UNIVERSITA' POLITECNICA DELLE MARCHE

FACOLTA' DI INGEGNERIA

Corso di Laurea triennale in Ingegneria Informatica e dell'Automazione

IMPLEMENTAZIONE E TESTING REAL TIME DI UN ALGORITMO PER L'EMULAZIONE DI UN AMPLIFICATORE A VALVOLE

IMPLEMENTATION AND TESTING OF A REAL TIME ALGORITHM FOR A TUBE AMPLIFIER EMULATION

Relatore: Chiar.ma

Prof. **CECCHI STEFANIA**

Correlatori:

Dott. **TERENZI ALESSANDRO**

Prof. **SPINSANTE SUSANNA**

Tesi di Laurea di:

NIKPALI RICCARDO

A.A. 2019/2020

Abstract

Il progetto alla base di questa tesi di laurea si pone come obiettivo la realizzazione di un plugin in grado di emulare la distorsione caratteristica delle valvole saturate di un amplificatore per chitarra, che possa processare in tempo reale e a bassa latenza un segnale audio.

Il plugin è stato implementato sotto forma di codice C all'interno dell'ambiente di sviluppo (IDE) Visual Studio. Il codice compilato è stato poi eseguito sulla piattaforma per l'elaborazione del segnale digitale (DSP) NU-Tech [1]. Si è partiti quindi da una funzione non lineare che realizza un clipping asimmetrico, approssimando l'effetto che le valvole a triodo hanno sugli ingressi di una certa ampiezza, come illustrato in [2].

Indice

1. La distorsione valvolare: come nasce e come può essere emulata

1.1 Gli Amplificatori Valvolari

All'inizio del ventesimo secolo negli Stati Uniti la chitarra era uno strumento dalla crescente popolarità, la cui struttura era ancora vicina a quella che oggi è conosciuta come chitarra classica, caratterizzata da un suono dolce ma dal volume limitato. Proprio la ricerca di un volume maggiore ha portato con il tempo all'introduzione di diverse innovazioni come l'utilizzo delle corde di acciaio, portando quindi alla creazione delle cosiddette chitarre acustiche.
Lo sviluppo di questo strumento è sempre andato di pari passo con la nascita e l'evoluzione di nuovi generi quali il blues e il jazz, che iniziavano a muovere i loro primi passi negli stati del Sud come Mississippi e Louisiana intorno agli anni '10 e '20. All'interno dei complessi musicali il numero di strumenti e musicisti continuava ad aumentare, andando a nascondere il suono dei chitarristi che adottavano pertanto le soluzioni più disparate per ovviare a questo problema, ad esempio aggiungendo rudimentali microfoni a carbone all'interno dello strumento.

La svolta arriva nel 1931 con l'invenzione del pick-up, un trasduttore elettromagnetico capace di convertire la vibrazione delle corde in segnale elettrico da inviare ad un altoparlante. I risultati erano ancora ben lontani dall'essere soddisfacenti, in quanto in primo luogo le chitarre presentavano grandi casse di risonanza che mandavano in feedback il suono emesso, inoltre i sistemi di amplificazione disponibili non erano in grado di amplificare correttamente un range di frequenze adeguato allo strumento. È qui che entrano in gioco gli amplificatori valvolari: fino ad allora erano stati utilizzati per i sistemi di audiodiffusione nei cinema, ma grazie alla loro portabilità e in particolare alla qualità del suono riscuotono presto successo tra i musicisti del periodo.
Successivamente a cavallo degli anni '40 e '50, anche a seguito all'avvento delle chitarre elettriche, in cui la cassa armonica era completamente assente e sostituita da un unico blocco di legno, si inizia a sperimentare con i suoni ottenibili dagli amplificatori, portandone all'estremo le possibilità. Alzando il volume ad alti livelli si crea una distorsione, cioè una compressione sul segnale che le valvole effettuano quando vanno in saturazione per via della loro dinamica limitata. Infatti se l'ampiezza dell'ingresso supera una certa soglia, che dipende dal tipo di valvola, dall'amplificatore e dalla tensione di alimentazione, non si può garantire un'amplificazione lineare, portando di conseguenza al "taglio" della parte di segnale eccedente il limite della valvola. Solitamente è un effetto indesiderato che si tende ad evitare per la maggior parte delle applicazioni musicali, ma che nel caso della chitarra elettrica risultava talmente efficace da diventarne parte integrante del suono stesso [3].

