
        rinv=1/r;
        ww=2*pi/round(rinv*data.samplerate);
        p=str2num(phase);
        static=findobj(gcf,'Tag','StaticText210');
        set(static,'String',rate);
        static1=findobj(gcf,'Tag','StaticText310');
        set(static1,'String',amp);
        if strcmp(lfwtype,'sin')
            stati=findobj(gcf,'Tag','StaticText110');
            set(stati,'String','Sine');
            if strcmp(rsst,'1')
                g=(1-exp(-(n*damp)).*sin(ww.*n+p));
                a=int32(g);
            end
        end
    end
end

```

```

elseif strcmp(rsst, '0')
    a=(1-exp(-(n*damp)).*sin(ww.*n+p));
end
if strcmp(wall, '1')
    b=a+0.1*sin(2*pi*50*t);
elseif strcmp(wall, '0')
    b=a;
end
if strcmp(rectif, '1')
    lfw10=double(abs(b));
elseif strcmp(rectif, '0')
    lfw10=double(b);
end
elseif strcmp(lfwtype, 'cos')
    stati=findobj(gcf, 'Tag', 'StaticText110');
    set(stati, 'String', 'Cosine');
    if strcmp(rsst, '1')
        g=(1-exp(-(n*damp)).*cos(ww.*n+p));
        a=int32(g);
    elseif strcmp(rsst, '0')
        a=(1-exp(-(n*damp)).*cos(ww.*n+p));
    end
    if strcmp(wall, '1')
        b=a+0.1*sin(2*pi*50*t);
    elseif strcmp(wall, '0')
        b=a;
    end
    if strcmp(rectif, '1')
        lfw10=double(abs(b));
    elseif strcmp(rectif, '0')
        lfw10=double(b);
    end
elseif strcmp(lfwtype, 'tri')
    stati=findobj(gcf, 'Tag', 'StaticText110');
    set(stati, 'String', 'Triangular');
    if strcmp(rsst, '1')
        g=(1-exp(-(n*damp)).*sawtooth(ww.*n, p+0.5));
        a=int32(g);
    elseif strcmp(rsst, '0')
        a=(1-exp(-(n*damp)).*sawtooth(ww.*n, p+0.5));
    end
    if strcmp(wall, '1')
        b=a+0.1*sin(2*pi*50*t);
    elseif strcmp(wall, '0')
        b=a;
    end
    if strcmp(rectif, '1')

```

```

        lfw10=double(abs(b));
elseif strcmp(rectif,'0')
        lfw10=double(b);
end
elseif strcmp(lfwtype,'squ')
        stati=findobj(gcf,'Tag','StaticText110');
        set(stati,'String','Square');
        if strcmp(rsst,'1')
                g=(1-exp(-(n*damp)).*square(ww.*n,p));
                a=int32(g);
        elseif strcmp(rsst,'0')
                a=(1-exp(-(n*damp)).*square(ww.*n,p));
        end
if strcmp(wall,'1')
        b=a+0.1*sin(2*pi*50*t);
elseif strcmp(wall,'0')
        b=a;
end
if strcmp(rectif,'1')
        lfw10=double(abs(b));
elseif strcmp(rectif,'0')
        lfw10=double(b);
end
elseif strcmp(lfwtype,'exp')
        hl=warndlg('Sorry but this is not a type of LFW usable with this kind of
algorithm','Warning!!!');
        uiwait(hl);
elseif strcmp(lfwtype,'saw')
        stati=findobj(gcf,'Tag','StaticText110');
        set(stati,'String','Sawtooth');
        if strcmp(rsst,'1')
                g=(1-exp(-(n*damp)).*sawtooth(ww.*n,p));
                a=int32(g);
        elseif strcmp(rsst,'0')
                a=(1-exp(-(n*damp)).*sawtooth(ww.*n,p));
        end
if strcmp(wall,'1')
        b=a+0.1*sin(2*pi*50*t);
elseif strcmp(wall,'0')
        b=a;
end
if strcmp(rectif,'1')
        lfw10=double(abs(b));
elseif strcmp(rectif,'0')
        lfw10=double(b);
end
elseif strcmp(lfwtype,'ran')

```

```

stati=findobj(gcf,'Tag','StaticText110');
set(stati,'String','Pseudo Random');
if strcmp(rsst,'1')
    g=(1-exp(-(n*damp)).*sin(rand(1)*ww.*n+p));
    a=int32(g);
elseif strcmp(rsst,'0')
    a=(1-exp(-(n*damp)).*sin(rand(1)*ww.*n+p));
end
if strcmp(wall,'1')
    b=a+0.1*sin(2*pi*50*t);
elseif strcmp(wall,'0')
    b=a;
end
if strcmp(rectif,'1')
    lfw10=double(abs(b));
elseif strcmp(rectif,'0')
    lfw10=double(b);
end
end
for i=1:100,
    waitbar(i/100);
end
close(h);
elseif popup10==2
    lfomaker;
elseif popup10==3
    static=findobj(gcf,'Tag','StaticText210');
    set(static,'String','Smart');
    static1=findobj(gcf,'Tag','StaticText310');
    set(static1,'String','Smart');
    stati=findobj(gcf,'Tag','StaticText110');
    set(stati,'String','Smart');
    if data.channel==1
        len=data.count;
    elseif data.channel==2
        len=(data.count)/2;
    end
    assignin('base','maxdelays',md);
    assignin('base','lengthfile',len);
    assignin('base','samp',data.samplerate);
    h=waitbar(0,'Passing data to SmartPlayer System');
    save('xchangesmart');
    for i=1:100,
        waitbar(i/100);
    end
    close(h);
    smartplayer;

```

```

h3=warndlg('Please maintain this Dialog Opened (dont push OK) until you have
finished to use the SmartPlayer System (he has the control of the entire system
now!!!)', 'WARNING!!!');
    uiwait(h3);
    passed=load('xchangesmart.mat');
    lfw10=passed.lfw;
    lfw10=abs(lfw10)';
end
case 'getvoicell'
    popup11handle=findobj(gcf,'Tag','PopupMenu11');
    popup11=get(popup11handle,'Value');
    mdh=findobj(gcf,'Tag','EditText20');
    md=ceil((str2num(get(mdh,'String'))/1000)*data.samplerate);
    if popup11==1
        [filename,filepath]=uigetfile('*.lfo','Get LFW from which file?');
        path=[filepath];
        name=[filename];
        fid=fopen(strcat(path,name),'r');
        lfwtype=fscanf(fid,'%s\n%');
        rsst=fscanf(fid,'%s\n%');
        wall=fscanf(fid,'%s\n%');
        rate=fscanf(fid,'%s\n%');
        amp=fscanf(fid,'%s\n%');
        phase=fscanf(fid,'%s\n%');
        offset=fscanf(fid,'%s\n%');
        damping=fscanf(fid,'%s\n%');
        rectif=fscanf(fid,'%s\n%');
        status=fclose(fid);
        h=waitbar(0,'Gathering LFO Parameters. Please Wait...');
        x=data.y;
        n=1:length(x)+md;
    am11=str2num(amp);
    damp=str2num(damping);
        r=str2num(rate);
        rinv=1/r;
        ww=2*pi/round(rinv*data.samplerate);
        p=str2num(phase);
        static=findobj(gcf,'Tag','StaticText211');
        set(static,'String',rate);
        static1=findobj(gcf,'Tag','StaticText311');
        set(static1,'String',amp);
        if strcmp(lfwtype,'sin')
            stati=findobj(gcf,'Tag','StaticText111');
            set(stati,'String','Sine');
            if strcmp(rsst,'1')
                g=(1-exp(-(n*damp)).*sin(ww.*n+p));
                a=int32(g);
            end
        end
    end
end

```

```

elseif strcmp(rsst, '0')
    a=(1-exp(-(n*damp)).*sin(ww.*n+p));
end
if strcmp(wall, '1')
    b=a+0.1*sin(2*pi*50*t);
elseif strcmp(wall, '0')
    b=a;
end
if strcmp(rectif, '1')
    lfw11=double(abs(b));
elseif strcmp(rectif, '0')
    lfw11=double(b);
end
elseif strcmp(lfwtype, 'cos')
    stati=findobj(gcf, 'Tag', 'StaticText111');
    set(stati, 'String', 'Cosine');
    if strcmp(rsst, '1')
        g=(1-exp(-(n*damp)).*cos(ww.*n+p));
        a=int32(g);
    elseif strcmp(rsst, '0')
        a=(1-exp(-(n*damp)).*cos(ww.*n+p));
    end
    if strcmp(wall, '1')
        b=a+0.1*sin(2*pi*50*t);
    elseif strcmp(wall, '0')
        b=a;
    end
    if strcmp(rectif, '1')
        lfw11=double(abs(b));
    elseif strcmp(rectif, '0')
        lfw11=double(b);
    end
elseif strcmp(lfwtype, 'tri')
    stati=findobj(gcf, 'Tag', 'StaticText111');
    set(stati, 'String', 'Triangular');
    if strcmp(rsst, '1')
        g=(1-exp(-(n*damp)).*sawtooth(ww.*n, p+0.5));
        a=int32(g);
    elseif strcmp(rsst, '0')
        a=(1-exp(-(n*damp)).*sawtooth(ww.*n, p+0.5));
    end
    if strcmp(wall, '1')
        b=a+0.1*sin(2*pi*50*t);
    elseif strcmp(wall, '0')
        b=a;
    end
    if strcmp(rectif, '1')

```

```

        lfw11=double(abs(b));
elseif strcmp(rectif,'0')
        lfw11=double(b);
end
elseif strcmp(lfwtype,'squ')
        stati=findobj(gcf,'Tag','StaticText111');
        set(stati,'String','Square');
        if strcmp(rsst,'1')
                g=(1-exp(-(n*damp)).*square(ww.*n,p));
                a=int32(g);
        elseif strcmp(rsst,'0')
                a=(1-exp(-(n*damp)).*square(ww.*n,p));
        end
if strcmp(wall,'1')
        b=a+0.1*sin(2*pi*50*t);
elseif strcmp(wall,'0')
        b=a;
end
if strcmp(rectif,'1')
        lfw11=double(abs(b));
elseif strcmp(rectif,'0')
        lfw11=double(b);
end
elseif strcmp(lfwtype,'exp')
        h1=warndlg('Sorry but this is not a type of LFW usable with this kind of
algorithm','Warning!!!');
        uiwait(h1);
elseif strcmp(lfwtype,'saw')
        stati=findobj(gcf,'Tag','StaticText111');
        set(stati,'String','Sawtooth');
        if strcmp(rsst,'1')
                g=(1-exp(-(n*damp)).*sawtooth(ww.*n,p));
                a=int32(g);
        elseif strcmp(rsst,'0')
                a=(1-exp(-(n*damp)).*sawtooth(ww.*n,p));
        end
if strcmp(wall,'1')
        b=a+0.1*sin(2*pi*50*t);
elseif strcmp(wall,'0')
        b=a;
end
if strcmp(rectif,'1')
        lfw11=double(abs(b));
elseif strcmp(rectif,'0')
        lfw11=double(b);
end
elseif strcmp(lfwtype,'ran')

```

```

stati=findobj(gcf,'Tag','StaticText111');
set(stati,'String','Pseudo Random');
if strcmp(rsst,'1')
    g=(1-exp(-(n*damp)).*sin(rand(1)*ww.*n+p));
    a=int32(g);
elseif strcmp(rsst,'0')
    a=(1-exp(-(n*damp)).*sin(rand(1)*ww.*n+p));
end
if strcmp(wall,'1')
    b=a+0.1*sin(2*pi*50*t);
elseif strcmp(wall,'0')
    b=a;
end
if strcmp(rectif,'1')
    lfw11=double(abs(b));
elseif strcmp(rectif,'0')
    lfw11=double(b);
end
end
for i=1:100,
    waitbar(i/100);
end
close(h);
elseif popup11==2
    lfomaker;
elseif popup11==3
    static=findobj(gcf,'Tag','StaticText211');
    set(static,'String','Smart');
    static1=findobj(gcf,'Tag','StaticText311');
    set(static1,'String','Smart');
    stati=findobj(gcf,'Tag','StaticText111');
    set(stati,'String','Smart');
    if data.channel==1
        len=data.count;
    elseif data.channel==2
        len=(data.count)/2;
    end
    assignin('base','maxdelays',md);
    assignin('base','lengthfile',len);
    assignin('base','samp',data.samplerate);
    h=waitbar(0,'Passing data to SmartPlayer System');
    save('xchangesmart');
    for i=1:100,
        waitbar(i/100);
    end
    close(h);
    smartplayer;

```

```

h3=warndlg('Please maintain this Dialog Opened (dont push OK) until you have
finished to use the SmartPlayer System (he has the control of the entire system
now!!!)', 'WARNING!!!');
    uiwait(h3);
    passed=load('xchangesmart.mat');
    lfw11=passed.lfw;
    lfw11=abs(lfw11)';
end
case 'getvoicel2'
    popup12handle=findobj(gcf,'Tag','PopupMenu12');
    popup12=get(popup12handle,'Value');
    mdh=findobj(gcf,'Tag','EditText20');
    md=ceil((str2num(get(mdh,'String'))/1000)*data.samplerate);
    if popup12==1
        [filename,filepath]=uigetfile('*.lfo','Get LFW from which file?');
        path=[filepath];
        name=[filename];
        fid=fopen(strcat(path,name),'r');
        lfwtype=fscanf(fid,'%s\n%');
        rsst=fscanf(fid,'%s\n%');
        wall=fscanf(fid,'%s\n%');
        rate=fscanf(fid,'%s\n%');
        amp=fscanf(fid,'%s\n%');
        phase=fscanf(fid,'%s\n%');
        offset=fscanf(fid,'%s\n%');
        damping=fscanf(fid,'%s\n%');
        rectif=fscanf(fid,'%s\n%');
        status=fclose(fid);
        h=waitbar(0,'Gathering LFO Parameters. Please Wait...');
        x=data.y;
        n=1:length(x)+md;
    am12=str2num(amp);
    damp=str2num(damping);
        r=str2num(rate);
        rinv=1/r;
        ww=2*pi/round(rinv*data.samplerate);
        p=str2num(phase);
        static=findobj(gcf,'Tag','StaticText212');
        set(static,'String',rate);
        static1=findobj(gcf,'Tag','StaticText312');
        set(static1,'String',amp);
        if strcmp(lfwtype,'sin')
            stati=findobj(gcf,'Tag','StaticText112');
            set(stati,'String','Sine');
            if strcmp(rsst,'1')
                g=(1-exp(-(n*damp)).*sin(ww.*n+p));
                a=int32(g);
            end
        end
    end
end

```

```

elseif strcmp(rsst, '0')
    a=(1-exp(-(n*damp)).*sin(ww.*n+p));
end
if strcmp(wall, '1')
    b=a+0.1*sin(2*pi*50*t);
elseif strcmp(wall, '0')
    b=a;
end
if strcmp(rectif, '1')
    lfw12=double(abs(b));
elseif strcmp(rectif, '0')
    lfw12=double(b);
end
elseif strcmp(lfwtype, 'cos')
    stati=findobj(gcf, 'Tag', 'StaticText112');
    set(stati, 'String', 'Cosine');
    if strcmp(rsst, '1')
        g=(1-exp(-(n*damp)).*cos(ww.*n+p));
        a=int32(g);
    elseif strcmp(rsst, '0')
        a=(1-exp(-(n*damp)).*cos(ww.*n+p));
    end
    if strcmp(wall, '1')
        b=a+0.1*sin(2*pi*50*t);
    elseif strcmp(wall, '0')
        b=a;
    end
    if strcmp(rectif, '1')
        lfw12=double(abs(b));
    elseif strcmp(rectif, '0')
        lfw12=double(b);
    end
elseif strcmp(lfwtype, 'tri')
    stati=findobj(gcf, 'Tag', 'StaticText112');
    set(stati, 'String', 'Triangular');
    if strcmp(rsst, '1')
        g=(1-exp(-(n*damp)).*sawtooth(ww.*n, p+0.5));
        a=int32(g);
    elseif strcmp(rsst, '0')
        a=(1-exp(-(n*damp)).*sawtooth(ww.*n, p+0.5));
    end
    if strcmp(wall, '1')
        b=a+0.1*sin(2*pi*50*t);
    elseif strcmp(wall, '0')
        b=a;
    end
    if strcmp(rectif, '1')

```

```

        lfw12=double(abs(b));
elseif strcmp(rectif,'0')
    lfw12=double(b);
end
elseif strcmp(lfwtype,'squ')
    stati=findobj(gcf,'Tag','StaticText112');
    set(stati,'String','Square');
    if strcmp(rsst,'1')
        g=(1-exp(-(n*damp)).*square(ww.*n,p));
        a=int32(g);
    elseif strcmp(rsst,'0')
        a=(1-exp(-(n*damp)).*square(ww.*n,p));
    end
if strcmp(wall,'1')
    b=a+0.1*sin(2*pi*50*t);
elseif strcmp(wall,'0')
    b=a;
end
if strcmp(rectif,'1')
    lfw12=double(abs(b));
elseif strcmp(rectif,'0')
    lfw12=double(b);
end
elseif strcmp(lfwtype,'exp')
    h1=warndlg('Sorry but this is not a type of LFW usable with this kind of
algorithm','Warning!!!');
    uiwait(h1);
elseif strcmp(lfwtype,'saw')
    stati=findobj(gcf,'Tag','StaticText112');
    set(stati,'String','Sawtooth');
    if strcmp(rsst,'1')
        g=(1-exp(-(n*damp)).*sawtooth(ww.*n,p));
        a=int32(g);
    elseif strcmp(rsst,'0')
        a=(1-exp(-(n*damp)).*sawtooth(ww.*n,p));
    end
if strcmp(wall,'1')
    b=a+0.1*sin(2*pi*50*t);
elseif strcmp(wall,'0')
    b=a;
end
if strcmp(rectif,'1')
    lfw12=double(abs(b));
elseif strcmp(rectif,'0')
    lfw12=double(b);
end
elseif strcmp(lfwtype,'ran')

```

```

stati=findobj(gcf,'Tag','StaticText112');
set(stati,'String','Pseudo Random');
if strcmp(rsst,'1')
    g=(1-exp(-(n*damp)).*sin(rand(1)*ww.*n+p));
    a=int32(g);
elseif strcmp(rsst,'0')
    a=(1-exp(-(n*damp)).*sin(rand(1)*ww.*n+p));
end
if strcmp(wall,'1')
    b=a+0.1*sin(2*pi*50*t);
elseif strcmp(wall,'0')
    b=a;
end
if strcmp(rectif,'1')
    lfw12=double(abs(b));
elseif strcmp(rectif,'0')
    lfw12=double(b);
end
end
for i=1:100,
    waitbar(i/100);
end
close(h);
elseif popup12==2
    lfomaker;
elseif popup12==3
    static=findobj(gcf,'Tag','StaticText212');
    set(static,'String','Smart');
    static1=findobj(gcf,'Tag','StaticText312');
    set(static1,'String','Smart');
    stati=findobj(gcf,'Tag','StaticText112');
    set(stati,'String','Smart');
    if data.channel==1
        len=data.count;
    elseif data.channel==2
        len=(data.count)/2;
    end
    assignin('base','maxdelays',md);
    assignin('base','lengthfile',len);
    assignin('base','samp',data.samplerate);
    h=waitbar(0,'Passing data to SmartPlayer System');
    save('xchangesmart');
    for i=1:100,
        waitbar(i/100);
    end
    close(h);
    smartplayer;

```

```

h3=warndlg('Please maintain this Dialog Opened (dont push OK) until you have
finished to use the SmartPlayer System (he has the control of the entire system
now!!!)', 'WARNING!!!');
uiwait(h3);
passed=load('xchangesmart.mat');
lfw12=passed.lfw;
lfw12=abs(lfw12)';
end

```

Questo case consente la visualizzazione delle LFW utilizzate per il processo di Chorusing. La forma d'onda viene visualizzata solo se la checkbox relativa ad essa, è spuntata. Il plot viene fatto in funzione del tempo del segnale originale.