Nonostante possa essere considerata una tecnologia superata che può essere ricreata anche digitalmente tramite microprocessori e computer, la valvola conserva il suo fascino analogico che le ha consentito di resistere per tutto questo tempo nel campo della riproduzione audio, sia negli impianti hi-fi, sia appunto per rendere unico il suono della chitarra.

Adesso sono pertanto affrontati i passaggi che porteranno all'implementazione di un algoritmo in grado di emulare il comportamento di un amplificatore valvolare. In questo capitolo sono dati alcuni cenni teorici sulla valvola, esponendo di seguito l'attuale stato dell'arte circa l'emulazione di sistemi non lineari. È proposto poi un modello matematico tale da schematizzare la distorsione valvolare, sul quale si basa l'algoritmo in sé: successivamente è spiegato il funzionamento di quest'ultimo. Nel secondo vengono introdotti in breve gli strumenti utilizzati ovvero NU-Tech e Librerie Intel. Nel terzo capitolo è affrontata la parte relativa all'implementazione prima in MATLAB e poi nel quarto in NU-Tech, entrando nel dettaglio delle singole operazioni dove necessario. Infine nel quinto sono effettuati dei test sul plugin realizzato, in maniera da osservarne gli effetti su segnali audio di diversa natura.

1.2 Funzionamento delle valvole

La valvola è un componente elettronico attivo che sfrutta l'effetto termoionico per modulare in ampiezza un segnale elettrico [4], usata all'interno dei circuiti che compongono i vari stadi di un amplificatore. Questo è realizzato quindi con valvole che possono essere triodi o pentodi.

Il primo è composto da tre terminali: anodo, catodo e griglia. Il suo funzionamento in breve prevede la generazione di un flusso di elettroni (emessi per effetto termoionico) dal catodo verso l'anodo, controllato dalla tensione della griglia: il segnale è dato in ingresso proprio in quest'ultima, mentre l'uscita corrisponde alla corrente nell'anodo. Nel dominio del tempo, dato un ingresso di ampiezza elevata, la saturazione della valvola produce un clipping asimmetrico che taglia i picchi del segnale solamente nella metà a valori negativi. Al contempo nello spettro tale distorsione porta alla comparsa delle armoniche superiori di ordine pari e in particolare della seconda armonica, in aggiunta alla frequenza del segnale in ingresso.
Il pentodo ha invece cinque terminali, avendo introdotto due ulteriori griglie. A fronte di ingressi con ampiezze elevate si ottiene un segnale in uscita che presenta questa volta un clipping simmetrico, ovvero in cui entrambi i picchi positivi e negativi vengono compressi e tagliati. Nel dominio della frequenza si osservano invece diverse armoniche superiori di ordine dispari [5].

1.3 Stato dell'arte

L'amplificazione valvolare e la strumentazione analogica in generale sono ad oggi ancora preferite nel campo musicale per diverse motivazioni, soprattutto qualitative. Presentano però alcuni svantaggi: negli amplificatori per chitarra elettrica ad esempio peso, dimensione, costi e consumo elettrico risultano spesso essere eccessivi, accoppiati ad una scarsa durabilità delle valvole.
Per questo motivo il Digital Signal Processing è ampiamente usato nelle applicazioni audio, al fine di realizzare un'emulazione di dispositivi analogici che si avvicini quanto più possibile al suono caratterizzante questi ultimi, garantendo tutti i vantaggi della computazione digitale. Solo di recente si è riusciti ad ottenere dei risultati soddisfacenti per via delle non linearità introdotte dall'elettronica analogica, che necessita pertanto ai fini dell'emulazione di un'elaborazione digitale basata su modellazioni matematiche non lineari.