```

case 'plotlfw'
check1h=findobj(gcf,'Tag','Checkbox1');
check1=get(check1h,'Value');
check2h=findobj(gcf,'Tag','Checkbox2');
check2=get(check2h,'Value');
check3h=findobj(gcf,'Tag','Checkbox3');
check3=get(check3h,'Value');
check4h=findobj(gcf,'Tag','Checkbox4');
check4=get(check4h,'Value');
check5h=findobj(gcf,'Tag','Checkbox5');
check5=get(check5h,'Value');
check6h=findobj(gcf,'Tag','Checkbox6');
check6=get(check6h,'Value');
check7h=findobj(gcf,'Tag','Checkbox7');
check7=get(check7h,'Value');
check8h=findobj(gcf,'Tag','Checkbox8');
check8=get(check8h,'Value');
check9h=findobj(gcf,'Tag','Checkbox9');
check9=get(check9h,'Value');
check10h=findobj(gcf,'Tag','Checkbox10');
check10=get(check10h,'Value');
check11h=findobj(gcf,'Tag','Checkbox11');
check11=get(check11h,'Value');
check12h=findobj(gcf,'Tag','Checkbox12');
check12=get(check12h,'Value');
h5=figure('Name','LFW Used Plot','NumberTitle','off');
if check1==1
t=(1:length(lfw1))/data.samplerate);
subplot(431);
plot(t,lfw1);
ylabel('Depth');
title('LFW1');

```

```
end
if check2==1
    t=(1:length(lfw2))/data.samplerate);
    subplot(432);
    plot(t,lfw2);
    ylabel('Depth');
    title('LFW2');
end
if check3==1
    t=(1:length(lfw3))/data.samplerate);
    subplot(433);
    plot(t,lfw3);
    ylabel('Depth');
    title('LFW3');
end
if check4==1
    t=(1:length(lfw4))/data.samplerate);
    subplot(434);
    plot(t,lfw4);
    ylabel('Depth');
    title('LFW4');
end
if check5==1
    t=(1:length(lfw5))/data.samplerate);
    subplot(435);
    plot(t,lfw5);
    ylabel('Depth');
    title('LFW5');
end
if check6==1
    t=(1:length(lfw6))/data.samplerate);
    subplot(436);
    plot(t,lfw6);
    ylabel('Depth');
    title('LFW6');
end
if check7==1
    t=(1:length(lfw7))/data.samplerate);
    subplot(437);
    plot(t,lfw7);
    ylabel('Depth');
    title('LFW7');
end
if check8==1
    t=(1:length(lfw8))/data.samplerate);
    subplot(438);
    plot(t,lfw8);
```

```

        ylabel('Depth');
        title('LFW8');
    end
    if check9==1
        t=(1:length(lfw9))/data.samplerate);
        subplot(4,3,9);
        plot(t,lfw9);
        ylabel('Depth');
        title('LFW9');
    end
    if check10==1
        t=(1:length(lfw10))/data.samplerate);
        subplot(4,3,10);
        plot(t,lfw10);
        ylabel('Depth');
        title('LFW10');
    end
    if check11==1
        t=(1:length(lfw11))/data.samplerate);
        subplot(4,3,11);
        plot(t,lfw11);
        ylabel('Depth');
        title('LFW11');
    end
    if check12==1
        t=(1:length(lfw12))/data.samplerate);
        subplot(4,3,12);
        plot(t,lfw12);
        ylabel('Depth');
        title('LFW12');
    end
end

```

Questo case rappresenta l'applicazione dell'algoritmo che possiamo riassumere così: viene valutato se il segnale è stereofonico o meno e si ricava la porzione di segnale originale che si vuole processare. Si ricava il valore del Base Delay, del Vibrato Rate per ogni voce e si inizializza la linea di ritardo con un numero di campioni definito dal samplerate moltiplicato dal Base Delay time e il vettore di uscita. Per ogni voce si valuta se essa è selezionata per il processing e ne si calcola il suo effetto relativo sul segnale. Al termine dell'algoritmo, si sommano i risultati di ogni voce, per ottenere il suono di Chorus finale, se la funzione di costruzione della stereofonia è abilitata, si costruisce il campo stereofonico, assegnando ad ogni voce un settore (in porzioni di un 1/12 dal momento che le voci sono al massimo 12) con la voce centrale diretta al centro e si sommano le voci. Il segnale di uscita viene poi riscalato nell'intervallo [-1:1) e plottato.

```

case 'apply'
    if data.channel==2
        chanh=findobj(gcf,'Tag','PopupMenu20');
        chan=get(chanh,'Value');
        if chan==1
            x=data.lefty;
        elseif chan==2
            x=data.righty;
        end
    end
    mdh=findobj(gcf,'Tag','EditText20');
    md=ceil((str2num(get(mdh,'String'))/1000)*data.samplerate);
    viberate1h=findobj(gcf,'Tag','EditText1');
    viberate1=str2num(get(viberate1h,'String'));
    viberate2h=findobj(gcf,'Tag','EditText2');
    viberate2=str2num(get(viberate2h,'String'));
    viberate3h=findobj(gcf,'Tag','EditText3');
    viberate3=str2num(get(viberate3h,'String'));
    viberate4h=findobj(gcf,'Tag','EditText4');
    viberate4=str2num(get(viberate4h,'String'));
    viberate5h=findobj(gcf,'Tag','EditText5');
    viberate5=str2num(get(viberate5h,'String'));
    viberate6h=findobj(gcf,'Tag','EditText6');
    viberate6=str2num(get(viberate6h,'String'));
    viberate7h=findobj(gcf,'Tag','EditText7');
    viberate7=str2num(get(viberate7h,'String'));
    viberate8h=findobj(gcf,'Tag','EditText8');
    viberate8=str2num(get(viberate8h,'String'));
    viberate9h=findobj(gcf,'Tag','EditText9');
    viberate9=str2num(get(viberate9h,'String'));
    viberate10h=findobj(gcf,'Tag','EditText10');
    viberate10=str2num(get(viberate10h,'String'));
    viberate11h=findobj(gcf,'Tag','EditText11');
    viberate11=str2num(get(viberate11h,'String'));
    viberate12h=findobj(gcf,'Tag','EditText12');
    viberate12=str2num(get(viberate12h,'String'));
    n=1:length(x)+md;
    r=round((str2num(get(mdh,'String'))/1000)*data.samplerate);
    h=waitbar(0,'Chorus in progress (this could take a lot). Please wait...');
    for i=1:100,
        waitbar(i/100);
    end
    z=zeros(md,1);
    m=max(abs(x));
    y=[z;x;z];

```

```

check1h=findobj(gcf,'Tag','Checkbox1');
check1=get(check1h,'Value');
check2h=findobj(gcf,'Tag','Checkbox2');
check2=get(check2h,'Value');
check3h=findobj(gcf,'Tag','Checkbox3');
check3=get(check3h,'Value');
check4h=findobj(gcf,'Tag','Checkbox4');
check4=get(check4h,'Value');
check5h=findobj(gcf,'Tag','Checkbox5');
check5=get(check5h,'Value');
check6h=findobj(gcf,'Tag','Checkbox6');
check6=get(check6h,'Value');
check7h=findobj(gcf,'Tag','Checkbox7');
check7=get(check7h,'Value');
check8h=findobj(gcf,'Tag','Checkbox8');
check8=get(check8h,'Value');
check9h=findobj(gcf,'Tag','Checkbox9');
check9=get(check9h,'Value');
check10h=findobj(gcf,'Tag','Checkbox10');
check10=get(check10h,'Value');
check11h=findobj(gcf,'Tag','Checkbox11');
check11=get(check11h,'Value');
check12h=findobj(gcf,'Tag','Checkbox12');
check12=get(check12h,'Value');
if check1==1
    bal1=[1 0];
    if popup1==1
        b1=n+md-round((r/2)*lfw1);
    elseif popup1==3
        b1=n+md-round((r/2)*lfw1');
    end
    f=(1:length(b1))/data.samplerate;
    if viberatel==0
        g=ones(size(b1));
    else
        g=sin(2*pi*viberatel*f);
    end
    vib1=reshape(g,length(b1),1);
elseif check1==0
    d=length(x)+md;
    bal1=zeros(d,2);
    am1=1;
    b1=1;
    vib1=0;
end
if check2==1
    bal2=[1/12 11/12];

```

```

if popup2==1
    b2=n+md-round((r/2).*1fw2);
elseif popup2==3
    b2=n+md-round((r/2).*1fw2');
end
f=(1:length(b2))/data.samplerate;
if viberate2==0
    g=ones(size(b2));
else
    g=sin(2*pi*viberate2*f);
end
vibe2=reshape(g,length(b2),1);
elseif check2==0
    d=length(x)+md;
    bal2=zeros(d,2);
    am2=1;
    b2=1;
    vibe2=0;
end
if check3==1
    bal3=[2/12 10/12];
    if popup3==1
        b3=n+md-round((r/2).*1fw3);
    elseif popup3==3
        b3=n+md-round((r/2).*1fw3');
    end
    f=(1:length(b3))/data.samplerate;
    if viberate3==0
        g=ones(size(b3));
    else
        g=sin(2*pi*viberate3*f);
    end
    vibe3=reshape(g,length(b3),1);
elseif check3==0
    d=length(x)+md;
    bal3=zeros(d,2);
    am3=1;
    b3=1;
    vibe3=0;
end
if check4==1
    bal4=[3/12 9/12];
    if popup4==1
        b4=n+md-round((r/2).*1fw4);
    elseif popup4==3
        b4=n+md-round((r/2).*1fw4');
    end
end

```

```

f=(1:length(b4))/data.samplerate;
if viberate4==0
    g=ones(size(b4));
else
    g=sin(2*pi*viberate4*f);
end
vibe4=reshape(g,length(b4),1);
elseif check4==0
    d=length(x)+md;
    bal4=zeros(d,2);
    am4=1;
    b4=1;
    vibe4=0;
end
if check5==1
    bal5=[4/12 8/12];
    if popup5==1
        b5=n+md-round((r/2).*1fw5);
    elseif popup5==3
        b5=n+md-round((r/2).*1fw5');
    end
    f=(1:length(b5))/data.samplerate;
    if viberate5==0
        g=ones(size(b5));
    else
        g=sin(2*pi*viberate5*f);
    end
    vibe5=reshape(g,length(b5),1);
elseif check5==0
    d=length(x)+md;
    bal5=zeros(d,2);
    am5=1;
    b5=1;
    vibe5=0;
end
if check6==1
    bal6=[5/12 7/12];
    if popup6==1
        b6=n+md-round((r/2).*1fw6);
    elseif popup6==3
        b6=n+md-round((r/2).*1fw6');
    end
    f=(1:length(b6))/data.samplerate;
    if viberate6==0
        g=ones(size(b6));
    else
        g=sin(2*pi*viberate6*f);

```

```

    end
    vibe6=reshape(g,length(b6),1);
elseif check6==0
    d=length(x)+md;
    bal6=zeros(d,2);
    am6=1;
    b6=1;
    vibe6=0;
end
if check7==1
    bal7=[7/12 5/12];
    if popup7==1
        b7=n+md-round((r/2).*lfw7);
    elseif popup7==3
        b7=n+md-round((r/2).*lfw7');
    end
    f=(1:length(b7))/data.samplerate;
    if viberate7==0
        g=ones(size(b7));
    else
        g=sin(2*pi*viberate7*f);
    end
    vibe7=reshape(g,length(b7),1);
elseif check7==0
    d=length(x)+md;
    bal7=zeros(d,2);
    am7=1;
    b7=1;
    vibe7=0;
end
if check8==1
    bal8=[8/12 4/12];
    if popup8==1
        b8=n+md-round((r/2).*lfw8);
    elseif popup8==3
        b8=n+md-round((r/2).*lfw8');
    end
    f=(1:length(b8))/data.samplerate;
    if viberate8==0
        g=ones(size(b8));
    else
        g=sin(2*pi*viberate8*f);
    end
    vibe8=reshape(g,length(b8),1);
elseif check8==0
    d=length(x)+md;
    bal8=zeros(d,2);

```

```

    am8=1;
    b8=1;
    vibe8=0;
end
if check9==1
    bal9=[9/12 3/12];
    if popup9==1
        b9=n+md-round((r/2).*lfw9);
    elseif popup9==3
        b9=n+md-round((r/2).*lfw9');
    end
    f=(1:length(b9))/data.samplerate;
    if viberate9==0
        g=ones(size(b9));
    else
        g=sin(2*pi*viberate9*f);
    end
    vibe9=reshape(g,length(b9),1);
elseif check9==0
    d=length(x)+md;
    bal9=zeros(d,2);
    am9=1;
    b9=1;
    vibe9=0;
end
if check10==1
    bal10=[10/12 2/12];
    if popup10==1
        b10=n+md-round((r/2).*lfw10);
    elseif popup10==3
        b10=n+md-round((r/2).*lfw10');
    end
    f=(1:length(b10))/data.samplerate;
    if viberate10==0
        g=ones(size(b10));
    else
        g=sin(2*pi*viberate10*f);
    end
    vib10=reshape(g,length(b10),1);
elseif check10==0
    d=length(x)+md;
    bal10=zeros(d,2);
    am10=1;
    b10=1;
    vib10=0;
end
if check11==1

```

```

ball1=[11/12 1/12];
if popup1==1
    b11=n+md-round((r/2).*lfw11);
elseif popup1==3
    b11=n+md-round((r/2).*lfw11');
end
f=(1:length(b11))/data.samplerate;
if viberate1==0
    g=ones(size(b11));
else
    g=sin(2*pi*viberate1*f);
end
vib11=reshape(g,length(b11),1);
elseif check1==0
    d=length(x)+md;
    ball1=zeros(d,2);
    am1=1;
    b11=1;
    vib11=0;
end
if check12==1
    ball12=[0 1];
    if popup12==1
        b12=n+md-round((r/2).*lfw12);
    elseif popup12==3
        b12=n+md-round((r/2).*lfw12');
    end
    f=(1:length(b12))/data.samplerate;
    if viberate12==0
        g=ones(size(b12));
    else
        g=sin(2*pi*viberate12*f);
    end
    vib12=reshape(g,length(b12),1);
elseif check12==0
    d=length(x)+md;
    ball12=zeros(d,2);
    am12=1;
    b12=1;
    vib12=0;
end
slidelh=findobj(gcf,'Tag','Slider1');
slide1=get(slidelh,'Value');
phoniah=findobj(gcf,'Tag','Checkbox20');
phonia=get(phoniah,'Value');
if phonia==0

```

```

        y=(1-
slide1).*y(n+md)+slide1.*(vibe1.*(am1*y(b1))+vibe2.*(am2*y(b2))+vibe3.*(am3*y(b3))+vibe4.
*(am4*y(b4))+vibe5.*(am5*y(b5))+vibe6.*(am6*y(b6))+vibe7.*(am7*y(b7))+vibe8.*(am8*y(b8))+
vibe9.*(am9*y(b9))+vibe10.*(am10*y(b10))+vibe11.*(am11*y(b11))+vibe12.*(am12*y(b12)));
        m=m/max(abs(y));
        y=m*y;
elseif phonia==1
        bal0=[0.5 0.5];
        y=[(1-
slide1).*y(n+md)*bal0)+(slide1.*(vibe1.*(am1*y(b1))*bal1)+(vibe2.*(am2*y(b2))*bal2)+(vibe
3.*(am3*y(b3))*bal3)+(vibe4.*(am4*y(b4))*bal4)+(vibe5.*(am5*y(b5))*bal5)+(vibe6.*(am6*y(b
6))*bal6)+(vibe7.*(am7*y(b7))*bal7)+(vibe8.*(am8*y(b8))*bal8)+(vibe9.*(am9*y(b9))*bal9)+(
vibe10.*(am10*y(b10))*bal10)+(vibe11.*(am11*y(b11))*bal11)+(vibe12.*(am12*y(b12))*bal12)
];
end
close(h);
t=[1:length(y)]/data.samplerate];
if data.quantize==8
        y=(y-128)/128;
elseif data.quantize==16
        y=y/32768;
end
figure(h1);
subplot(212);
plot(t,y);
title('Wet Signal');
ylabel('Normalized Values');
if phonia==0
        legend('Monaural Chorused Signal');
elseif phonia==1
        legend('Left Chorused Signal','Right Chorused Signal');
end

```

Questo case, permette di salvare il risultato del processo di modulazione diretta del segnale su un file in formato Microsoft PCM WAVE FORMAT o Motion Picture Expert Group (MPEG1) Layer 3, conservando i parametri di samplerate e profondità di quantizzazione del segnale originale. Per l'encoding si utilizza il software LAME con una modalità di "encoding Extreme" a 256 Kbps con segnale ricampionato a 44.1 KHz.

```

case 'saveasf'
        button3=questdlg('Would you like to save this signal for further processing?','Save
file...','Microsoft PCM WAVE FORMAT','Motion Picture Expert Group (MPEG-1) Layer
3','No','No');
        if strcmp(button3,'Microsoft PCM WAVE FORMAT')

```

```

[newfile,newpath] = uiputfile('*.wav','Save as...');
newpath1=[newpath];
newfile1=[newfile];
h2=waitbar(0,'Saving in Progress... Please Wait');
    wavwrite(y,data.samplerate,data.quantize,strcat(newpath1,newfile1));
for i=1:100,
    waitbar(i/100);
end
close(h2);
elseif strcmp(button3,'Motion Picture Expert Group (MPEG-1) Layer 3')
    [newfile,newpath] = uiputfile('*.mp3','Save as...');
    newpath1=[newpath];
    newfile1=[newfile];
    wavwrite(y,data.samplerate,data.quantize,'tmpfile.wav');
    string=strcat(newpath1,newfile1,'.mp3');
    if data.channel==1
        h2=waitbar(0,'Encoding and Saving in Progress... Please Wait');
        cmd=['lame --quiet -b 256 --resample 44.1 tmpfile.wav',' ',string];
        dos(cmd);
        for i=1:100,
            waitbar(i/100);
        end
        close(h2);
        delete('tmpfile.wav');
    elseif data.channel==2
        h2=waitbar(0,'Encoding and Saving in Progress... Please Wait');
        cmd=['lame --quiet -b 256 --resample 44.1 -m s tmpfile.wav',' ',string];
        dos(cmd);
        for i=1:100,
            waitbar(i/100);
        end
        close(h2);
        delete('tmpfile.wav');
    end
end
end

```

Questo case mostra le informazioni relative all'applicazione, utilizzando una help dialog.

```

case 'credits'
    h=helpdlg('This is the Chorus FX inside of Nova - Mod Environment Tools built by
Zambelli Cristian. For best performance, we suggest to turn off all background
application, in order to get maximum memory throughput by your system.','Information');

```

Questo case esegue il termine dell'applicazione, rilasciando l'handle della figura dalla memoria, inviando il comando *close*.

```

case 'close'
    close(gcf);
end

```

Terminata la valutazione del codice del sistema di Chorusing, valutiamo il codice del sistema di Flanger – Phaser, considerando che l'unica parte variabile è data dall'applicazione dell'algoritmo e che in questo sistema, solamente una voce viene ricavata e l'altra opportunamente sfasata.

```

%Function File esterno per i callback del flanger
%
%written by Zambelli Cristian
function fcnPFlanger(action)
global x data t md lfw1 lfw2 popup1 y h1 aml;
data=load('xchangedata.mat');
switch(action)
case 'retrieve'
    t=( [1:data.count]/data.samplerate);
    x=data.y;
    h=waitbar(0,'Retrieving the Original Signal...');
    for i=1:100,
        waitbar(i/100);
    end
    close(h);
    if data.channel==1
        if data.quantize==8
            z=(x-128)/128;
        elseif data.quantize==16
            z=x/32768;
        end
    elseif data.channel==2
        chan=findobj(gcf,'Tag','PopupMenu20');
        chan=get(chan,'Value');
        if chan==1
            x=data.lefty;
        elseif chan==2
            x=data.righty;
        end
        if data.quantize==8
            z=(x-128)/128;
        elseif data.quantize==16
            z=x/32768;
        end
    end
end
end

```

```

h1=figure('Name','Signal Plot','NumberTitle','off');
subplot(211);
plot(t,z);
title('Dry Signal');
ylabel('Normalized Values');
if data.channel==1
    legend('Monaural Signal');
elseif data.channel==2
    if chan==1
        legend('Left Channel');
    elseif chan==2
        legend('Right Channel');
    end
end
end
case 'playdry'
tr=data.count/data.samplerate;
inc=0.01;
texe=clock;soundsc(x,data.samplerate,data.quantize);e=etime(clock,texe);
texe2=clock;h=waitbar(0,'Playback time (Dry)...');e2=etime(clock,texe2);
for i=0:inc:(tr-e-e2),
    waitbar(i/(tr-e-e2));
    pause(inc);
end
close(h);
case 'playwet'
tr=data.count/data.samplerate;
inc=0.01;
texe=clock;sound(y,data.samplerate,data.quantize);e=etime(clock,texe);
texe2=clock;h=waitbar(0,'Playback time (Wet)...');e2=etime(clock,texe2);
for i=0:inc:(tr-e-e2),
    waitbar(i/(tr-e-e2));
    pause(inc);
end
close(h);

```

Le LFW che ammettono lo sfasamento sono quelle basate su un andamento di tipo sinusoidale.

```

case 'getvoicel'
popup1handle=findobj(gcf,'Tag','PopupMenu1');
popup1=get(popup1handle,'Value');
mdh=findobj(gcf,'Tag','EditText20');
md=ceil((str2num(get(mdh,'String'))/1000)*data.samplerate);
if popup1==1
    [filename,filepath]=uigetfile('*.lfo','Get LFW from which file?');
    path=[filepath];
    name=[filename];

```

```

fid=fopen(strcat(path,name),'r');
lftype=fscanf(fid,'%s\n%');
rsst=fscanf(fid,'%s\n%');
wall=fscanf(fid,'%s\n%');
rate=fscanf(fid,'%s\n%');
amp=fscanf(fid,'%s\n%');
phase=fscanf(fid,'%s\n%');
offset=fscanf(fid,'%s\n%');
damping=fscanf(fid,'%s\n%');
rectif=fscanf(fid,'%s\n%');
status=fclose(fid);
h=waitbar(0,'Gathering LFO Parameters. Please Wait...');
x=data.y;
n=2:length(x)+md;
aml=str2num(amp);
damp=str2num(damping);
r=str2num(rate);
rinv=1/r;
ww=2*pi/round(rinv*data.samplerate);
p=str2num(phase);
static=findobj(gcf,'Tag','StaticText21');
set(static,'String',rate);
static1=findobj(gcf,'Tag','StaticText31');
set(static1,'String',amp);
nph=findobj(gcf,'Tag','EditText3');
npl=str2num(get(nph,'String'));
np=(npl*2*pi)/360;
if strcmp(lftype,'sin')
    stati=findobj(gcf,'Tag','StaticText11');
    set(stati,'String','Sine');
    if strcmp(rsst,'1')
        g=(1-exp(-(n*damp)).*sin(ww.*n+p));
        q=(1-exp(-(n*damp)).*sin(ww.*n+(p+np)));
        a=int32(g);
        e=int32(q);
    elseif strcmp(rsst,'0')
        a=(1-exp(-(n*damp)).*sin(ww.*n+p));
        e=(1-exp(-(n*damp)).*sin(ww.*n+(p+np)));
    end
    if strcmp(wall,'1')
        b=a+0.1*sin(2*pi*50*t);
        c=e+0.1*sin(2*pi*50*t);
    elseif strcmp(wall,'0')
        b=a;
        c=e;
    end
    if strcmp(rectif,'1')

```

```

        lfw1=double(abs(b));
        lfw2=double(abs(c));
elseif strcmp(rectif,'0')
        lfw1=double(b);
        lfw2=double(c);
end
elseif strcmp(lfwtype,'cos')
        stati=findobj(gcf,'Tag','StaticText11');
        set(stati,'String','Cosine');
        if strcmp(rsst,'1')
                g=(1-exp(-(n*damp)).*cos(ww.*n+p));
                q=(1-exp(-(n*damp)).*cos(ww.*n+(p+np)));
                a=int32(g);
                e=int32(q);
        elseif strcmp(rsst,'0')
                a=(1-exp(-(n*damp)).*cos(ww.*n+p));
                e=(1-exp(-(n*damp)).*cos(ww.*n+(p+np)));
        end
        if strcmp(wall,'1')
                b=a+0.1*sin(2*pi*50*t);
                c=e+0.1*sin(2*pi*50*t);
        elseif strcmp(wall,'0')
                b=a;
                c=e;
        end
        if strcmp(rectif,'1')
                lfw1=double(abs(b));
                lfw2=double(abs(c));
        elseif strcmp(rectif,'0')
                lfw1=double(b);
                lfw2=double(c);
        end
elseif strcmp(lfwtype,'tri')
        stati=findobj(gcf,'Tag','StaticText11');
        set(stati,'String','Triangular');
        if strcmp(rsst,'1')
                g=(1-exp(-(n*damp)).*sawtooth(ww.*n,p+0.5));
                a=int32(g);
        elseif strcmp(rsst,'0')
                a=(1-exp(-(n*damp)).*sawtooth(ww.*n,p+0.5));
        end
        if strcmp(wall,'1')
                b=a+0.1*sin(2*pi*50*t);
        elseif strcmp(wall,'0')
                b=a;
        end
        if strcmp(rectif,'1')

```

```

        lfw1=double(abs(b));
        lfw2=lfw1;
elseif strcmp(rectif,'0')
        lfw1=double(b);
        lfw2=lfw1;
end
elseif strcmp(lfwtype,'squ')
        stati=findobj(gcf,'Tag','StaticText11');
        set(stati,'String','Square');
        if strcmp(rsst,'1')
                g=(1-exp(-(n*damp)).*square(ww.*n,p));
                a=int32(g);
        elseif strcmp(rsst,'0')
                a=(1-exp(-(n*damp)).*square(ww.*n,p));
        end
if strcmp(wall,'1')
        b=a+0.1*sin(2*pi*50*t);
elseif strcmp(wall,'0')
        b=a;
end
if strcmp(rectif,'1')
        lfw1=double(abs(b));
        lfw2=lfw1;
elseif strcmp(rectif,'0')
        lfw1=double(b);
        lfw2=lfw1;
end
elseif strcmp(lfwtype,'exp')
        h1=warndlg('Sorry but this is not a type of LFW usable with this kind of
algorithm','Warning!!!');
        uiwait(h1);
elseif strcmp(lfwtype,'saw')
        stati=findobj(gcf,'Tag','StaticText11');
        set(stati,'String','Sawtooth');
        if strcmp(rsst,'1')
                g=(1-exp(-(n*damp)).*sawtooth(ww.*n,p));
                a=int32(g);
        elseif strcmp(rsst,'0')
                a=(1-exp(-(n*damp)).*sawtooth(ww.*n,p));
        end
if strcmp(wall,'1')
        b=a+0.1*sin(2*pi*50*t);
elseif strcmp(wall,'0')
        b=a;
end
if strcmp(rectif,'1')
        lfw1=double(abs(b));

```

```

        lfw2=lfw1;
elseif strcmp(rectif,'0')
    lfw1=double(b);
    lfw2=lfw1;
end
elseif strcmp(lfwtype,'ran')
    stati=findobj(gcf,'Tag','StaticText11');
    set(stati,'String','Pseudo Random');
    if strcmp(rsst,'1')
        g=(1-exp(-(n*damp)).*sin(rand(1)*ww.*n+p));
        q=(1-exp(-(n*damp)).*sin(rand(1)*ww.*n+(p+np)));
        a=int32(g);
        e=int32(q);
    elseif strcmp(rsst,'0')
        a=(1-exp(-(n*damp)).*sin(rand(1)*ww.*n+p));
        e=(1-exp(-(n*damp)).*sin(rand(1)*ww.*n+(p+np)));
    end
    if strcmp(wall,'1')
        b=a+0.1*sin(2*pi*50*t);
        c=e+0.1*sin(2*pi*50*t);
    elseif strcmp(wall,'0')
        b=a;
        c=e;
    end
end
if strcmp(rectif,'1')
    lfw1=double(abs(b));
    lfw2=double(abs(c));
elseif strcmp(rectif,'0')
    lfw1=double(b);
    lfw2=double(c);
end
end
for i=1:100,
    waitbar(i/100);
end
close(h);
elseif popup1==2
    lfomaker;
elseif popup1==3
    static=findobj(gcf,'Tag','StaticText21');
    set(static,'String','Smart');
    static1=findobj(gcf,'Tag','StaticText31');
    set(static1,'String','Smart');
    stati=findobj(gcf,'Tag','StaticText11');
    set(stati,'String','Smart');
    if data.channel==1
        len=data.count;

```

```

elseif data.channel==2
    len=(data.count)/2;
end
assignin('base','maxdelays',md);
assignin('base','lengthfile',len);
assignin('base','samp',data.samplerate);
h=waitbar(0,'Passing data to SmartPlayer System');
save('xchangesmart');
for i=1:100,
    waitbar(i/100);
end
close(h);
smartplayer;
h3=warndlg('Please maintain this Dialog Opened (dont push OK) until you have
finished to use the SmartPlayer System (he has the control of the entire system
now!!!)', 'WARNING!!!');
uiwait(h3);
passed=load('xchangesmart.mat');
lfw1=passed.lfw;
lfw1=abs(lfw1)';
lfw2=lfw1;
aml=1;
end
case 'plotlfw'
h5=figure('Name','LFW Used Plot','NumberTitle','off');
checkh=findobj(gcf,'Tag','Checkbox20');
check=get(checkh,'Value');
if check==0
    t=(1:length(lfw1))/data.samplerate);
    plot(t,lfw1);
    ylabel('LFW Depth');
    title('LFW1');
end
if check==1
    t=(1:length(lfw1))/data.samplerate);
    subplot(211);
    plot(t,lfw1);
    ylabel('LFW Depth');
    title('LFW1');
    subplot(212);
    plot(t,lfw2);
    ylabel('LFW Depth');
    title('LFW2');
end
end

```

L'algoritmo di Flanger – Phaser, è simile al Chorus, solamente che le voci sono solo due e i tempi di ritardo sono ridotti al minimo, in modo da applicare il filtraggio comb. Da notare inoltre che il calcolo non avviene per n che parte da 1, ma in questo caso da 2, perché altrimenti non si sarebbe potuta implementare la soluzione del feedback, che richiede i valori di $y(n-1)$ ²⁴. Anche qui come nel Chorus, avviene la costruzione del campo stereofonico, imponendo uno sfasamento delle voci e un semicerchio perfettamente diviso in tre settori, in cui la voce centrale (che in questo caso è il segnale originale) è sempre diretta verso il centro e le altre due alle estremità.

```

case 'apply'
    if data.channel==2
        chanh=findobj(gcf,'Tag','PopupMenu20');
        chan=get(chanh,'Value');
        if chan==1
            x=data.lefty;
        elseif chan==2
            x=data.righty;
        end
    end
    mdh=findobj(gcf,'Tag','EditText20');
    md=ceil((str2num(get(mdh,'String'))/1000)*data.samplerate);
    viberatelh=findobj(gcf,'Tag','EditText1');
    viberatel=str2num(get(viberatelh,'String'));
    n=2:length(x)+md;
    r=round((str2num(get(mdh,'String'))/1000)*data.samplerate);
    h=waitbar(0,'Flanging in progress (this could take a lot). Please wait...');
    for i=1:100,
        waitbar(i/100);
    end
    z=zeros(md,1);
    m=max(abs(x));
    y=[z;x;z];
    checkh=findobj(gcf,'Tag','Checkbox20');
    check=get(checkh,'Value');
    if check==0
        if popup1==1
            b1=n+md-round((r/2)*lfw1);
        elseif popup1==3
            b1=n+md-round((r/2)*lfw1');
        end
        f=(1:length(b1))/data.samplerate;
        if viberatel==0
            g=ones(size(b1));

```

²⁴ $Y(0)$ è un indice proibito in Matlab, e genera un errore di run – time.

```

else
    g=sin(2*pi*viberate1*f);
end
vibel=reshape(g,length(b1),1);
fdepthh=findobj(gcf,'Tag','EditText2');
fdepth=str2num(get(fdepthh,'String'));
feedh=findobj(gcf,'Tag','EditText21');
feed=str2num(get(feedh,'String'));
subtrh=findobj(gcf,'Tag','Checkbox2');
subtr=get(subtrh,'Value');
if subtr==0
    y(n)=y(n+md)+fdepth.*(vibel.*(am1*y(b1)))+feed*y(n-1);
    m=m/max(abs(y));
    y=m*y;
elseif subtr==1
    y(n)=y(n+md)+fdepth.*(vibel.*(am1*y(b1)))-feed*y(n-1);
    m=m/max(abs(y));
    y=m*y;
end
elseif check==1
    if popup1==1
        b1=n+md-round((r/2)*lfw1);
        b2=n+md-round((r/2)*lfw2);
    elseif popup1==3
        b1=n+md-round((r/2)*lfw1');
        b2=n+md-round((r/2)*lfw2');
    end
f=(1:length(b1))/data.samplerate;
if viberate1==0
    g=ones(size(b1));
else
    g=sin(2*pi*viberate1*f);
end
vibel=reshape(g,length(b1),1);
fdepthh=findobj(gcf,'Tag','EditText2');
fdepth=str2num(get(fdepthh,'String'));
feedh=findobj(gcf,'Tag','EditText21');
feed=str2num(get(feedh,'String'));
subtrh=findobj(gcf,'Tag','Checkbox2');
subtr=get(subtrh,'Value');
bal0=[0.5 0.5];
bal1=[1 0];
bal2=[0 1];
if subtr==0

```

```

y=[y(n+md)*bal0+fdepth.*((vibel.*(am1*y(b1)))*bal1+(vibel.*(am1*y(b2)))*bal2)+feed*y(n-1)*bal0];

```

```

elseif subtr==1
    y=[y(n+md)*bal0+fdepth.*((vibel.*(am1*y(b1)))*bal1+(vibel.*(am1*y(b2)))*bal2)-
feed*y(n-1)*bal0];
end
end
close(h);
t=[(1:length(y))/data.samplerate];
if data.quantize==8
    y=(y-128)/128;
elseif data.quantize==16
    y=y/32768;
end
figure(h1);
subplot(212);
plot(t,y);
title('Wet Signal');
ylabel('Normalized Values');
if check==0
    legend('Monaural Flanged Signal');
elseif check==1
    legend('Left Flanged Signal','Right Flanged Signal');
end
case 'saveasf'
    button3=questdlg('Would you like to save this signal for further processing?','Save
file...','Microsoft PCM WAVE FORMAT','Motion Picture Expert Group (MPEG-1) Layer
3','No','No');
    if strcmp(button3,'Microsoft PCM WAVE FORMAT')
        [newfile,newpath] = uiputfile('*.wav','Save as...');
        newpath1=[newpath];
        newfile1=[newfile];
        h2=waitbar(0,'Saving in Progress... Please Wait');
        wavwrite(y,data.samplerate,data.quantize,strcat(newpath1,newfile1));
        for i=1:100,
            waitbar(i/100);
        end
        close(h2);
    elseif strcmp(button3,'Motion Picture Expert Group (MPEG-1) Layer 3')
        [newfile,newpath] = uiputfile('*.mp3','Save as...');
        newpath1=[newpath];
        newfile1=[newfile];
        wavwrite(y,data.samplerate,data.quantize,'tmpfile.wav');
        string=strcat(newpath1,newfile1,'.mp3');
        if data.channel==1
            h2=waitbar(0,'Encoding and Saving in Progress... Please Wait');
            cmd=['lame --quiet -b 256 --resample 44.1 tmpfile.wav',' ',string];
            dos(cmd);
            for i=1:100,

```