Il metodo più diretto è lo Static Waveshaping ovvero modellazione statica delle onde: consiste nell'applicare una mappatura non lineare istantanea e tempo invariante sulla variabile d'ingresso verso la variabile di uscita. La distorsione è ottenuta inviando il segnale in una funzione non lineare attraverso un fattore di scala. Ad esempio in [Araya Suyama 6] tale funzione è della forma:

$$f(x) = \frac{3x}{2}\left(1 - \frac{x^2}{3}\right)$$

dove $-1 < x < 1$; risultando relativamente lineare viene applicata tre volte in serie al segnale. In [Diodic 7] è proposta una funzione non lineare simmetrica della forma:

$$f(x) = (|2x| - x^2)\, sign(x)$$

dove $sign(x) = 1$ se $x > 0$, $sign(x) = -1$ altrimenti. Nello stesso [Diodic 7] si ha una funzione non lineare asimmetrica della seguente forma, alquanto complessa:

$$f(x) = \begin{cases} -\frac{3}{4}\{1 - [1 - (|x| - 0.032847)]^{12} + \frac{1}{3}(|x| - 0.032847)\} + 0.01 & \text{per } -1 \le x < -0.8905 \\ -6.153\, x^2 + 3.9375\, x & \text{per } -0.8905 \le x < 0.320018 \\ 0.630035 & \text{per } 0.0320018 \le x \le 1 \end{cases}$$

Tutte le funzioni non lineari appena proposte sono parte di brevetti anteriori risalenti agli anni novanta, le quali però mancano di una regione prettamente lineare e soprattutto di un controllo parametrico e dinamico. In figura se ne osservano le curve statiche caratteristiche.

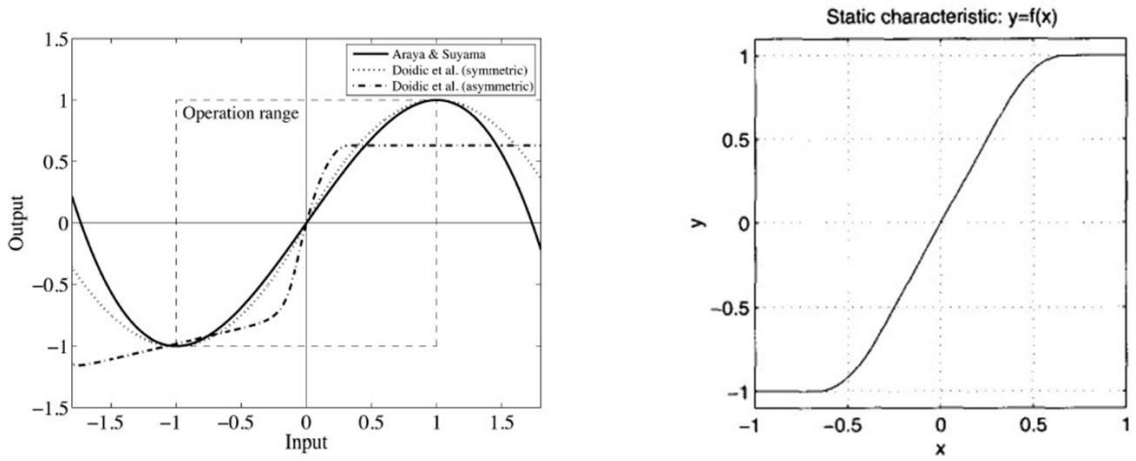


Figura 1: curve caratteristiche statiche delle funzioni lineari [6, 7] a sinistra; [8] a destra