```
        waitbar(i/100);
    end
    close(h2);
    delete('tmpfile.wav');
elseif data.channel==2
    h2=waitbar(0,'Encoding and Saving in Progress... Please Wait');
    cmd=['lame --quiet -b 256 --resample 44.1 -m s tmpfile.wav',' ',string];
    dos(cmd);
    for i=1:100,
        waitbar(i/100);
    end
    close(h2);
    delete('tmpfile.wav');
end
end
case 'credits'
    h=helpdlg('This is the Flanger - Phaser FX inside of Nova - Mod Environment Tools
built by Zambelli Cristian. For best performance, we suggest to turn off all background
application, in order to get maximum memory throughput by your system.','Information');
case 'close'
    close(gcf);
end
```

Capitolo 6

Conclusioni del progetto

6.1 Conclusioni principali

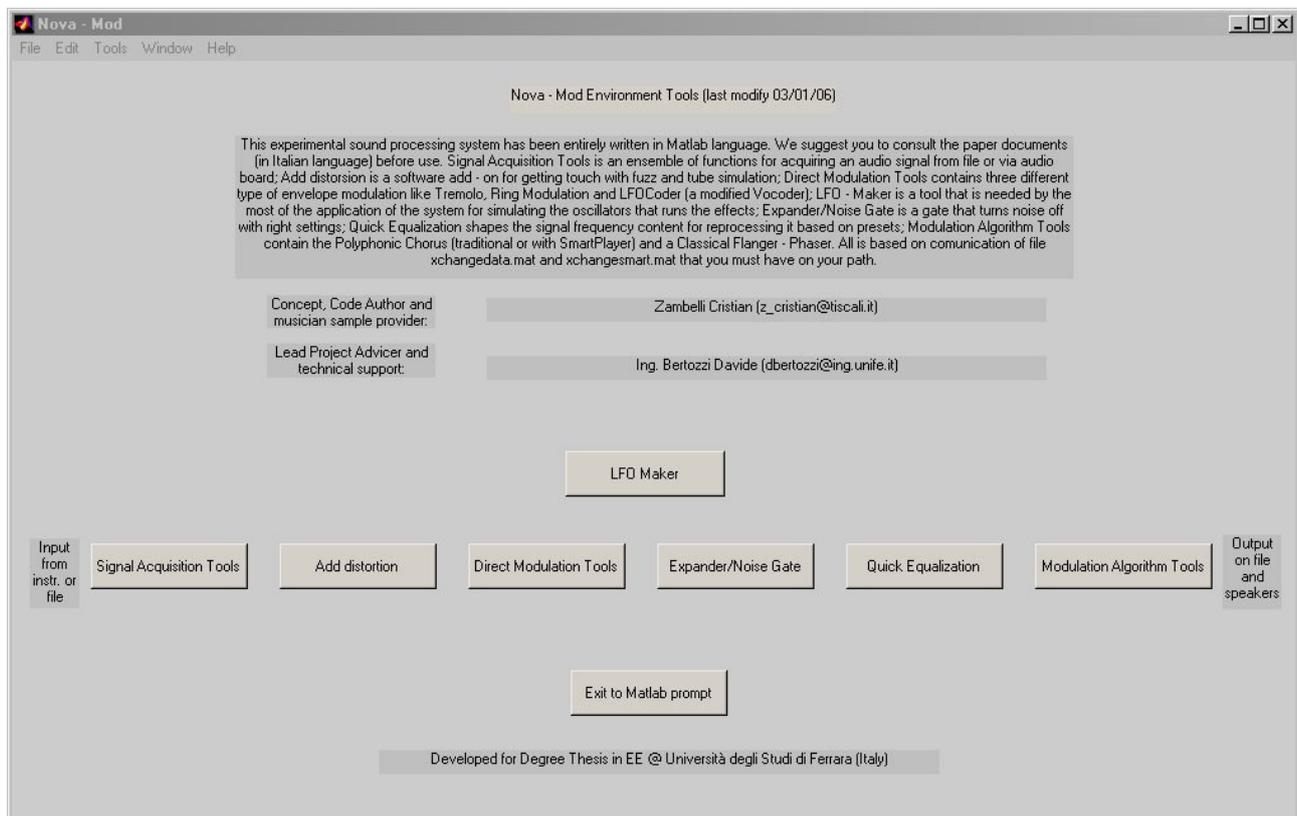


Figura 49: Schermata iniziale del software Nova - Mod Environment Tools

Dopo aver terminato le features di tutto il sistema Nova – Mod Environment Tools, è arrivato il momento di trarre delle conclusioni sul lavoro svolto. Ciò che si è voluto far emergere con questo lavoro, è stato il concepimento primario di un software per la manipolazione di un segnale audio, sia real – time acquired che file – based acquired, basato su un software math – oriented come Matlab, in modo da intensificare il legame sempre più imponente tra la matematica per l’elaborazione dei segnali e l’ingegneria del suono. Proprio lo stesso utilizzo di Matlab, ci ha permesso di utilizzare delle funzioni di processing avanzato come il filtraggio IIR ad esempio, che avrebbero richiesto molto più tempo di sviluppo in un ambiente compiled – type. Mentre da una parte però, si è riuscito ad abbattere la barriera della qualità degli algoritmi e delle implementazioni via software delle strategie architetturali alla base di ogni singolo componente del sistema Nova – Mod, dall’altra, ci siamo dovuti pesantemente scontrare con le limitazioni poste dall’ambiente di

sviluppo, a causa di una non perfetta gestione del sistema memoria – microprocessore (che ha richiesto notevoli sforzi di design per arginare pesanti sgravi nello speed behaviour) e di un discovery bug rate di Matlab esageratamente alto per un software di questa categoria (si pensi che per ogni esecuzione del software, venivano scoperti almeno 2 bug di Matlab, che rendevano inutilizzabile il codice, con la necessità di riprogettarlo ogni volta...). Per poter cercare di migliorare l'overall system speed, si è tentato di effettuare un porting (cross – compiling) degli script da linguaggio Matlab ad un linguaggio C/C++, ma a causa della ridotta compatibilità delle Graphics Library offerte dalla MathWorks e dal compilatore di Microsoft del Developer Studio .Net, ciò non è stato possibile. C'è da dire però, che anche con un operazione di questo tipo, non si sarebbe tratto particolare giovamento, a causa di operazioni matematiche di indicizzazione (come la riallocazione degli spazi lineari), che avrebbero richiesto lo stesso tempo di esecuzione dal microprocessore. In sintesi, questo software ha presentato qualche innovazione nell'audio processing, mantenendo però tutte le funzionalità classiche di un normale sistema di alta fascia, cercando di competere su tutti i fronti, perdendo o vincendo non importa, ma pur sempre competendo per cercare di migliorare se stesso... Come dice un famoso motto di un'azienda di memorie a semiconduttore:

...Never Stop Thinking

ed è quello che il software e chi lo ha realizzato sta continuando a fare.

6.2 Confronto con software di alta fascia Syntrillium Cool Edit Pro

6.2.1 Features supportate

Riportiamo in questa tabella, le features supportate dal sistema di processing audio Nova – Mod e dal software Cool Edit Pro, riuniti per categoria.

	Nova – Mod Environment Tools	Syntrillium Cool Edit Pro	Miglior software
Parte del software relativa all'acquisizione del segnale audio da processare	1. Real – Time Acquisition supportata con samplerate fino a 44.1 KHz e fino a 16 bit di quantizzazione	1. Real – Time Acquisition supportata con samplerate fino a 48 KHz ed emulazione di 32 bit di	1. Cool Edit Pro 2. Cool Edit Pro 3. Nova – Mod

	<ol style="list-style-type: none"> 2. By file acquisition supportata con file formato wav e mp3 3. PSD calcolata sul segnale con tutti i metodi possibili da Matlab a qualità superiore 4. Spettrogramma calcolato con ogni metodo di finestrazione a 256 punti 5. Scalatura delle unità di misura del plot del segnale da processare 	<p>quantizzazione</p> <ol style="list-style-type: none"> 2. By file acquisition con file supportati in molteplici modalità di formato 3. PSD calcolata con finestrazione standard di Welch 4. Spettrogramma calcolato con quasi tutti i metodi di finestrazione a punti indefiniti 5. Scalatura delle unità di misura del plot del segnale da processare 	<ol style="list-style-type: none"> 4. Tie 5. Tie
Parte del software relativa alla generazione delle LFW per i processi di modulazione	<ol style="list-style-type: none"> 1. Applicazione LFO Maker con file proprietari su cui sono salvati i risultati di ogni tipo di forma d'onda creabile, molteplici parametri di simulazione, tra cui rumore di alimentazione, digitalizzazione, rettificazione, ecc. 	<ol style="list-style-type: none"> 1. Non implementato 	<ol style="list-style-type: none"> 1. Nova – Mod
Parte del software relativa all'applicazione della modulazione diretta sul segnale audio	<ol style="list-style-type: none"> 1. Tremolo FX implementato con possibilità di scelta dei parametri di forma d'onda modulante con creazione proprietaria ed eliminazione dei sub – armonici 2. Ring Modulator implementato con le stesse funzionalità di parametrizzazione 	<ol style="list-style-type: none"> 1. Amplitude Modulation implementata senza possibilità di parametrizzazione delle forme d'onda modulanti e senza eliminazione dei sub – armonici 2. Non implementato 3. Implementazione di 	<ol style="list-style-type: none"> 1. Nova – Mod 2. Nova – Mod 3. Nova – Mod

	<p>relative al Tremolo FX</p> <p>3. LFOCoder FX una variante del VoCoder implementata con la tecnologia a tre LFO</p>	<p>un semplice VoCoder</p>	
<p>Parte del software relativa all'applicazione delle distorsioni sul segnale audio</p>	<p>1. Exponential Law Fuzz con scelta dei parametri di Gain e Level della distorsione</p> <p>2. Tube Simulator modeller di tubi termoionici per l'amplificazione e la distorsione di un segnale audio, con parametri di simulazione fedeli all'originale</p>	<p>1. Simulatore di distorsione a forma d'onda definibile via grafica e asimmetrizzazione della distorsione</p> <p>2. Non implementato</p>	<p>1. Cool Edit Pro</p> <p>2. Nova – Mod</p>
<p>Parte del software relativa all'applicazione del noise gating sul segnale audio</p>	<p>1. Gate con isteresi simulato con molteplici parametri riferiti ad un'unità di expanding presente sul mercato</p>	<p>1. Sistemi avanzati di eliminazione del rumore basati su gate clipping e click remover</p>	<p>1. Cool Edit Pro</p>
<p>Parte del software relativa all'applicazione dell'equalizzazione del segnale audio</p>	<p>1. Sistema di preset basato su alcune configurazioni classiche nel mondo dell'audio processing</p>	<p>1. Sistemi avanzati di equalizzazione basati su filtri matematici molteplici o su semplici filtri IIR</p>	<p>1. Cool Edit Pro</p>
<p>Parte del software relativa all'applicazione degli algoritmi di modulazione sul segnale audio</p>	<p>1. Real Polyphonic Chorus a 12 voci con scelta di parametri diversi per ogni voce e fino a 12 LFW diverse, mixing finale, ricreazione dell'ambiente stereofonico, independent processing per ogni canale di segnali</p>	<p>1. Chorus basato sulla pseudo – randomizzazione lineare di LFW e con un numero di voci indefinito, ma senza la specifica dei parametri</p> <p>2. Flanger con molteplici funzionalità e</p>	<p>1. Nova – Mod</p> <p>2. Nova – Mod</p> <p>3. Nova – Mod</p>

	<p>stereofonici</p> <p>2. Classical Flanger – Phaser con funzionalità di Stereo Phasing e di Feedbacking e scelta di LFW</p> <p>3. Sistema di simulazione del comportamento umano tramite l'applicazione di un modello che valuta le capacità di un esecutore</p>	<p>parametri definibili via utente, ma senza la scelta delle LFW</p> <p>3. Non implementato</p>	
--	---	---	--

Per 9 scores a 5 il software Nova – Mod dimostra di avere più features rispetto al sistema di Syntrillium Cool Edit Pro. C'è da dire però che, mentre Nova – Mod si è focalizzato su un lavoro più esteso dal punto di vista matematico e della esecuzione degli algoritmi, Cool Edit ha senza ombra di dubbio, un'estensione più commerciale, anche per la migliore grafica proposta nell'intero software. Proprio su questo punto, si vuole suggerire alla Syntrillium e anche ad altre software – house presenti nell'audio market, di trascurare a volte, l'effetto “luccichio da albero di natale” e di concentrarsi maggiormente sulla qualità e sull'aspetto tecnico dei loro algoritmi. Chiaramente questa vuole essere una critica costruttiva...

6.2.2 Speed Test caricamento dei file PCM

Riportiamo i dati ottenuti in forma grafica dal capitolo 3 sull'acquisizione dei file Microsoft PCM:

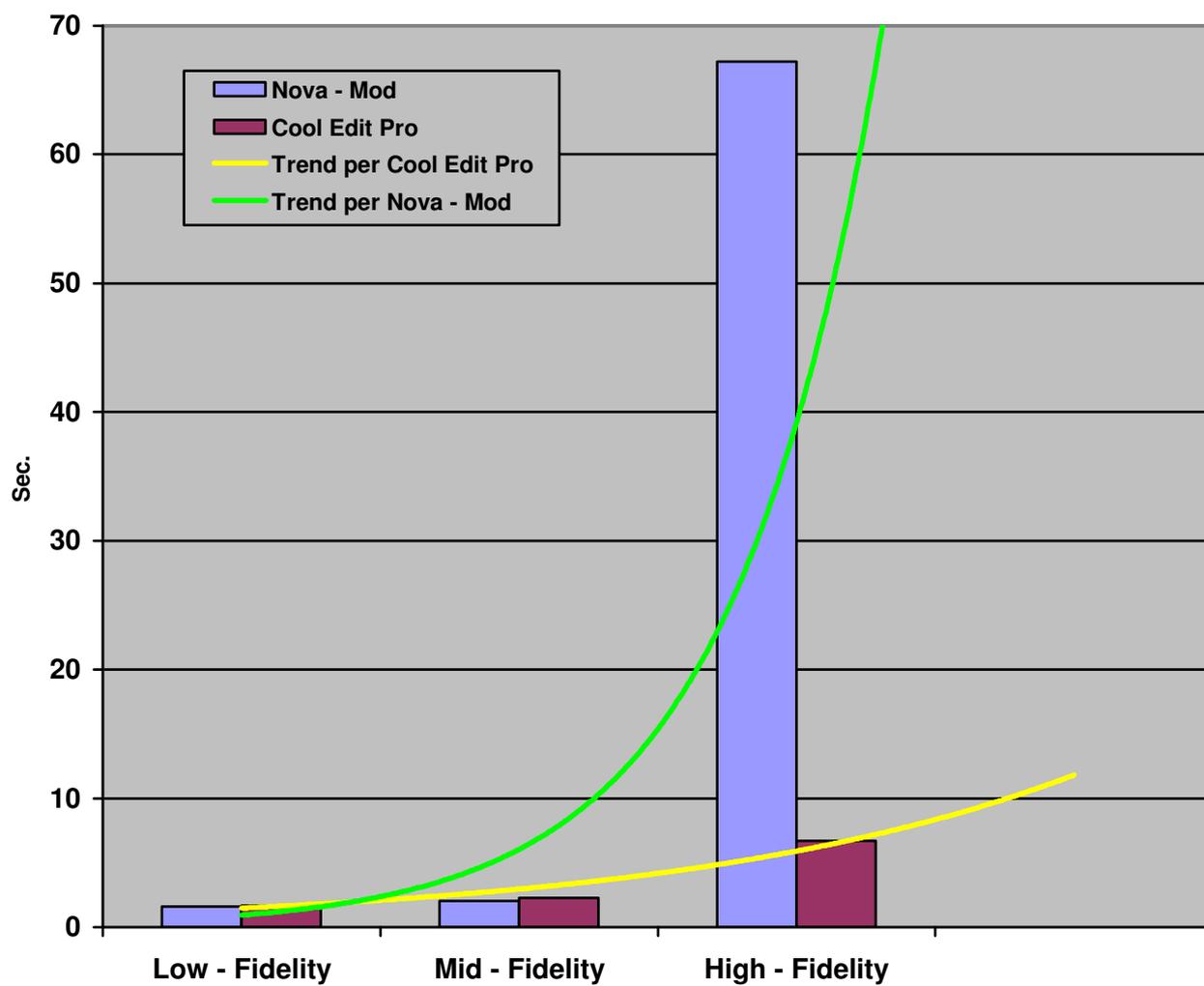


Figura 50: Speed Test velocità di caricamento PCM

6.2.3 Speed Test decodifica file Motion Picture Expert Group (MPEG-1) Layer 3

Analizziamo i risultati del confronto fra velocità di decodifica tra file MP3 e mostriamo i risultati nel seguente grafico:

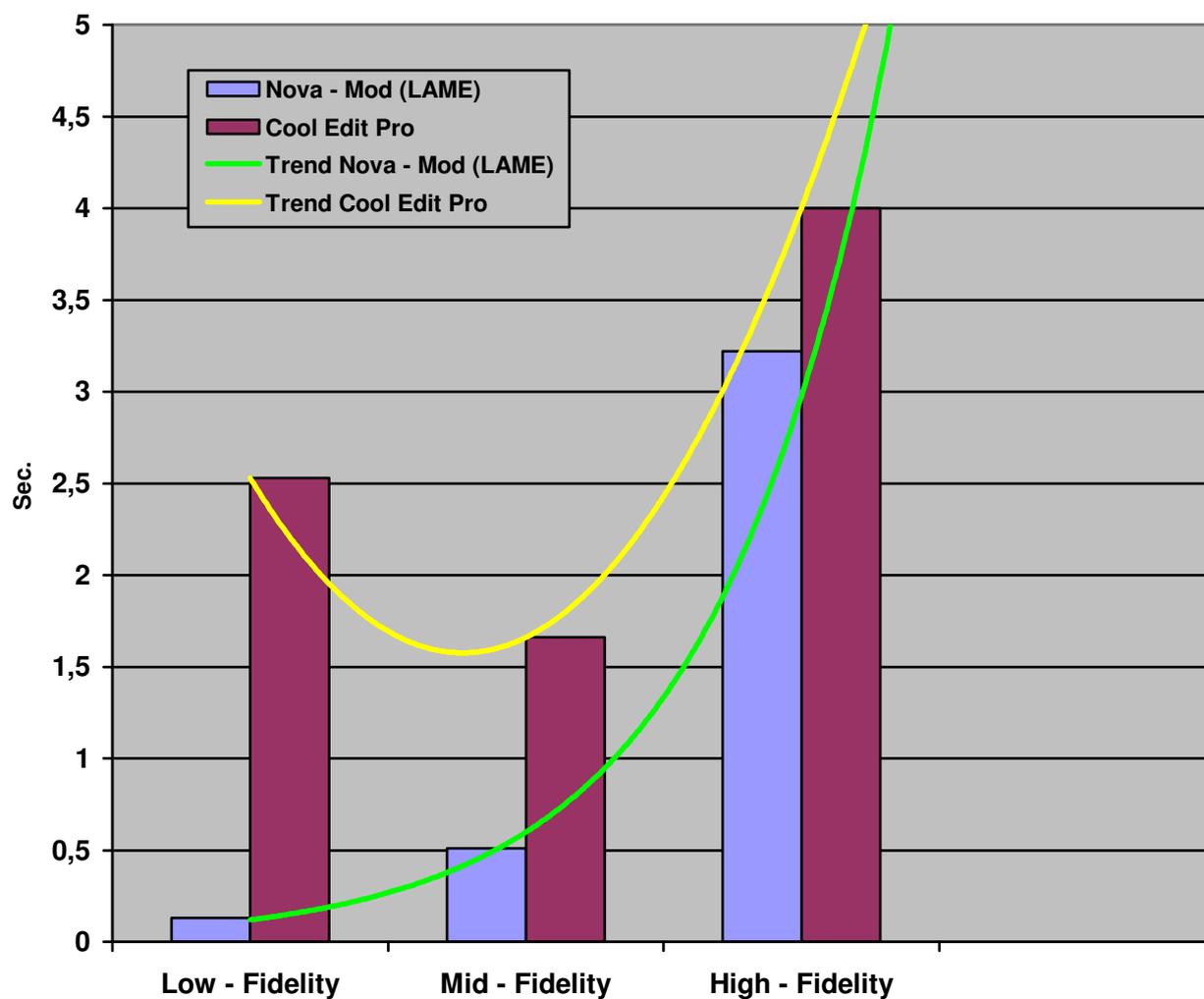


Figura 51: Speed Test sulla decodifica dei file in formato MP3

6.2.4 Speed Test computazione degli effetti

Valutiamo in questo grafico, la velocità di applicazione degli algoritmi, considerando che per ognuno di essi, si stia utilizzando il massimo carico di lavoro a cui possono essere sottoposti e che stiano processando lo stesso tipo di file:

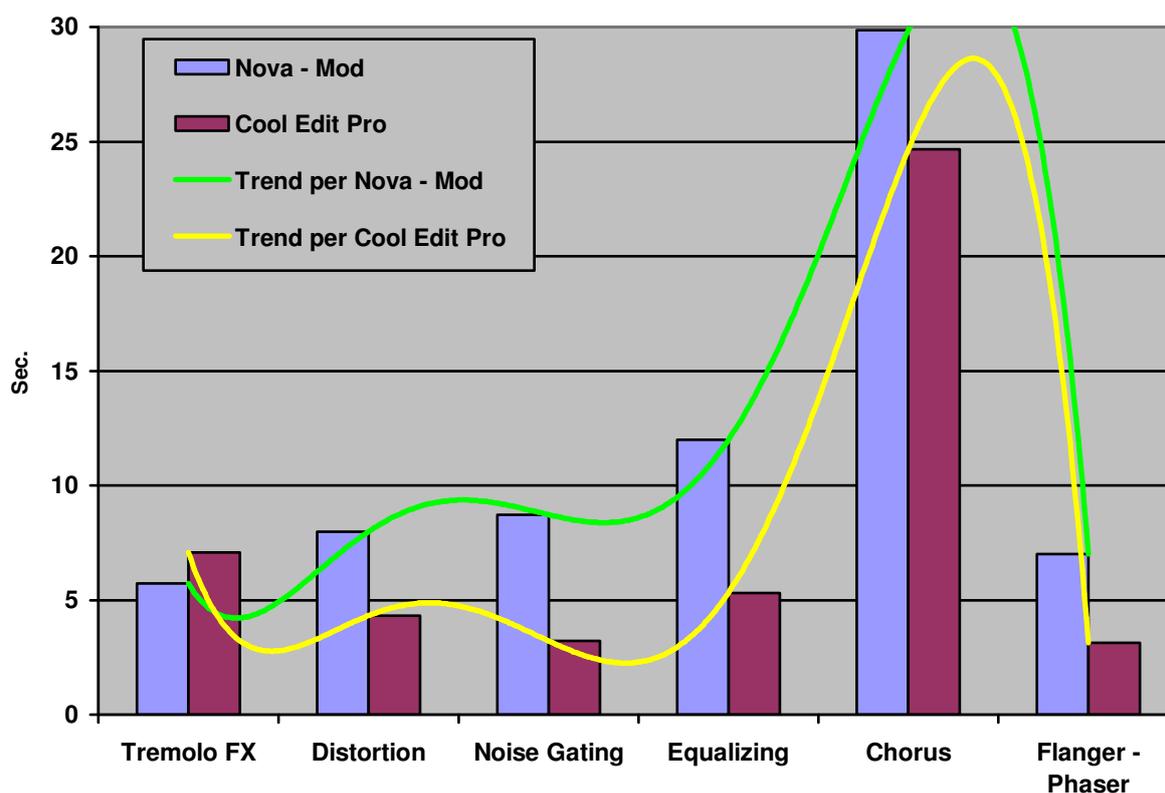


Figura 52: Speed Test su velocità di esecuzione degli effetti

6.3 User – Friendliness del sistema

Per valutare la capacità di comunicare con l'utente e di essere un sistema comprensibile anche a chi di audio processing non se ne intende, valutiamo la user – friendliness del software Nova – Mod Environment Tools, eseguendo 7 tests in forma anonima su persone con un diverso grado di conoscenza dei sistemi PC e dell'audio processing, valutando i risultati ottenuti:

- Soggetto 1 – Non conosce il PC e nemmeno l'audio processing: a questo soggetto abbiamo chiesto di segnalarci le eventuali differenze tra il software Nova – Mod e Cool Edit Pro e l'unica differenza principale riscontrata è che l'interfaccia grafica di Cool Edit Pro risulta

migliore e permette una migliore visualizzazione del segnale audio acquisito, con la sensazione che il software di Syntrillium offra più funzioni rispetto al nostro sviluppato; è stato chiesto di eseguire il playback di un file audio pre – acquisito e il tempo necessario per eseguire l'operazione è stato di 34.78 sec per il Nova – Mod contro i 2 min 3 secondi e 23 centesimi di secondo di Cool Edit Pro.

- Soggetto 2 – Media conoscenza del PC ma non conosce l'audio processing: anche a questo soggetto è stato chiesto di segnalarci eventuali differenze e l'unica cosa riscontrata sembra una diversa visualizzazione della waveform relativa al segnale acquisito; è stato chiesto di calcolare il valor medio statistico del segnale acquisito ma non trova la funzionalità su Cool Edit Pro, mentre ci si avvicina in Nova – Mod; è stato chiesto anche di localizzare dove si trova l'effetto di Flanger – Phaser, che è stato localizzato in 15,05 sec su Nova – Mod, ma non è stato trovato su Cool Edit Pro.
- Soggetto 3 – Ottima conoscenza del PC ma conosce a malapena l'audio processing: data l'ottima padronanza con i sistemi a microprocessore del soggetto in esame, abbiamo chiesto ad esso, di effettuare un test sulla verifica delle capacità computazionali del software, a livello di impegno del microprocessore e della memoria RAM; le sue valutazioni sono state piuttosto negative per quanto riguarda l'utilizzo delle risorse della memoria RAM (dal momento che si utilizza un workspace globale di Matlab), mentre si sono dimostrate migliori le valutazioni sull'utilizzo del microprocessore, notando che anche con un multi – tasking molto pesante (molti processi concorrenti attivi), il software Nova – Mod riesce, al contrario di Cool Edit Pro, a mantenere discrete prestazioni.
- Soggetto 4 – Media conoscenza del PC e discreta conoscenza dell'audio processing: questo soggetto appartiene alla categoria di media conoscenza del settore, per cui gli abbiamo chiesto di applicare un semplice effetto di Chorus al segnale audio pre – acquisito, ed il risultato è stato di soli 5,64 sec per individuare dove si trovava l'effetto, contro i 10,34 di Cool Edit Pro.
- Soggetto 5 – Media conoscenza del PC e ottima conoscenza dell'audio processing: essendo questo soggetto, un esperto di audio processing, abbiamo chiesto ad esso, di applicare una catena di effetti dall'inizio alla fine; la sua valutazione è stata negativa, in quanto il software

Cool Edit Pro, prevede l'applicazione degli effetti sullo stesso segnale, senza dover salvare e riaprire ogni volta un nuovo file.

- Soggetto 6 – Ottima conoscenza del PC e ottima conoscenza dell'audio processing: abbiamo chiesto a questo soggetto di eseguire delle operazioni un po' più complesse delle normali valutate per gli altri testers; abbiamo chiesto infatti di calcolare il periodogramma secondo la metodologia MUSIC e si è notato che il tempo impiegato per trovare questa funzione in Nova – Mod è stato di soli 7,45 sec, mentre in Cool Edit Pro questa funzionalità non è stata trovata ed anzi classificata addirittura inesistente; abbiamo chiesto una valutazione del software Nova – Mod, ed anche se ci sono alcuni punti da migliorare notevolmente, il risultato è del tutto comparabile ai normali software per audio processing.
- Soggetto 7 – Professionista del settore: a questo soggetto è parso inutile chiedere l'esecuzione di test specifici, per questo abbiamo solamente valutato una sua opinione sullo sviluppo del software; “Il software presenta alcuni bug da risolvere specialmente per la parte relativa all'acquisizione dei file audio e per quanto riguarda l'applicazione degli algoritmi di modulazione (peraltro molto ben implementati), ma niente che possa essere di impedimento al funzionamento del software stesso, piuttosto si tratta di qualcosa che va a scapito della user – friendliness del software. Discreto software e consigliabile dal punto di vista didattico”, queste sono state le sue parole.

6.4 Tempi di applicazione di un processing completo

Un buon indicatore di qualità di un software di qualsiasi tipo, è la possibilità di svolgere l'operazione più complessa (o quasi), nel minor tempo possibile, evitando i tempi morti dovuti alla unfriendliness del sistema (es. impiego 20 minuti per trovare il pulsante che mi permette di salvare il file, ecc.) e alla scarsa documentazione dello stesso. Per questo motivo, eseguiamo ora 3 esperimenti, che consistono nell'applicazione di un processing completo su un segnale audio qualsiasi, misurando i tempi effettivi di applicazione dell'effetto complessivo finale. Questo ci aiuterà a capire se il nostro software è correttamente progettato o meno. I tests verranno eseguiti e da noi valutati in maniera del tutto imparziale:

- Test 1: Acquisizione di un file MP3 High – Fidelity da 16 sec. stereofonico e applicazione di un tremolo con LFW a 5 Hz, distorsione, noise gating e flanger con LFW a 0.6 Hz senza

Feedback e senza ricostruzione del segnale stereofonico: eseguendo il test con il software Nova – Mod, il tempo complessivo impiegato è di 5 min 0 sec e 59 centesimi di secondo, con una progressione del tempo di applicazione, quando si applicano gli effetti di noise gating e di distorsione (infatti si tratta di effetti non particolarmente ottimizzati); eseguendo il test con il software Cool Edit Pro, il tempo complessivo è invece di 4 min 13 sec e 15 centesimi di secondo, con un andamento pressoché lineare dei tempi di applicazione di ogni singolo effetto.

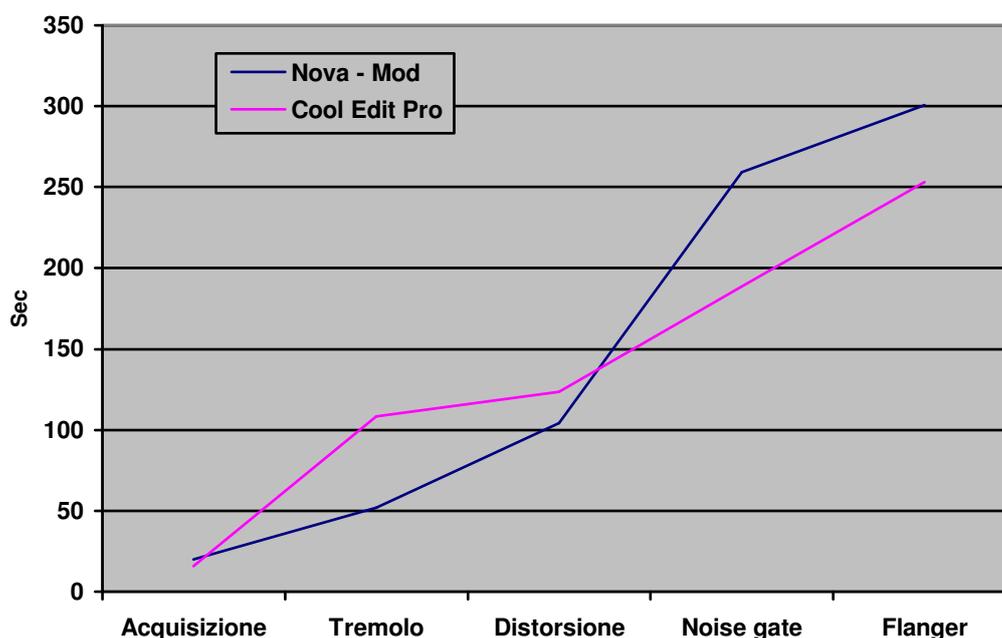


Figura 53: Grafico del Test 1 sui tempi di applicazione

C'è da fare una considerazione importante su questo grafico, ed è che i risultati relativi a Cool Edit Pro, sono un po' falsati dal fatto che non esistendo tutte le funzionalità implementate in Nova – Mod (quale ad esempio la creazione della LFW modulante), ci siamo dovuti accontentare di valori fasulli per i processi di Tremolo e di Noise Gate.

- Test 2: Acquisizione di un real – time signal Mid – Fidelity da 15 sec. monaurale e applicazione di una distorsione di tipo fuzzed, equalizzazione e chorus a 7 voci con LFW a 0.6 Hz con ricostruzione del campo stereofonico: eseguendo il test con il software Nova – Mod, il tempo complessivo è di 3 min 7 sec e 46 centesimi di secondo; eseguendo il test con il software Cool Edit Pro, il tempo complessivo è di 3 min 9 sec e 46 centesimi di secondo.

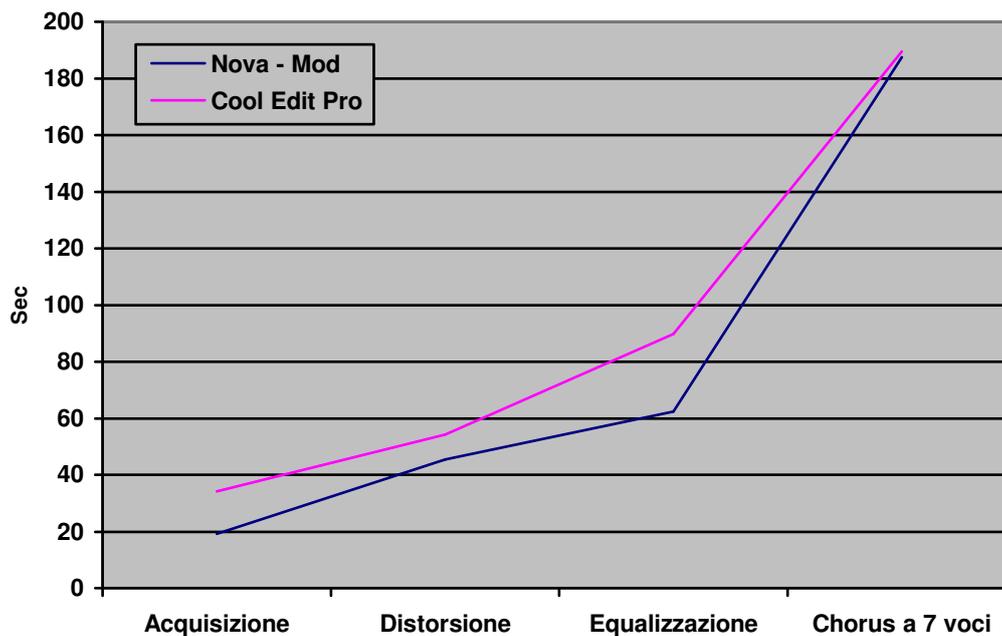


Figura 54: Grafico del Test 2 sui tempi di applicazione

- Test 3: Acquisizione di un file PCM High – Fidelity da 25 sec monoaurale con applicazione di tutta la catena di effetti presenti in Nova – Mod: distorsione, tremolo, LFOCoding, noise gating, equalizzazione, chorus a 12 voci e flanger – phaser stereofonico con negative feedback. Il tempo complessivo di applicazione per il software Nova – Mod è stato di 6 min 54 sec e 64 centesimi di secondo, mentre per il software Cool Edit Pro è stato di 6 min 38 sec e 24 centesimi di secondo.

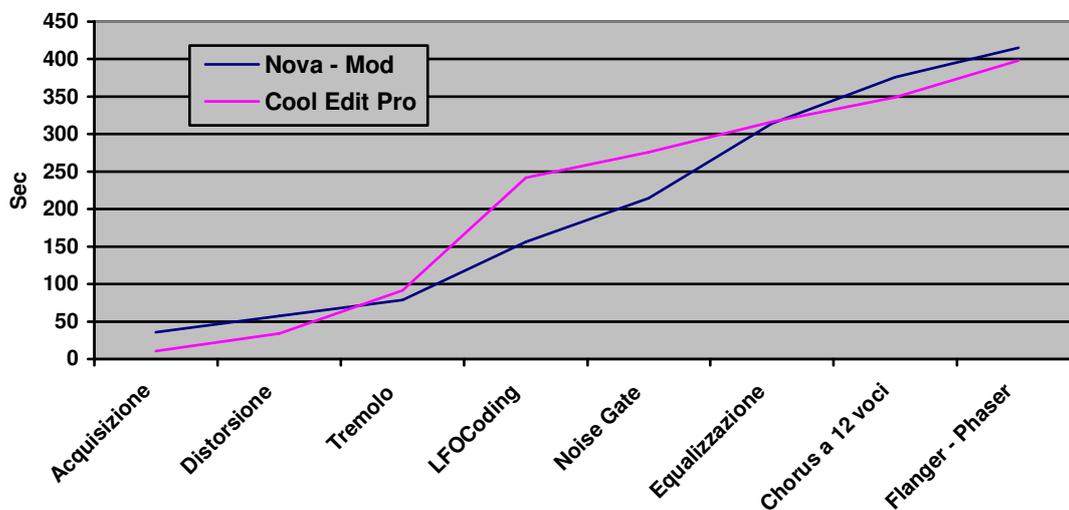


Figura 55: Grafico del Test 3 sui tempi di applicazione

Appendice A

We need Distortion!

A.1 C'è distorsione e distorsione ...

Cos'è un'unità di overdrive, distorsione, fuzz? La risposta a questa domanda richiede una distinzione tra i singoli termini che abbiamo precedentemente citato: il termine overdrive è utilizzato per definire una amplificazione del segnale che tende a “saturare” in maniera molto ridotta l'uscita di un amplificatore audio, ed è un suono che negli anni '60 si otteneva tipicamente andando a posizionare il volume di un amplificatore al massimo, in modo che le valvole dello stadio di uscita, non riuscendo a gestire un segnale di ampiezza elevata, lo distorcevano leggermente (la stessa cosa la si ottiene con un normale riproduttore stereo alzando al massimo il volume); il termine distorsione invece si riferisce ad un suono più marcato e più duro, con un elevato contenuto di armoniche di ordine dispari nello spettro del segnale, le quali aggiungono al segnale un senso di impurità, questo suono era tipico dell'utilizzo di diodi al silicio nello stadio di uscita, i quali “tagliavano” in maniera netta il segnale sui picchi più elevati); il termine fuzz invece, si riferisce al suono di distorsione più “sporco”, il quale contiene molte armoniche di ordine dispari e fenomeni di distorsione di intermodulazione che intervegono sul segnale, donando al suono di uscita una tonalità simile a quella che si otteneva negli anni '70 andando a “tagliare” con un rasoio o una lama qualsiasi l'altoparlante dell'amplificatore. Lo scopo quindi di unità di questo tipo è quello di “sporcare” il segnale audio per dargli una caratteristica più aggressiva o più marcata a seconda del genere musicale. Ma come lavora una distorsione dal punto di vista “matematico – elettronico”? Bene, supponiamo di prendere il suono più puro possibile esistente in natura, la sinusoide:

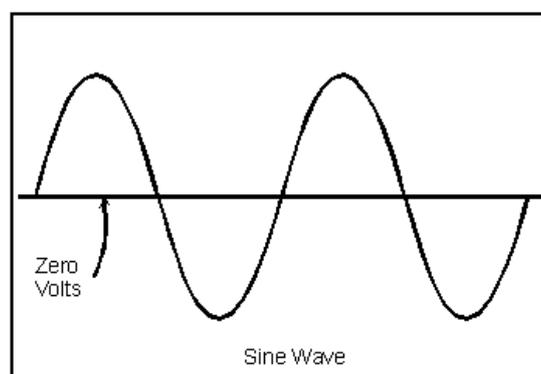


Figura 56: Sinusoide pura senza DC Offset

In realtà nessuno strumento musicale, ne tantomeno la voce umana, riusciranno mai a produrre un'onda di questo tipo e si tende più ad avvicinarsi ad onde di forma triangolare più che sinusoidali. Nella musica, il significato di un'onda sta in quelli che vengono definiti armonici fondamentali e parziali. L'armonica fondamentale è una sola, come dice il termine stesso, e da la caratteristica sonora basandosi sulla frequenza che essa rappresenta come ad esempio il LA a 440 Hz. Risulterà evidente che il LA a 880 Hz sarà un "ottava" più alto rispetto alla nota precedente, e così via moltiplicando per un numero N intero rispetto alla fondamentale. Una nota quindi sarà il risultato della somma di questi singoli armonici parziali. Parleremo di Distorsione Armonica Totale (THD) quando consideriamo dalla forma d'onda finale, i singoli contributi apportati dai vari armonici parziali e considerando le varie operazioni di attenuazione e di "taglio" delle varie componenti armoniche spettrali del segnale. Dal punto di vista elettronico allora, possiamo fare delle considerazioni sulla suddivisione dei termini overdrive, distorsione, fuzz. Se andiamo a tagliare (operazione di clipping della sinusoide) i punti di ampiezza più elevati nella sinusoide, otteniamo l'introduzione nello spettro del segnale di armonici parziali, ed in base al grado con cui andiamo a tagliare la sinusoide otteniamo un diverso tipo di suono distorto in uscita. Quando la sinusoide viene "clippata" come mostrato in figura, allora si parla di un suono di tipo overdrive, ovvero un suono morbido che ha una caratteristica tonale calda ma allo stesso tempo suona in maniera pressoché distorta:

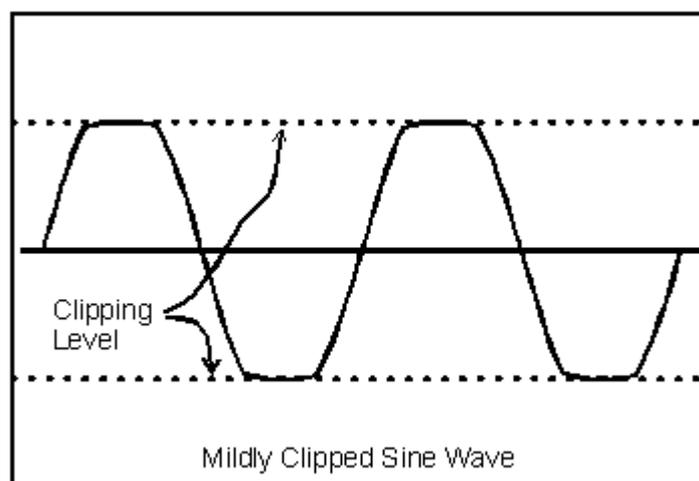


Figura 57: Sinusoide tagliata in maniera lieve

Se invece andiamo a "clippare" il segnale in maniera molto incisiva, otteniamo un suono di distorsione pura comprendente molti più armonici di un singolo overdrive e che da in uscita un suono secco e travolgente tipico dell' heavy metal:

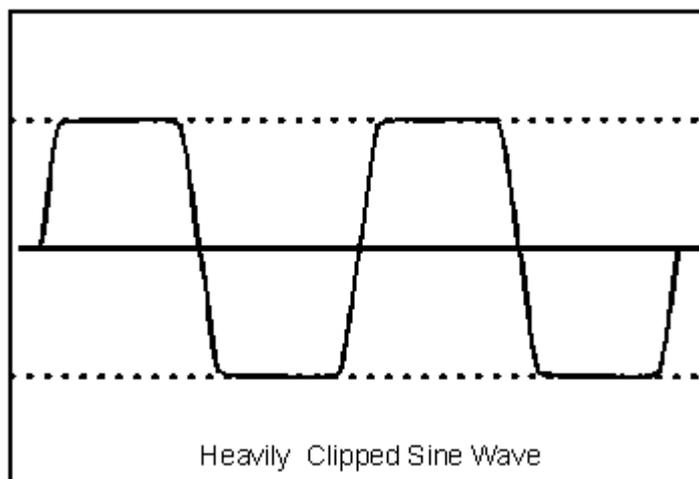


Figura 58: Sinusoide fortemente tagliata

Per ottenere invece il caratteristico suono del fuzz è richiesta un'elaborazione maggiore a livello matematico sul segnale, anche perché si deve amplificare il segnale a livelli elevatissimi facendolo assomigliare sempre di più ad un'onda quadra piuttosto che ad una sinusoidale. Bisogna inoltre porre l'attenzione su un fenomeno completamente anti-musicale che prende il nome di distorsione di intermodulazione che possiamo spiegare come segue: se prendiamo una sinusoide a 440 Hz e una a 880 Hz intermodolandole tra di loro otteniamo quattro armonici che sono 440 Hz – 880 Hz – 440 Hz (la differenza delle prime due note) – 1320 Hz (che è tre volte la fondamentale) e fino a qui andrebbe tutto bene perché si otterrebbe un suono piuttosto “puro”. Tuttavia se prendessimo una nota a 440 Hz, la sua ottava e una nota posizionata una “terza” (riferendosi alla scala diatonica maggiore) sopra a 1467 Hz, si ottengono le note originali ma con l'aggiunta di componenti armoniche a 1026 Hz e 1907 Hz, che suonano in maniera molto stonata e impacciata. Ecco il motivo per il quale bisogna porre attenzione nella realizzazione di un'unità fuzz.

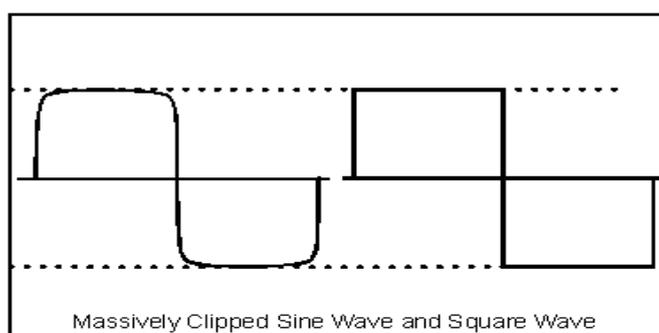


Figura 59: Metodo di rappresentazione di una sinusoide tramite onda quadra, per ottenere il fuzz

A.2 Metodi di realizzazione degli effetti di distorsione “analog based”

Per la realizzazione dell’overdrive – distorsione – fuzz possono essere utilizzate differenti configurazioni circuitali che oltre ad amplificare il segnale effettuano l’operazione di “clipping” del contenuto informativo, ma sostanzialmente ne riassumiamo qui alcune: configurazione a Common Cathode Triode (triodo a catodo comune), nella quale il clipping del segnale avviene in maniera asimmetrica, per dare al segnale di uscita un suono tipicamente più “impastato”, ma allo stesso tempo morbido, tanto è che questa configurazione prende anche il nome di “Preamp Distortion”; configurazione Double Ended Penthode (pentodo o tubo termoionico connesso a massa in push - pull), ed è una distorsione tipicamente più violenta che aggiunge al segnale un numero elevato di armoniche dispari tra cui tipicamente la terza, purtroppo però hanno la proprietà di spostare il punto di polarizzazione della valvola quando sono troppo eccitate dal segnale e questo causa compressione sonora variando l’attacco e il rilascio del segnale audio, alterandone totalmente la dinamica; configurazione Voltage Feedback/Biased Bipolar (retroazione in tensione o transistor bipolare a cambio di punto di riposo), che causa una aggiunta di due prominenti armoniche come la terza e la quinta, che inseriscono nel segnale un suono di tipo Fuzz tipico degli amplificatori a cono tagliati; configurazione Back to Back Diodes (diodi schiena a schiena), nella quale i diodi sono connessi in controfase l’uno con l’altro, in modo da tagliare i picchi del segnale in maniera uniforme, con la proprietà che se i diodi sono al germanio il suono risulterà molto morbido tipo overdrive, se invece i diodi sono al silicio il suono sarà tipicamente più aggressivo tipo distorsione. Ovviamente questi sono solo alcuni dei molti e molti metodi utilizzati per avere una buona distorsione.

A.3 Simulazione con Matlab

In aggiunta alle funzionalità dell’environment Nova – Mod Tools, inseriamo un sistema di simulazione di distorsioni da utilizzare con i segnali provenienti dal Signal Acquisition System.

- Exponential Law Fuzz: questo sistema, simula il comportamento di un’unità per la distorsione di tipo Fuzz, molto in voga negli anni ’70, basandosi su una legge di tipo esponenziale. L’applicazione permette di stabilire il guadagno introdotto sul segnale e il rate of mixing finale, per stabilire la quantità di segnale Dry e di segnale Wet in uscita. Il controllo Gain, ha la funzione di definire il “carattere” della distorsione: per maggiori valori di Gain, si ottiene una distorsione molto sporca, confusa e ricca di frequenze medie (400 –

800 Hz), tipica degli stadi a valvole degli amplificatori Marshall in clipping; per valori di Gain medi si ottiene un suono distorto, ma adatto ad ogni genere musicale, essendo il suo carattere non troppo incisivo; per valori di Gain bassi, si ottiene in uscita un suono di overdrive naturale, molto caldo e adatto per parti molto soft