Anche in [8] si propone un approccio alla modellazione non lineare che rientra nella tipologia del metodo di Static Waveshaping, mediante una funzione non lineare che realizza un clipping simmetrico del segnale. La forma è la seguente:

$$f(x) = \begin{cases} 2x \; per \; 0 \leq x \leq 1/3 \\ \dfrac{3-(2-3x)^2}{3} \; per \; 1/3 \leq x \leq 2/3 \\ 1 \; per \; 2/3 \leq x \leq 1 \end{cases}$$

Come si vedrà più avanti l'algoritmo oggetto di studio si andrà a basare proprio su questo metodo di modellazione statica delle onde. In particolare le scelta della funzione è influenzata dalla capacità di realizzare una distorsione asimmetrica fedele a quella tipica delle valvole a triodo, garantendo allo stesso tempo un controllo parametrico, una maggior semplicità dell'algoritmo e di consenguenza un minor impiego di potenza computazionale rispetto alle tecniche esposte di seguito.

Altri metodi disponibili sono quelli analitici, in cui rientra l'espansione in serie di Volterra [9, 10] cioè una rappresentazione dei sistemi basata su un'espansione non lineare della teoria dei sistemi lineari. Vi sono poi le tecniche basate sulla simulazione dei circuiti analogici: solitamente gli algoritmi di questo tipo sono implementati attraverso il software SPICE [11]. In questa categoria rientrano anche i Wave Digital Filters [12], una classe speciale di filtri con parametri mappati direttamente da quantità fisiche che consentono pertanto di calcolare le equazioni differenziali di reti elettriche.
Una proposta di recente teorizzazione vede l'utilizzo di una rete neurale artificiale [13], un modello computazionale usato solitamente nel campo dell'automatica che risulta particolarmente efficace per realizzare emulazioni. Con questo metodo l'errore del valore efficace tra il segnale elaborato tramite neural-network e segnale in uscita dall'amplificatore valvolare è minore di 1%.

Infine si possono valutare alcuni software disponibili sul mercato. Un esempio è dato dal plugin *Ace* prodotto dalla Shuttered Glass Audio e distribuito gratuitamente [14]. È basato sul circuito di un classico amplificatore valvolare anni '50, che presenta due triodi nello stadio di preamplificazione e una valvola di potenza: sono state aggiunte inoltre ulteriori funzioni come il controllo della quantità di feedback e il settaggio della frequenza di campionamento. Dispone di una propria interfaccia grafica dalla quale possono essere modificati i parametri ed impiegabile anche dal vivo grazie ad un uso ridotto della CPU.

Un altro plugin è il SDRR sviluppato dalla Klanghelm, acquistabile presso il loro sito [15]. Offre svariate funzionalità, in particolare un'emulazione digitale di due diversi modelli di stadi di preamplificazione valvolare combinabili tra loro. Tra le altre modalità disponibili, la DIGI mode consente di scegliere esattamente quali e quante armoniche superiori aggiungere al segnale, oppure la DESK mode che imita la saturazione tipica delle consolle di missaggio analogiche.

1.4 Un modello matematico per le valvole

Si propone adesso un metodo per emulare il comportamento degli amplificatori valvolari mediante Digital Signal Processing, implementando pertanto un algoritmo che sia in grado di realizzare un clipping asimmetrico sui valori d'ingresso basandosi su una funzione non lineare che andrà a verificare le seguenti condizioni [16].
La distorsione ottenuta, a fronte di ingressi di bassa ampiezza, dovrà essere tale da aggiungere in uscita nel dominio della frequenza la seconda armonica superiore tipica del triodo, mentre nel dominio del tempo le alterazioni sono contenute. Aumentando il livello dell'input, nello spettro appaiono ulteriori armoniche di ordine pari e dispari, con le prime che risultano di intensità maggiore, mentre i picchi positivi del segnale (ipotizzando una forma originariamente sinusoidale) iniziano a venire tagliati. Per ampiezze ancor maggiori si presenta infine un clipping anche sui picchi negativi, pur in quantità minori rispetto alle metà positive, verificando dunque l'asimmetria di questo tipo di distorsione. Nello spettro aumentano stavolta le armoniche di ordine dispari, con le altre che risultano comunque presenti.
La funzione non lineare proposta nell'approccio utilizzato è la seguente:

$$f(x) = \frac{x - Q}{1 - e^{-dist*(x-Q)}} + \frac{Q}{1 - e^{dist*Q}}, \qquad Q \neq 0, \qquad x \neq Q$$