```
%Function File esterno per i callback del fuzz
%
%written by Zambelli Cristian
function fcnFuzz(action)
global x data t h1 y r;
data=load('xchangedata.mat');
switch(action)
case 'retrieve'
    t=( [1:data.count]/data.samplerate);
    x=data.y;
    h=waitbar(0,'Preparing the Dry Signal...');
    for i=1:100,
        waitbar(i/100);
    end
    close(h);
    if data.channel==1
        if data.quantize==8
            z=(x-128)/128;
        elseif data.quantize==16
            z=x/32768;
        end
    elseif data.channel==2
        if data.quantize==8
            z=(x-128)/128;
        elseif data.quantize==16
            z=x/32768;
        end
    end
    h1=figure('Name','Signal Plot','NumberTitle','off');
    subplot(211);
    plot(t,z);
    title('Dry Signal');
    ylabel('Normalized Values');
    if data.channel==1
        legend('Monaural Signal');
    elseif data.channel==2
        legend('Left Channel','Right Channel');
    end
case 'apply'
    h=waitbar(0,'Applying Fuzz. Get ready to playback...');
```

```

slider1h=findobj(gcf,'Tag','Slider1');
sli1=get(slider1h,'Value');
slider2h=findobj(gcf,'Tag','Slider2');
sli2=get(slider2h,'Value');
if data.channel==1
    q=x*(sli1*200)/max(abs(x));
    z=sign(-q).*(1-exp(sign(-q).*q));
    y=sli2*z*max(abs(x))/max(abs(z))+(1-sli2)*x;
    y=y*max(abs(x))/max(abs(y));
elseif data.channel==2
    q1=(x(:,1))*(sli1*200)/max(abs(x(:,1)));
    z1=sign(-q1).*(1-exp(sign(-q1).*q1));
    y1=sli2*z1*max(abs(x(:,1)))/max(abs(z1))+(1-sli2)*(x(:,1));
    y1=y1*max(abs(x(:,1)))/max(abs(y1));
    q2=(x(:,2))*(sli1*200)/max(abs(x(:,2)));
    z2=sign(-q2).*(1-exp(sign(-q2).*q2));
    y2=sli2*z2*max(abs(x(:,2)))/max(abs(z2))+(1-sli2)*(x(:,2));
    y2=y2*max(abs(x(:,2)))/max(abs(y2));
    y=[y1 y2];
end
for i=1:100,
    waitbar(i/100);
end
close(h);
if data.channel==1
if data.quantize==8
    r=(y-128)/128;
elseif data.quantize==16
    r=y/32768;
end
elseif data.channel==2
    if data.quantize==8
        r=(y-128)/128;
    elseif data.quantize==16
        r=y/32768;
    end
end
figure(h1);
subplot(212);
plot(t,r);
title('Wet Signal');
ylabel('Normalized Values');
if data.channel==1
    legend('Monaural Signal');
elseif data.channel==2
    legend('Left Channel','Right Channel');
end

```

```

tr=data.count/data.samplerate;
inc=0.01;
texe=clock;soundsc(y,data.samplerate,data.quantize);e=etime(clock,texe);
texe2=clock;h=waitbar(0,'Playback time...');e2=etime(clock,texe2);
for i=0:inc:(tr-e-e2),
    waitbar(i/(tr-e-e2));
    pause(inc);
end
close(h);
case 'saveasf'
    button3=questdlg('Would you like to save this signal for further processing?','Save
file...','Microsoft PCM WAVE FORMAT','Motion Picture Expert Group (MPEG-1) Layer
3','No','No');
    if strcmp(button3,'Microsoft PCM WAVE FORMAT')
        [newfile,newpath] = uinputfile('*.wav','Save as...');
        newpath1=[newpath];
        newfile1=[newfile];
        h2=waitbar(0,'Saving in Progress... Please Wait');
        wavwrite(r,data.samplerate,data.quantize, strcat(newpath1,newfile1));
        for i=1:100,
            waitbar(i/100);
        end
        close(h2);
    elseif strcmp(button3,'Motion Picture Expert Group (MPEG-1) Layer 3')
        [newfile,newpath] = uinputfile('*.mp3','Save as...');
        newpath1=[newpath];
        newfile1=[newfile];
        wavwrite(r,data.samplerate,data.quantize,'tmpfile.wav');
        string=strcat(newpath1,newfile1,'.mp3');
        if data.channel==1
            h2=waitbar(0,'Encoding and Saving in Progress... Please Wait');
            cmd=['lame --quiet -b 256 --resample 44.1 tmpfile.wav',' ',string];
            dos(cmd);
            for i=1:100,
                waitbar(i/100);
            end
            close(h2);
            delete('tmpfile.wav');
        elseif data.channel==2
            h2=waitbar(0,'Encoding and Saving in Progress... Please Wait');
            cmd=['lame --quiet -b 256 --resample 44.1 -m s tmpfile.wav',' ',string];
            dos(cmd);
            for i=1:100,
                waitbar(i/100);
            end
            close(h2);
            delete('tmpfile.wav');
        end
    end
end

```

```

        end
    end
case 'close'
    close(gcf);
end

```

- Tube Simulator: questo sistema permette la simulazione di uno stadio di amplificazione del segnale, interamente valvolare. I parametri da settare sono molteplici e rispecchiano fedelmente il comportamento di qualsiasi valvola si voglia simulare, da una EL34 a una KT88. Il Gain stabilisce il guadagno introdotto dalla valvola sul segnale, il Character stabilisce il “carattere” della valvola e mi dice se questa è una valvola da distorsione potente o da semplice overdrive, il rate of mixing mi dà una proporzione del segnale di uscita Wet, miscelato a quello Dry; il Q – Point mi definisce il punto di polarizzazione della valvola e più i valori sono negativi più lineare sarà il comportamento della funzione di trasferimento del componente; la posizione del polo del filtro passa – alto simula la capacità della valvola di rimuovere la componente DC dal segnale e deve essere un valore prossimo a 1 in modulo; la posizione del polo del filtro passa – basso simula le capacità parassite della valvola, in grado di degradare il segnale. Questa simulazione permette una notevole combinazione di suoni, dall’overdrive naturale ad una distorsione incisiva, ma sempre tenendo conto della caratteristica di “morbidezza” del suono, introdotta da una valvola

```

%Function File esterno per i callback del tube simulator
%
%written by Zambelli Cristian
function fcnTubesim(action)
global x data t h1 y r;
data=load('xchangedata.mat');
switch(action)
case 'retrieve'
    t=( [1:data.count]/data.samplerate);
    x=data.y;
    h=waitbar(0,'Preparing the Dry Signal...');
    for i=1:100,
        waitbar(i/100);
    end
    close(h);
    if data.channel==1
    if data.quantize==8
        z=(x-128)/128;
    elseif data.quantize==16
        z=x/32768;

```



```

    end;
end;
y=sli3*z*max(abs(x))/max(abs(z))+(1-sli3)*x;
y=y*max(abs(x))/max(abs(y));
y=filter([1 -2 1],[1 -2*edit2 edit2^2],y);
y=filter([1-edit3],[1 -edit3],y);
elseif data.channel==2
q1=x(:,1)*sli1/max(abs(x(:,1)));
if edit1==0
    z1=q1./(1-exp(-sli2*q));
    for i=1:length(q)
        if q1(i)==edit1
            z1(i)=1/sli2;
        end;
    end;
else
    z1=(q1-edit1)./(1-exp(-sli2*(q1-edit1)))+edit1/(1-exp(sli2*edit1));
    for i=1:length(q1)
        if q1(i)==edit1
            z1(i)=1/sli2+edit1/(1-exp(sli2*edit1));
        end;
    end;
end;
y1=sli3*z1*max(abs(x(:,1)))/max(abs(z1))+(1-sli3)*x(:,1);
y1=y1*max(abs(x(:,1)))/max(abs(y1));
y1=filter([1 -2 1],[1 -2*edit2 edit2^2],y1);
y1=filter([1-edit3],[1 -edit3],y1);
q2=x(:,2)*sli1/max(abs(x(:,2)));
if edit1==0
    z2=q2./(1-exp(-sli2*q2));
    for i=1:length(q2)
        if q2(i)==edit1
            z2(i)=1/sli2;
        end;
    end;
else
    z2=(q2-edit1)./(1-exp(-sli2*(q2-edit1)))+edit1/(1-exp(sli2*edit1));
    for i=1:length(q2)
        if q2(i)==edit1
            z2(i)=1/sli2+edit1/(1-exp(sli2*edit1));
        end;
    end;
end;
y2=sli3*z2*max(abs(x(:,2)))/max(abs(z2))+(1-sli3)*x(:,2);
y2=y2*max(abs(x(:,2)))/max(abs(y2));
y2=filter([1 -2 1],[1 -2*edit2 edit2^2],y2);
y2=filter([1-edit3],[1 -edit3],y2);

```

```

        y=[y1 y2];
    end
    for i=1:100,
        waitbar(i/100);
    end
    close(h);
    if data.channel==1
        if data.quantize==8
            r=(y-128)/128;
        elseif data.quantize==16
            r=y/32768;
        end
    elseif data.channel==2
        if data.quantize==8
            r=(y-128)/128;
        elseif data.quantize==16
            r=y/32768;
        end
    end
    figure(h1);
    subplot(212);
    plot(t,r);
    title('Wet Signal');
    ylabel('Normalized Values');
    if data.channel==1
        legend('Monaural Signal');
    elseif data.channel==2
        legend('Left Channel','Right Channel');
    end
    tr=data.count/data.samplerate;
    inc=0.01;
    texe=clock;soundsc(y,data.samplerate,data.quantize);e=etime(clock,texe);
    texe2=clock;h=waitbar(0,'Playback time...');e2=etime(clock,texe2);
    for i=0:inc:(tr-e-e2),
        waitbar(i/(tr-e-e2));
        pause(inc);
    end
    close(h);
case 'saveasf'
    button3=questdlg('Would you like to save this signal for further processing?','Save
file...','Microsoft PCM WAVE FORMAT','Motion Picture Expert Group (MPEG-1) Layer
3','No','No');
    if strcmp(button3,'Microsoft PCM WAVE FORMAT')
        [newfile,newpath] = uiputfile('*.wav','Save as...');
        newpath1=[newpath];
        newfile1=[newfile];
        h2=waitbar(0,'Saving in Progress... Please Wait');

```

```

        wavwrite(r,data.samplerate,data.quantize, strcat(newpath1,newfile1));
    for i=1:100,
        waitbar(i/100);
    end
    close(h2);
elseif strcmp(button3,'Motion Picture Expert Group (MPEG-1) Layer 3')
    [newfile,newpath] = uiputfile('*.mp3','Save as...');
    newpath1=[newpath];
    newfile1=[newfile];
    wavwrite(r,data.samplerate,data.quantize,'tmpfile.wav');
    string=strcat(newpath1,newfile1,'.mp3');
    if data.channel==1
        h2=waitbar(0,'Encoding and Saving in Progress... Please Wait');
        cmd=['lame --quiet -b 256 --resample 44.1 tmpfile.wav',' ',string];
        dos(cmd);
        for i=1:100,
            waitbar(i/100);
        end
        close(h2);
        delete('tmpfile.wav');
    elseif data.channel==2
        h2=waitbar(0,'Encoding and Saving in Progress... Please Wait');
        cmd=['lame --quiet -b 256 --resample 44.1 -m s tmpfile.wav',' ',string];
        dos(cmd);
        for i=1:100,
            waitbar(i/100);
        end
        close(h2);
        delete('tmpfile.wav');
    end
end
end
case 'close'
    close(gcbf);
end

```

Appendice B

Expanding e la tecnica di Noise Reduction

B.1 Come funziona il sistema

L'expander è il nome di un processore dinamico di segnale. Come dice il nome, essa amplifica il range dinamico del segnale, in modo che i segnali a basso livello siano attenuati, mentre i segnali di intensità più forte non subiscano un trattamento. Questo comportamento è opposto al funzionamento principale di un altro processore dinamico: il compressore. Il Noise Gate è il processo di expanding, portato all'estremo, dove il segnale di ingresso può essere in alcuni casi attenuato fortemente o addirittura eliminato. L'expander è essenzialmente un amplificatore con guadagno controllato e variabile. Il guadagno non è mai maggiore di uno, ed è direttamente controllato dal livello del segnale di ingresso. Quando il livello del segnale di ingresso è molto elevato, l'expander ha un guadagno pressoché unitario e quando il livello del segnale decade, decade anche il guadagno del sistema, in modo che il segnale venga percepito a volume minore.

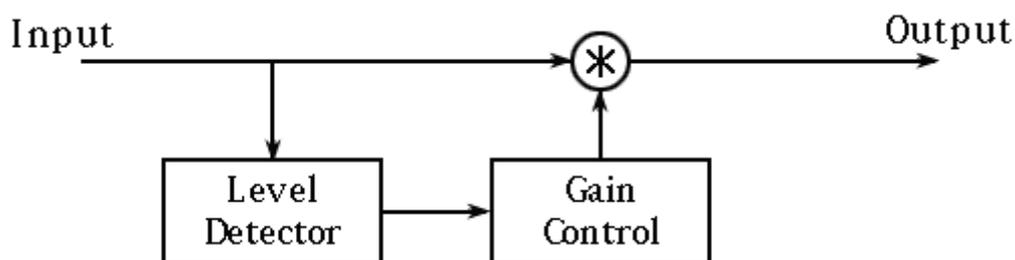


Figura 60: Diagramma di flusso di un Expander

Quando si usano unità di questo tipo, conviene riferire il livello del segnale di ingresso in dB al livello del segnale di uscita in dB, come mostrato in figura 2. Se la linea che stabilisce la caratteristica I/O del sistema, ha una pendenza di 45° , il guadagno dell'expander è pari a 1 e il livello di uscita del segnale è identico al livello di ingresso. Un cambiamento della pendenza della linea, implica un cambiamento del guadagno del sistema e molti expander hanno una linea piuttosto marcata. Il punto in cui cambia la pendenza della linea, prende il nome di Threshold o soglia ed è un valore, nella maggior parte dei casi, regolabile. Quando il livello del segnale di ingresso è al di sotto del valore di soglia, non accade nulla, ma quando il livello cade al di sotto di essa, entra la riduzione del guadagno. La riduzione del guadagno, ha la funzione di aumentare il range dinamico del segnale intero. I livelli di segnale a cui si fa riferimento, in generale non sono valori attuali, ma

vengono riferiti ad una media sopra un breve intervallo, spesso riferita al calcolo RMS (root – mean – square o valore efficace). Per esempio, una pura sinusoidale avrà valore nullo in alcuni casi, ma questo non significa assenza di segnale. Il Gain Control di un expander mostrerà un segnale con un andamento smoothed, ma mai a zero.

Expander Input/Output Characteristic

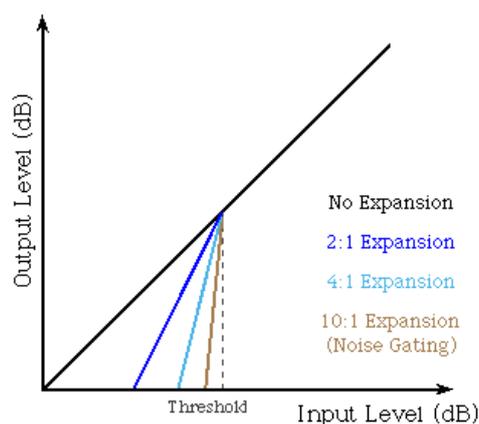


Figura 61: Caratteristica I/O di un expander

La quantità di espansione apportata al segnale, spesso viene riferita in termini di rapporto, ad es. 2:1, 4:1, ecc. Questo ha la funzione di dire, quando il livello del segnale è al di sotto del valore di soglia, di quanto il cambiamento di un livello di ingresso, impone un cambiamento del livello di uscita di 2, 4, 8, etc. volte. Quindi, un rapporto di espansione di 4:1, impone, se il livello di ingresso varia di 3dB, un decadimento di 12dB nel segnale di ingresso. Quando un expander è usato con una caratteristica I/O di un rapporto maggiore di 10:1 (caratteristica quasi orizzontale), viene riferito con il termine Noise Gate. In questo caso, il livello del segnale di ingresso, può essere notevolmente attenuato o addirittura eliminato, come una sorta di interruttore on/off per il segnale audio. Quando il segnale è a livello sufficientemente alto, l'interruttore si accende, ma quando decade invece al di sotto della soglia, l'interruttore si spegne e non c'è uscita. Il parametro chiave di un Noise Gate è quindi la soglia. Dal momento che la funzione di sensing del livello è una media a breve termine, questo processo richiede un periodo di tempo, perché il cambiamento di livello sia rilevato e quindi triggeri il sistema di guadagno. Come il compressore, l'expander si caratterizza in base ai tempi di attacco e di rilascio. Il tempo di attacco o attack time, è il tempo richiesto dall'expander per ristabilire il guadagno a 1, una volta che il segnale di ingresso supera la soglia. Allo stesso modo, il tempo in cui l'uscita viene ridotta quando il segnale cade al di sotto della soglia, viene definito tempo di rilascio o release time. I tempi hanno la funzione di dare all'expander una caratteristica di cambiamento del guadagno più morbida o più incisiva a seconda dei casi.

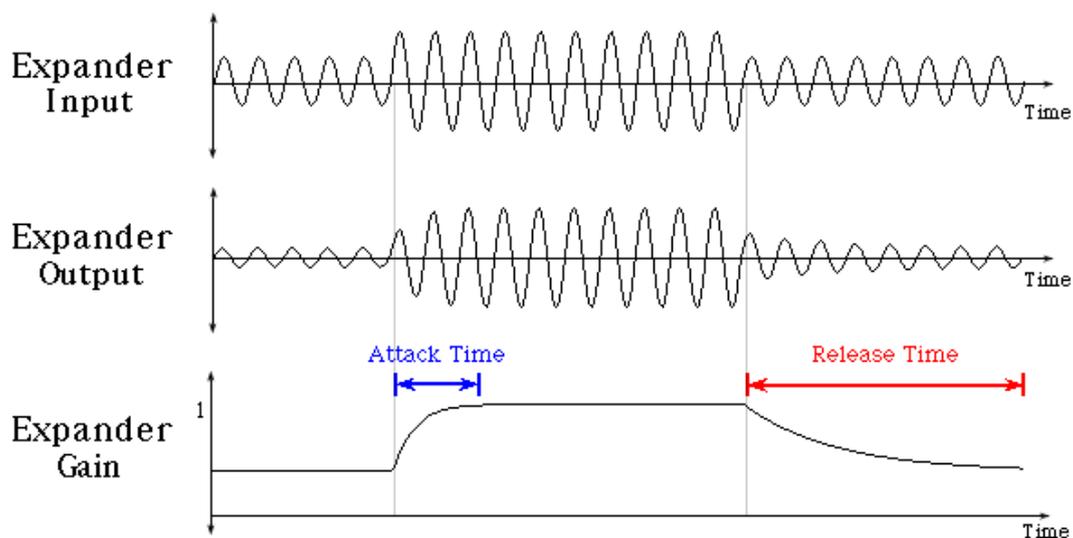


Figura 62: L'effetto di un expander su un segnale audio. Solo la porzione centrale di segnale è al di sopra del valore di soglia

B.2 Perché usare espansione e noise gating

Gli expanders trovano largo consumo nell'elettronica di consumo. Per esempio, le persone potrebbero usare gli expanders per produrre più dinamica nelle registrazioni su cassetta o vinile, le quali presentano un range dinamico piuttosto limitato. Un expander porterà la dinamica di una registrazione ad essere più drammatica durante un playback. Le più grandi applicazioni per gli expanders sono probabilmente nello studio del noise reduction. Questo aiuta la riduzione del feedback e dei rumori di ambiente e di corrente derivanti dagli strumenti musicali. I Noise Gates sono più spesso usati per eliminare rumori o fruscii anche quando il segnale in uscita da uno strumento musicale è nullo (tipicamente con la distorsione applicata). La soglia in questi casi, deve essere sufficientemente alta per eliminare i rumori di ambiente, ma senza eliminare la dinamica del suono dello strumento, in modo da non tagliare il sustain delle note in modo prematuro. Gli expanders vanno spesso accoppiati con i compressori per ridurre gli effetti del rumore durante la trasmissione audio. Un canale di trasmissione infatti, è dotato di un range dinamico limitato. Comprimere il segnale, significa che durante la trasmissione si vuole aumentare il livello del segnale per ridurre l'effetto del rumore del sistema (aumenta l'SNR). L'expander è invece usato alla ricezione per riportare il segnale alla sua caratteristica dinamica originale. Questo processo prende anche il nome di "companding". Questa è l'idea di fondo della nota tecnica di riduzione del rumore Dolby A. Nel piazzamento della catena degli effetti, il noise gate dovrebbe essere verso la fine del sound processing, tuttavia alcuni effetti come il delay e il riverbero richiedono di essere piazzati dopo il Gate per non alterare la caratteristica sonora del segnale audio.

B.3 Simulazione con Matlab

Per eliminare i rumori non voluti dall'applicazione degli effetti sul segnale audio introdotti con l'environment Nova – Mod Tools, abbiamo costruito un Noise Gate. Questo Gate permette il setup del valore di soglia, i valori per l'attivazione e la disattivazione del Gate, i tempi di attacco e di rilascio e il piazzamento del polo del filtro di rilevazione dell'involuppo del segnale (che dovrà essere molto basso per la rivelazione ottimale dell'involuppo). Riportiamo il codice dell'applicazione:

```
%Function File esterno per i callback del gate
%
%written by Zambelli Cristian
function fcnGate(action)
global x data t h1 y r;
data=load('xchangedata.mat');
switch(action)
case 'retrieve'
    t=(1:data.count)/data.samplerate);
    x=data.y;
    h=waitbar(0,'Preparing the Dry Signal...');
    for i=1:100,
        waitbar(i/100);
    end
    close(h);
    if data.channel==1
        if data.quantize==8
            z=(x-128)/128;
        elseif data.quantize==16
            z=x/32768;
        end
    elseif data.channel==2
        if data.quantize==8
            z=(x-128)/128;
        elseif data.quantize==16
            z=x/32768;
        end
    end
    h1=figure('Name','Signal Plot','NumberTitle','off');
    subplot(211);
    plot(t,z);
    title('Dry Signal');
    ylabel('Normalized Values');
    if data.channel==1
```



```

else
    g(i)=1;
    end;
    lthcnt=0;
    else
        g(i)=g(i-1);
        lthcnt=0;
        uthcnt=0;
    end;
end;
y=x.*g;
y=y*max(abs(x))/max(abs(y));
elseif data.channel==2
    rel=round(release*Fs);
    att=round(attack*Fs);
    g=zeros(size(x(:,1)));
    lthcnt=0;
    uthcnt=0;
    ht=round(holdtime*Fs);
    h=filter([(1-a)^2],[1.0000 -2*a a^2],abs(x(:,1)));
    h=h/max(h);
    for i=1:length(h)
        if (h(i)<=ltrhold) | ((h(i)<utrhold) & (lthcnt>0))
            lthcnt=lthcnt+1;
            uthcnt=0;
            if lthcnt>ht
                if lthcnt>(rel+ht)
                    g(i)=0;
                else
                    g(i)=1-(lthcnt-ht)/rel;
                end;
            elseif ((i<ht) & (lthcnt==i))
                g(i)=0;
            else
                g(i)=1;
            end;
        elseif (h(i)>=utrhold) | ((h(i)>ltrhold) & (uthcnt>0))
            uthcnt=uthcnt+1;
            if (g(i-1)<1)
                g(i)=max(uthcnt/att,g(i-1));
            end;
        else
            g(i)=1;
            end;
            lthcnt=0;
            else
                g(i)=g(i-1);
                lthcnt=0;
            end;
        end;
    end;
end;

```

```

        uthcnt=0;
    end;
end;
y1=x(:,1).*g;
y1=y1*max(abs(x(:,1)))/max(abs(y1));
rel=round(release*Fs);
att=round(attack*Fs);
g=zeros(size(x(:,2)));
lthcnt=0;
uthcnt=0;
ht=round(holdtime*Fs);
h=filter([(1-a)^2],[1.0000 -2*a a^2],abs(x(:,2)));
h=h/max(h);
for i=1:length(h)
    if (h(i)<=ltrhold) | ((h(i)<utrhold) & (lthcnt>0))
        lthcnt=lthcnt+1;
        uthcnt=0;
        if lthcnt>ht
            if lthcnt>(rel+ht)
                g(i)=0;
            else
                g(i)=1-(lthcnt-ht)/rel;
            end;
        elseif ((i<ht) & (lthcnt==i))
            g(i)=0;
        else
            g(i)=1;
        end;
        elseif (h(i)>=utrhold) | ((h(i)>ltrhold) & (uthcnt>0))
            uthcnt=uthcnt+1;
            if (g(i-1)<1)
                g(i)=max(uthcnt/att,g(i-1));
            else
                g(i)=1;
            end;
            lthcnt=0;
        else
            g(i)=g(i-1);
            lthcnt=0;
            uthcnt=0;
        end;
    end;
end;
y2=x(:,2).*g;
y2=y2*max(abs(x(:,2)))/max(abs(y2));
y=[y1 y2];
end
for i=1:100,

```

```

        waitbar(i/100);
end
close(h3);
if data.channel==1
if data.quantize==8
    r=(y-128)/128;
elseif data.quantize==16
    r=y/32768;
end
elseif data.channel==2
    if data.quantize==8
        r=(y-128)/128;
    elseif data.quantize==16
        r=y/32768;
    end
end
figure(h1);
subplot(212);
plot(t,r);
title('Gated Signal');
ylabel('Normalized Values');
if data.channel==1
    legend('Monaural Signal');
elseif data.channel==2
    legend('Left Channel','Right Channel');
end
tr=data.count/data.samplerate;
inc=0.01;
texe=clock;soundsc(y,data.samplerate,data.quantize);e=etime(clock,texe);
texe2=clock;h3=waitbar(0,'Playback time...');e2=etime(clock,texe2);
for i=0:inc:(tr-e-e2),
    waitbar(i/(tr-e-e2));
    pause(inc);
end
close(h3);
case 'saveasf'
    button3=questdlg('Would you like to save this signal for further processing?','Save
file...','Microsoft PCM WAVE FORMAT','Motion Picture Expert Group (MPEG-1) Layer
3','No','No');
    if strcmp(button3,'Microsoft PCM WAVE FORMAT')
        [newfile,newpath] = uiputfile('*.wav','Save as...');
        newpath1=[newpath];
        newfile1=[newfile];
        h2=waitbar(0,'Saving in Progress... Please Wait');
        wavwrite(r,data.samplerate,data.quantize, strcat(newpath1,newfile1));
        for i=1:100,
            waitbar(i/100);

```

```

end
close(h2);
elseif strcmp(button3,'Motion Picture Expert Group (MPEG-1) Layer 3')
    [newfile,newpath] = uiputfile('*.mp3','Save as...');
    newpath1=[newpath];
    newfile1=[newfile];
    wavwrite(r,data.samplerate,data.quantize,'tmpfile.wav');
    string=strcat(newpath1,newfile1,'.mp3');
    if data.channel==1
        h2=waitbar(0,'Encoding and Saving in Progress... Please Wait');
        cmd=['lame --quiet -b 256 --resample 44.1 tmpfile.wav',' ',string];
        dos(cmd);
        for i=1:100,
            waitbar(i/100);
        end
        close(h2);
        delete('tmpfile.wav');
    elseif data.channel==2
        h2=waitbar(0,'Encoding and Saving in Progress... Please Wait');
        cmd=['lame --quiet -b 256 --resample 44.1 -m s tmpfile.wav',' ',string];
        dos(cmd);
        for i=1:100,
            waitbar(i/100);
        end
        close(h2);
        delete('tmpfile.wav');
    end
end
end
case 'close'
    close(gcbf);
end

```

Appendice C

Equalizzazione di un segnale audio

C.1 Come funziona un equalizzatore

L'equalizzazione (EQ) è il processo di amplificare o tagliare certe componenti frequenziali in un segnale. Il nome nasce dall'applicazione che tenta di ottenere una risposta in frequenza piatta o senza colorazione. Ad esempio, quando si trasmettono segnali vocali analogici su cavi a lunga distanza, le alte frequenze sono attenuate a causa delle capacità parassite. Applicando alcuni filtri di equalizzazione, questa perdita può essere ripristinata, in modo che la voce suoni più "naturale" in fase di ricezione. L'equalizzazione è quindi un tool fondamentale nella registrazione musicale per portare nuova vita ad una certa quantità di frequenze.

C.1.1 Controllo di Tono

Il sistema di equalizzazione più comune è probabilmente il controllo di tono, presente nella maggior parte dei sistemi stereofonici. Essi provvedono un metodo semplice e veloce per aggiustare il suono, in modo che esso riesca ad incontrare il nostro gusto personale e che compensi parzialmente la risposta dell'ambiente circostante. Nella maggior parte dei casi si possono incontrare controlli nominati come "bass" e "treble". Ognuno di questi controlli è legato ad uno speciale tipo di filtro, detto filtro di shelving (tosatura) o più precisamente un filtro passa-basso di tosatura e passa-alto di tosatura. La risposta di questi filtri, la possiamo vedere in figura 1. (La maggior parte dei filtri ha un guadagno che cambia con la frequenza. I grafici sono rispettivi all'ampiezza della risposta in frequenza, che mostra il guadagno ad ogni componente frequenziale. Un guadagno maggiore di uno incrementerà il segnale, viceversa, se il guadagno è minore di uno, il suono viene tagliato.

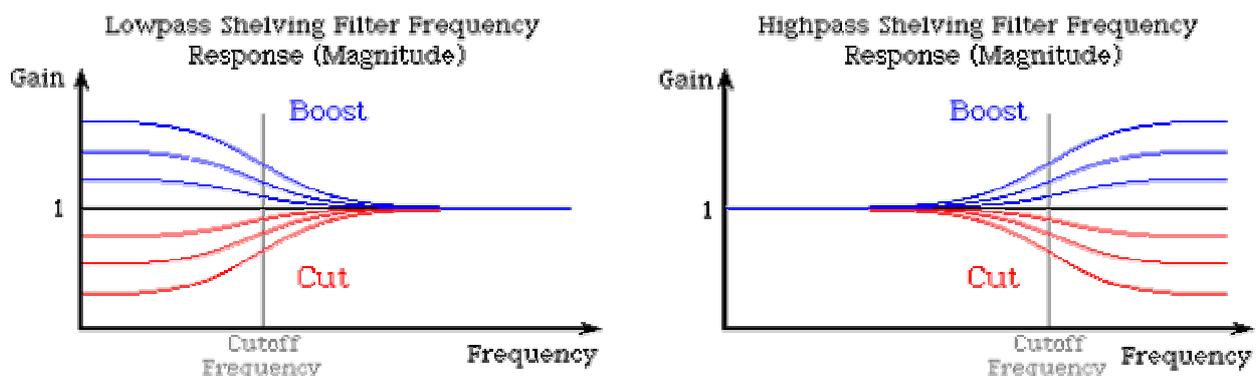


Figura 63: Risposta in frequenza dei filtri di tosatura

Nella maggior parte delle applicazioni, i filtri passa – basso e passa – alto, tentano di rimuovere totalmente una porzione di segnale dallo spettro. Ad esempio, un filtro passa – basso tenta di eliminare tutte le alte frequenze. I filtri di tosatura invece, non tentano di rimuovere nulla, amplificano o tagliano solo una porzione del segnale, lasciando il resto dello stesso, intatto. La frequenza dove la risposta in frequenza crea delle transizioni tra due livelli di guadagno (anche se è piuttosto graduale), è chiamata frequenza di cut – off. È possibile progettare un controllo di tono che permette la modifica di questa frequenza di cut – off, oltre al livello di taglio o amplificazione, ma questa è abitualmente fissata in fase di progetto e non può essere modificata dall'utente. In aggiunta ai controlli bass and treble, molto spesso si trovano controlli con il nome di “mid”, come nei comuni eq a 3 bande nei mixer di segnale. Come si può dedurre, questo controllo, si applica alle frequenze tra le alte e le basse. Questo è spesso riferito ad un band – pass filter o peaking (picco). Ancora una volta, non si cerca di isolare certe frequenze, ma di amplificare o attenuare una piccola porzione dello spettro, senza modificare il resto del segnale. Questo tipo di filtro generalmente non ha una frequenza di cut – off definita, ma è caratterizzato invece da altri due parametri. La frequenza alla quale il peaking filter è al guadagno massimo (o al minimo quando si attenua), è definita frequenza di centro banda. L'altra importante caratteristica è la larghezza di banda che definisce l'ampiezza del filtraggio (o quanto ampio è il range di frequenze affette). Generalmente si può modificare solo la quantità di amplificazione/attenuazione, mentre la frequenza di centro banda e la larghezza della stessa, sono fissate.

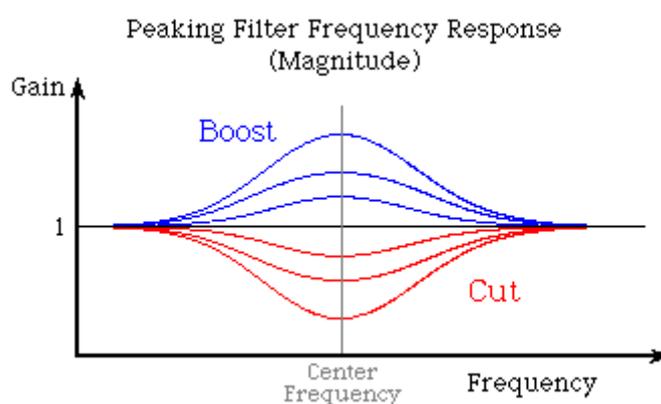


Figura 64: Risposta in frequenza del filtro di peaking

I controlli di tono sono un modo molto semplice di effettuare l'equalizzazione e sono costituiti spesso da due o tre filtri connessi in serie.

C.1.2 Equalizzatori Grafici

Gli equalizzatori grafici sono un passo successivo dei controlli di tono in termini di flessibilità e controllo e le operazioni rimangono ancora semplici. Un eq grafico è semplicemente un set di filtri in cui, la frequenza di centro banda è fissata e non può essere modificata. L'unico controllo è costituito da uno slider che stabilisce la quantità di amplificazione/attenuazione in ogni banda di frequenza. Questo parametro è spesso controllato da potenziometri in formato sliders. Questa interfaccia piuttosto intuitiva permette di capire la risposta in frequenza semplicemente guardando la posizione degli sliders. Siccome gli sliders sono una rappresentazione grafica della risposta in frequenza, ecco il nome di equalizzatori grafici. Alcuni sistemi HI-FI incorporano equalizzatori grafici, ma il loro uso primario è nel sound reinforcement e per compensare gli effetti dell'ambiente circostante. Per esempio, quando si suona in una arena, può essere desiderabile ottenere una risposta in frequenza piatta per il sistema sonoro. Le risonanze di una stanza e gli altoparlanti sono responsabili della colorazione di un suono. Con un equalizzatore grafico che copre la maggior parte dello spettro audio, si può contrattaccare la colorazione del suono, così se si suona in una stanza diversa ogni notte, il suono degli esecutori e degli strumenti resta consistente. A livello di effetti, semplici equalizzatori stompboxes possono essere molto utili per ottenere un aumento di volume o di cambiamento di tono specialmente nei soli. L'attuale implementazione degli equalizzatori grafici è diversa dai comuni controlli di tono. I potenziometri bass e treble nell'impianto stereo amplificano/attenuano solo certe bande di frequenza, mentre il resto rimane normale, cosicché si possano collegare in serie. Un equalizzatore grafico usa un set di band – pass filter che sono progettati per isolare completamente certe bande di frequenza. La figura 3 mostra la risposta in frequenza per un filtro passa – banda ideale. Per avere un controllo sopra l'intero spettro del segnale i filtri devono essere connessi in parallelo. Ogni filtro nell'equalizzatore grafico ha lo stesso ingresso. Il loro lavoro è di ammettere il passaggio di una banda ristretta di frequenze.

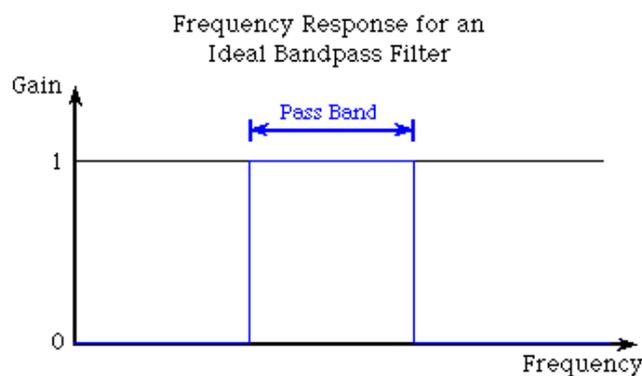


Figura 65: Risposta in frequenza di un filtro passa banda ideale. Tutti filtri reali possiedono invece un ripple

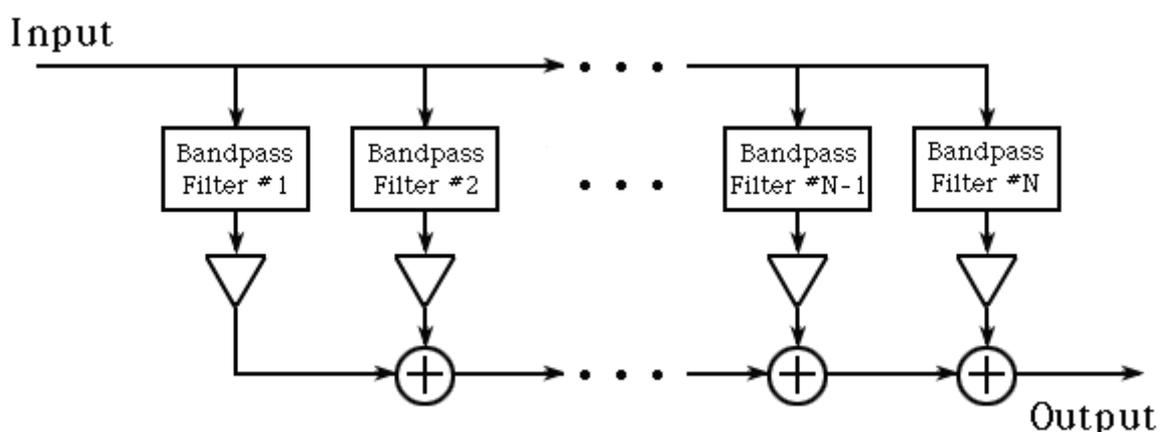


Figura 66: Equalizzatore grafico con N bande di controllo

Una volta che il segnale passa attraverso i filtri passa banda, è possibile manipolare ognuna di queste bande di frequenza indipendentemente, inserendo una sorta di controllo di guadagno. Gli sliders sull'eq grafico sono i controlli di guadagno in ogni banda. La connessione in parallelo dei filtri, comparata alla connessione in serie dei controlli di tono, è usata per ridurre alcuni effetti dannosi del filtraggio. La risposta in ampiezza mostrata infatti, non ci dice nulla sul comportamento del filtro. Esso infatti possiede una risposta di fase. Mentre in alcuni casi la distorsione di fase è accettabile (come negli effetti Phaser), nella maggior parte delle applicazioni di sound reinforcement, vogliamo assicurarci che il suono sia colorato il meno possibile. Per ogni filtro aggiunto in serie, la risposta in fase è aggiunta a quella degli altri filtri. La risposta in fase inoltre, ha la funzione di introdurre ritardo sul segnale. Se ci sono due o tre filtri nel controllo di tono, la connessione in serie è accettabile, ma con 15 o 31 bande, la distorsione di fase comincia a fare la differenza. Le frequenze di centro banda su gli eq grafici sono abitualmente equi – spaziate in ottave, e non in maniera lineare. Per esempio, si possono acquistare eq grafici con spaziature di $1/3$ o $1/6$ di ottava. Un eq con una spaziatura di $1/3$ di ottava è basato su un fattore di spaziatura di $2^{1/3}$,

che è circa 1.26. Quindi se la prima banda è a 100 Hz, la bande ad un terzo di ottava saranno: 126 Hz, 159 Hz, 200 Hz, ecc. Esiste comunque uno standard ISO sulle frequenze da utilizzare preferibilmente.

C.1.3 Equalizzatori Parametrici

Gli equalizzatori parametrici sono l'ultima richiesta in termini di flessibilità, ma richiedono una cura particolare per essere usati efficientemente. Un singolo eq parametrico ammette di settare non solo la quantità di amplificazione/attenuazione, ma anche la frequenza di centro banda e la larghezza della stessa. Dall'esperienza, si può imparare che applicare un'amplificazione determinata, si può ottenere un suono migliore da un certo strumento musicale. Per la cancellazione dei feedback, un eq parametrico con molta attenuazione (chiamato anche notch filter), può essere posizionato sulla frequenza in cui il feedback ha luogo. Per minimizzare l'effetto del filtro sul resto del suono, si usi una banda stretta. È possibile eliminare i feedback anche con eq grafici, ma se le bande sono piuttosto ampie, la maggior parte del suono verrà tagliata ed il suono risulterà innaturale. Un applicazione simile potrebbe consistere nella rimozione del rumore dei 50 Hz in una registrazione (ricordandosi di eliminarne anche gli armonici). I filtri di tosatura sono essenzialmente dei filtri parametrici, almeno matematicamente, perché nella maggior parte dei prodotti commerciali solo alcuni parametri sono modificabili dall'utente.

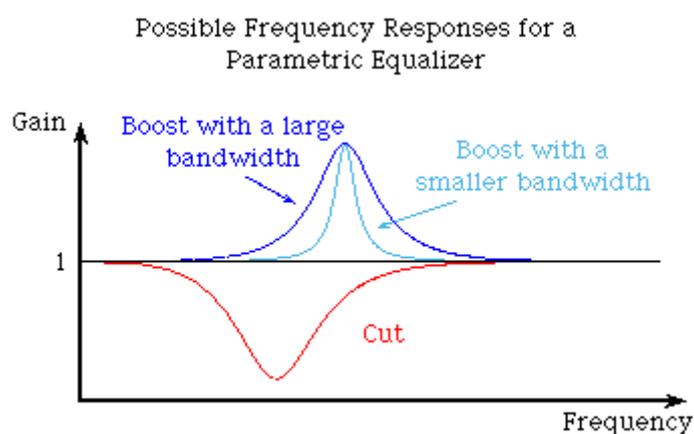


Figura 67: Risposta in frequenza di un equalizzatore parametrico

C.2 Sfere di applicazione

C.2.1 Presenza

Molti amplificatori hanno il controllo di presenza come parte del controllo di tono. Questo consiste semplicemente in un'amplificazione dalle medie alle alte frequenze, ad esempio da 2 a 6 KHz.

Questo controllo è così definito perché sembra sia in grado di alterare il suono di uno strumento nella fase di registrazione, il che dà una maggiore impressione di presenza dello stesso.

C.2.2 Speaker Cross – Over

Non è propriamente vero che gli speakers cross – over siano un tipo di equalizzatori. È molto difficile progettare speaker e cabinet che abbiano una risposta in frequenza piatta su tutto il range di frequenze dello spettro audio. Se la banda complessiva dello spettro audio è più stretta, il design risulta essere più semplice, viceversa, serve più di uno speaker. Per questo motivo la maggior parte dei sistemi HI-FI contiene almeno due o tre altoparlanti. Il più grande è per le basse frequenze ed è chiamato woofer. Il più piccolo, per le alte frequenze, è chiamato tweeter. Un terzo speaker può essere usato per la definizione delle medie frequenze. Si può danneggiare uno speaker se lo si forza a riprodurre frequenze per cui non è stato progettato, per cui alcuni speaker cabinets hanno al loro interno un circuito di cross – over. Uno speaker cross – over è una rete di filtraggio che prende il segnale audio in ingresso e lo divide in componenti che ogni speaker può gestire. Se si ha una combinazione di woofer e tweeter, il cross – over richiederà un filtro passa – basso e passa – alto (ma non di tosatura). Se il sistema ha anche uno speaker per il midrange, si può aggiungere anche un filtro passa – banda che sta all'interno del range di woofer e tweeter.

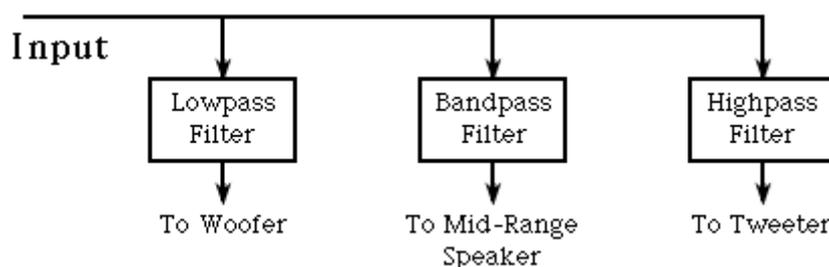


Figura 68: Esempio di un sistema di cross – over

C.3 Simulazione con Matlab

Il sistema di equalizzazione progettato con Matlab, implementa un sistema di equalizzazione in cui è possibile scegliere da una lista di preset, che tipo di risposta in frequenza si vuole applicare al segnale. I filtri sono applicati in parallelo ed utilizzano una configurazione con un filtraggio di tipo IIR Butterworth ad ordine variabile per preset. Riportiamo il codice dell'applicazione:

```
%Function File esterno per i callback del quick equalizer
%
%written by Zambelli Cristian
function fcnQeq(action)
global data y;
data=load('xchangedata.mat');
switch(action)
case 'eqit'
    s=data.y;
    if data.quantize==8
        s=(s-128)/128;
    elseif data.quantize==16
        s=s/32768;
    end
    listh=findobj(gcf,'Tag','Listbox1');
    list=get(listh,'Value');
    t=(1:data.count)/data.samplerate);
    if list==1
        h=waitbar(0,'Performing eq...');
        for i=1:100,
            waitbar(i/100);
        end
        close(h);
        [b1,a1]=butter(4,[50/((data.samplerate)/2) 400/((data.samplerate)/2)]);
        [b2,a2]=butter(4,400/((data.samplerate)/2),'high');
        if data.channel==1
            y=1.5*filter(b1,a1,s)+filter(b2,a2,s);
        elseif data.channel==2
            y(:,1)=1.5*filter(b1,a1,s(:,1))+filter(b2,a2,s(:,1));
            y(:,2)=1.5*filter(b1,a1,s(:,2))+filter(b2,a2,s(:,2));
        end
        h1=figure('Name','Signal Response','NumberTitle','off');
        subplot(211);
        plot(t,s);
        title('Original Signal');
        ylabel('Normalized Values');
        subplot(212);
```

```

    plot(t,y);
    title('Equalized Signal');
    ylabel('Normalized Values');
    h3=figure('Name','System Response','NumberTitle','off');
    freqz(b1,a1,data.samplerate/2);
tr=data.count/data.samplerate;
inc=0.01;
texe=clock;sound(y,data.samplerate,data.quantize);e=etime(clock,texe);
texe2=clock;h=waitbar(0,'Playback time (Wet)...');e2=etime(clock,texe2);
for i=0:inc:(tr-e-e2),
    waitbar(i/(tr-e-e2));
    pause(inc);
end
close(h);
end
if list==2
    h=waitbar(0,'Performing eq...');
    for i=1:100,
        waitbar(i/100);
    end
    close(h);
    [b1,a1]=butter(4,[2000/((data.samplerate)/2) 10000/((data.samplerate)/2)]);
    [b2,a2]=butter(4,2000/((data.samplerate)/2));
    if data.channel==1
        y=1.5*filter(b1,a1,s)+filter(b2,a2,s);
    elseif data.channel==2
        y(:,1)=1.5*filter(b1,a1,s(:,1))+filter(b2,a2,s(:,1));
        y(:,2)=1.5*filter(b1,a1,s(:,2))+filter(b2,a2,s(:,2));
    end
    h1=figure('Name','Signal Response','NumberTitle','off');
    subplot(211);
    plot(t,s);
    title('Original Signal');
    ylabel('Normalized Values');
    subplot(212);
    plot(t,y);
    title('Equalized Signal');
    ylabel('Normalized Values');
    h3=figure('Name','System Response','NumberTitle','off');
    freqz(b1,a1,data.samplerate/2);
tr=data.count/data.samplerate;
inc=0.01;
texe=clock;sound(y,data.samplerate,data.quantize);e=etime(clock,texe);
texe2=clock;h=waitbar(0,'Playback time (Wet)...');e2=etime(clock,texe2);
for i=0:inc:(tr-e-e2),
    waitbar(i/(tr-e-e2));
    pause(inc);
end

```

```

end
close(h);
end
if list==3
    h=waitbar(0,'Performing eq...');
    for i=1:100,
        waitbar(i/100);
    end
    close(h);
    [b1,a1]=butter(4,[50/((data.samplerate)/2) 400/((data.samplerate)/2)]);
    [b2,a2]=butter(4,400/((data.samplerate)/2),'high');
    if data.channel==1
        y=5*filter(b1,a1,s)+filter(b2,a2,s);
    elseif data.channel==2
        y(:,1)=5*filter(b1,a1,s(:,1))+filter(b2,a2,s(:,1));
        y(:,2)=5*filter(b1,a1,s(:,2))+filter(b2,a2,s(:,2));
    end
    h1=figure('Name','Signal Response','NumberTitle','off');
    subplot(211);
    plot(t,s);
    title('Original Signal');
    ylabel('Normalized Values');
    subplot(212);
    plot(t,y);
    title('Equalized Signal');
    ylabel('Normalized Values');
    h3=figure('Name','System Response','NumberTitle','off');
    freqz(b1,a1,data.samplerate/2);
    tr=data.count/data.samplerate;
    inc=0.01;
    texe=clock;sound(y,data.samplerate,data.quantize);e=etime(clock,texe);
    texe2=clock;h=waitbar(0,'Playback time (Wet)...');e2=etime(clock,texe2);
    for i=0:inc:(tr-e-e2),
        waitbar(i/(tr-e-e2));
        pause(inc);
    end
    close(h);
end
if list==4
    h=waitbar(0,'Performing eq...');
    for i=1:100,
        waitbar(i/100);
    end
    close(h);
    [b1,a1]=butter(4,[2000/((data.samplerate)/2) 10000/((data.samplerate)/2)]);
    [b2,a2]=butter(4,2000/((data.samplerate)/2));
    if data.channel==1

```

```

        y=5*filter(b1,a1,s)+filter(b2,a2,s);
elseif data.channel==2
    y(:,1)=5*filter(b1,a1,s(:,1))+filter(b2,a2,s(:,1));
    y(:,2)=5*filter(b1,a1,s(:,2))+filter(b2,a2,s(:,2));
end
h1=figure('Name','Signal Response','NumberTitle','off');
subplot(211);
plot(t,s);
title('Original Signal');
ylabel('Normalized Values');
subplot(212);
plot(t,y);
title('Equalized Signal');
ylabel('Normalized Values');
h3=figure('Name','System Response','NumberTitle','off');
freqz(b1,a1,data.samplerate/2);
tr=data.count/data.samplerate;
inc=0.01;
texe=clock;sound(y,data.samplerate,data.quantize);e=etime(clock,texe);
texe2=clock;h=waitbar(0,'Playback time (Wet)...');e2=etime(clock,texe2);
for i=0:inc:(tr-e-e2),
    waitbar(i/(tr-e-e2));
    pause(inc);
end
close(h);
end
if list==5
    h=waitbar(0,'Performing eq...');
    for i=1:100,
        waitbar(i/100);
    end
    close(h);
    [b1,a1]=butter(4,[500/((data.samplerate)/2) 2000/((data.samplerate)/2)]);
    [b2,a2]=butter(4,500/((data.samplerate)/2));
    [b3,a3]=butter(4,2000/((data.samplerate)/2),'high');
    if data.channel==1
        y=2*filter(b1,a1,s)+filter(b2,a2,s)+filter(b3,a3,s);
    elseif data.channel==2
        y(:,1)=2*filter(b1,a1,s(:,1))+filter(b2,a2,s(:,1))+filter(b3,a3,s(:,1));
        y(:,2)=2*filter(b1,a1,s(:,2))+filter(b2,a2,s(:,2))+filter(b3,a3,s(:,1));
    end
    h1=figure('Name','Signal Response','NumberTitle','off');
    subplot(211);
    plot(t,s);
    title('Original Signal');
    ylabel('Normalized Values');
    subplot(212);

```

```

    plot(t,y);
    title('Equalized Signal');
    ylabel('Normalized Values');
    h3=figure('Name','System Response','NumberTitle','off');
    freqz(b1,a1,data.samplerate/2);
tr=data.count/data.samplerate;
inc=0.01;
texe=clock;sound(y,data.samplerate,data.quantize);e=etime(clock,texe);
texe2=clock;h=waitbar(0,'Playback time (Wet)...');e2=etime(clock,texe2);
for i=0:inc:(tr-e-e2),
    waitbar(i/(tr-e-e2));
    pause(inc);
end
close(h);
end
if list==6
    h=waitbar(0,'Performing eq...');
    for i=1:100,
        waitbar(i/100);
    end
    close(h);
    [b1,a1]=butter(2,[500/((data.samplerate)/2) 2000/((data.samplerate)/2)],'stop');
    [b2,a2]=butter(2,500/((data.samplerate)/2));
    [b3,a3]=butter(2,2000/((data.samplerate)/2),'high');
    if data.channel==1
        y=2*filter(b1,a1,s)+filter(b2,a2,s)+filter(b3,a3,s);
    elseif data.channel==2
        y(:,1)=2*filter(b1,a1,s(:,1))+filter(b2,a2,s(:,1))+filter(b3,a3,s(:,1));
        y(:,2)=2*filter(b1,a1,s(:,2))+filter(b2,a2,s(:,2))+filter(b3,a3,s(:,1));
    end
    h1=figure('Name','Signal Response','NumberTitle','off');
    subplot(211);
    plot(t,s);
    title('Original Signal');
    ylabel('Normalized Values');
    subplot(212);
    plot(t,y);
    title('Equalized Signal');
    ylabel('Normalized Values');
    h3=figure('Name','System Response','NumberTitle','off');
    freqz(b1,a1,data.samplerate/2);
tr=data.count/data.samplerate;
inc=0.01;
texe=clock;sound(y,data.samplerate,data.quantize);e=etime(clock,texe);
texe2=clock;h=waitbar(0,'Playback time (Wet)...');e2=etime(clock,texe2);
for i=0:inc:(tr-e-e2),
    waitbar(i/(tr-e-e2));

```

```

    pause(inc);
end
close(h);
end
if list==7
    h=waitbar(0,'Performing eq...');
    for i=1:100,
        waitbar(i/100);
    end
    close(h);
    [b1,a1]=butter(3,[48/((data.samplerate)/2) 52/((data.samplerate)/2)],'stop');
    [b2,a2]=butter(3,52/((data.samplerate)/2),'high');
    if data.channel==1
        y=filter(b1,a1,s)+filter(b2,a2,s);
    elseif data.channel==2
        y(:,1)=filter(b1,a1,s(:,1))+filter(b2,a2,s(:,1));
        y(:,2)=filter(b1,a1,s(:,2))+filter(b2,a2,s(:,2));
    end
    h1=figure('Name','Signal Response','NumberTitle','off');
    subplot(211);
    plot(t,s);
    title('Original Signal');
    ylabel('Normalized Values');
    subplot(212);
    plot(t,y);
    title('Equalized Signal');
    ylabel('Normalized Values');
    h3=figure('Name','System Response','NumberTitle','off');
    freqz(b1,a1,data.samplerate/2);
    tr=data.count/data.samplerate;
    inc=0.01;
    texe=clock;sound(y,data.samplerate,data.quantize);e=etime(clock,texe);
    texe2=clock;h=waitbar(0,'Playback time (Wet)...');e2=etime(clock,texe2);
    for i=0:inc:(tr-e-e2),
        waitbar(i/(tr-e-e2));
        pause(inc);
    end
    close(h);
end
if list==8
    h=waitbar(0,'Performing eq...');
    for i=1:100,
        waitbar(i/100);
    end
    close(h);
    [b1,a1]=butter(2,25/((data.samplerate)/2),'high');
    if data.channel==1

```

```

        y=filter(b1,a1,s);
elseif data.channel==2
    y(:,1)=filter(b1,a1,s(:,1));
    y(:,2)=filter(b1,a1,s(:,2));
end
h1=figure('Name','Signal Response','NumberTitle','off');
subplot(211);
plot(t,s);
title('Original Signal');
ylabel('Normalized Values');
subplot(212);
plot(t,y);
title('Equalized Signal');
ylabel('Normalized Values');
h3=figure('Name','System Response','NumberTitle','off');
freqz(b1,a1,data.samplerate/2);
tr=data.count/data.samplerate;
inc=0.01;
texe=clock;sound(y,data.samplerate,data.quantize);e=etime(clock,texe);
texe2=clock;h=waitbar(0,'Playback time (Wet)...');e2=etime(clock,texe2);
for i=0:inc:(tr-e-e2),
    waitbar(i/(tr-e-e2));
    pause(inc);
end
close(h);
end
if list==9
    h=waitbar(0,'Performing eq...');
    for i=1:100,
        waitbar(i/100);
    end
    close(h);
    [b1,a1]=butter(3,[435/((data.samplerate)/2) 446/((data.samplerate)/2)]);
    if data.channel==1
        y=2*filter(b1,a1,s);
    elseif data.channel==2
        y(:,1)=2*filter(b1,a1,s(:,1));
        y(:,2)=2*filter(b1,a1,s(:,2));
    end
    h1=figure('Name','Signal Response','NumberTitle','off');
    subplot(211);
    plot(t,s);
    title('Original Signal');
    ylabel('Normalized Values');
    subplot(212);
    plot(t,y);
    title('Equalized Signal');

```

```

ylabel('Normalized Values');
h3=figure('Name','System Response','NumberTitle','off');
freqz(b1,a1,data.samplerate/2);
tr=data.count/data.samplerate;
inc=0.01;
texe=clock;sound(y,data.samplerate,data.quantize);e=etime(clock,texe);
texe2=clock;h=waitbar(0,'Playback time (Wet)...');e2=etime(clock,texe2);
for i=0:inc:(tr-e-e2),
    waitbar(i/(tr-e-e2));
    pause(inc);
end
close(h);
end
case 'saveasf'
    button3=questdlg('Would you like to save this signal for further processing?','Save
file...','Microsoft PCM WAVE FORMAT','Motion Picture Expert Group (MPEG-1) Layer
3','No','No');
    if strcmp(button3,'Microsoft PCM WAVE FORMAT')
        [newfile,newpath] = uiputfile('*.wav','Save as...');
        newpath1=[newpath];
        newfile1=[newfile];
        h2=waitbar(0,'Saving in Progress... Please Wait');
        wavwrite(y,data.samplerate,data.quantize, strcat(newpath1,newfile1));
        for i=1:100,
            waitbar(i/100);
        end
        close(h2);
    elseif strcmp(button3,'Motion Picture Expert Group (MPEG-1) Layer 3')
        [newfile,newpath] = uiputfile('*.mp3','Save as...');
        newpath1=[newpath];
        newfile1=[newfile];
        wavwrite(y,data.samplerate,data.quantize,'tmpfile.wav');
        string=strcat(newpath1,newfile1,'.mp3');
        if data.channel==1
            h2=waitbar(0,'Encoding and Saving in Progress... Please Wait');
            cmd=['lame --quiet -b 256 --resample 44.1 tmpfile.wav',' ',string];
            dos(cmd);
            for i=1:100,
                waitbar(i/100);
            end
            close(h2);
            delete('tmpfile.wav');
        elseif data.channel==2
            h2=waitbar(0,'Encoding and Saving in Progress... Please Wait');
            cmd=['lame --quiet -b 256 --resample 44.1 -m s tmpfile.wav',' ',string];
            dos(cmd);
            for i=1:100,

```

```
        waitbar(i/100);
    end
    close(h2);
    delete('tmpfile.wav');
end
end
case 'close'
    close(gcf);
end
```

Bibliografia

- Matlab Helpdesk System and Signal Processing Handbook, built – in Matlab v5.3(R11) written by MathWorks Corporation
- Harmony Central web pages @ <http://www.harmony-central.com>
- ZynAddSubFX Synth Master web pages
- Syntrillium SDK Cool Edit Pro @ <http://www.syntrillium.com>
- Joe's Dattorro Articles and the sixth DAFX Conference
- Audio Engineering Society Journal

Tutti i marchi citati in questa tesi, sono coperti da diritti di copyright e registrati dai rispettivi proprietari. I software utilizzati per la realizzazione della tesi sono stati forniti in modalità shareware, freeware o sotto licenza GNU/GPL. Per Syntrillium Cool Edit Pro, si è utilizzata la Evaluation Version 21 day – trial.

Ringraziamenti

È stato un lavoro lungo e faticoso, che di certo non avrei potuto compiere senza l'apporto di determinate persone. Persone che in ogni momento, hanno cercato di comprendere le mie necessità e le mie problematiche, aiutandomi a portare a termine questa non facile impresa. Primo fra tutti vorrei ringraziare l'Ing. Davide Bertozzi, perché grazie ad esso, sono riuscito a svolgere un progetto a cui tenevo da molto molto tempo. Proprio questa persona, mi ha aiutato in tutto e per tutto lo svolgimento della tesi, consigliandomi saggiamente per la realizzazione del software e facendomi sentire, ogni qualvolta dovevo varcare la soglia dell'ufficio per portare a revisionare un capitolo della tesi o per mostrargli il software appena sviluppato, come se stessi andando a parlare con un buon amico. È stata l'unica persona a fidarsi ciecamente di tutto questo... Ringrazio anche il gruppo di Acustica dell'Università di Ferrara per essersi interessati a questo progetto e non può che farmi piacere, sapere che qualcun altro in questa facoltà, desidera poter vedere all'opera il sistema realizzato. Vorrei anche ringraziare il Prof. Piero Olivo e scusarmi con lui per il disturbo recatogli, nel chiedere insistentemente a chi potevo rivolgermi per svolgere la tesi di laurea, ma vede Prof., non è stato facile nemmeno per me, essere rimbalzato da un po' tutti i professori e intanto vedere il trascorrere del tempo senza poter mettersi all'opera. Al di fuori dell'Università desidero ringraziare i miei fratelli dell'esercito, acquisiti durante le battaglie per la conquista della luna, nelle persone di: Denis Servelli, Mirko Cirelli, Sebastiano Barbirato e il Dr. Emanuele Checcoli (che purtroppo non fa più parte dell'esercito, ma che è ancora al nostro fianco) e chi legge queste righe sa a cosa mi riferisco MOONLIT KNOWS NO MERCY. Ringrazio anche i players di D&D e gli amici che in tutto questo tempo non ho potuto frequentare, per i mille impegni. Il mio più grande ringraziamento va però ai miei genitori Marzia e Marco, ai miei nonni e ai miei parenti tutti, perché in questi mesi si sono dovuti sorbire i miei orari impossibili e le mie sfuriate quando il software non funzionava; sono però contento per loro, perché ora possono finalmente vedere l'inizio del coronamento dei loro sogni. Per ultimo, ma non in ordine di importanza, desidero ringraziare Margherita, la mia compagna nella vita, la persona che per me significa tutto e anche se so che in questi mesi di duro lavoro non sono stato presente come lei avrebbe desiderato, ho continuato a pensare a lei in ogni momento della giornata, consolandomi con il fatto che il nostro amore continuerà a lungo anche dopo questo lavoro... Per tutti quelli che non ho ringraziato pubblicamente, compreso chi in questi mesi non ha creduto in me e ha ben pensato di deridere il mio lavoro ponendo sempre più ostacoli a quanti già non ce ne fossero, ho solo una parola da dire...

Grazie