I parametri di progettazione si basano su di un modello matematico per cui non deve avvenire alcuna distorsione a fronte di input di bassa ampiezza: infatti la derivata di $f(x)$ deve risultare $f'(0) \approx 1$ con $f(0) = 0$.

Di conseguenza la curva caratteristica statica associata è tale da tagliare e limitare i valori d'ingresso ampiamente negativi, mentre amplifica linearmente i valori positivi. Infatti spostando il punto di lavoro la funzione di trasferimento risulta più lineare per ingressi a bassa ampiezza.

La funzione richiede inoltre che siano verificate le condizioni $Q \neq 0,\ x \neq Q$ ma con dovuti accorgimenti è in grado di realizzare l'elaborazione del segnale d'ingresso anche nel caso in cui queste non siano vere. Nel primo caso, in cui il punto di lavoro Q è pari a zero, la funzione diventa della forma $f(x) = \frac{x}{1-e^{-dist(x)}}$. Se invece il valore dell'ingresso x è pari a quello del punto di lavoro Q, la sottrazione al numeratore risulta zero portando erroneamente a $f(x) = 0$. Quindi il risultato sarà calcolato come $z = \frac{1}{dist}$ se $Q = 0$, altrimenti come $z = \frac{1}{dist} + \frac{Q}{1-e^{dist(Q)}}$.

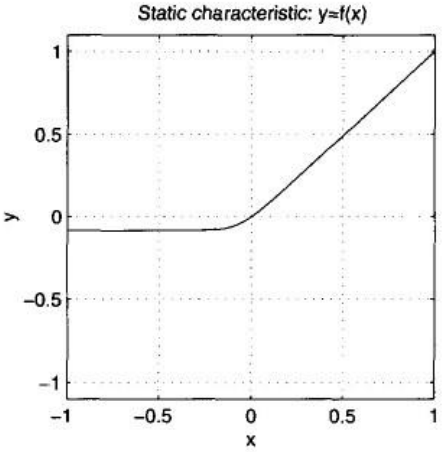


Figura 2: curva caratteristica statica del clipping asimmetrico per emulazione di amplificatori valvolari, con Q=-0.2 e dist=8

All'interno dell'algoritmo sono aggiunti inoltre due filtri passa alto e passa basso, rispettivamente per rimuovere la componente diretta dall'output e per dare forma alle armoniche più alte. Il filtro usato è del tipo Infinite Impulse Response (IIR) ovvero a risposta all'impulso infinita: sono definiti tali i sistemi per cui la risposta ad un impulso in input sarà non nulla per tempo che tende all'infinito per via della loro funzione di trasferimento razionale con coefficienti al numeratore e al denominatore. Nel caso in cui tale risposta impulsiva tenda a zero in un tempo finito parleremo di Finite Impulse Response (FIR) ovvero risposta all'impulso finita: in questo caso la funzione di trasferimento ha coefficienti solo al numeratore. Dunque la funzione su cui si basa è la seguente:

$$y[n] = \sum_{k=0}^{ord} b_k * x(n-k) - \sum_{k=1}^{ord} a_k * y(n-k)$$

dove $a_k = A_k/A_0$ e $b_k = B_k/A_0$ sono i coefficienti ridotti del filtro calcolati a partire da quelli iniziali $A_0 \dots A_{ord}$ e $B_0 \dots B_{ord}$ che sono stati passati alla funzione filtro come membri dei vettori argomento [17]. In particolare il valore dell'n-esimo campione dell'uscita y dipende da quelli precedenti, in quanto la struttura di tale filtro si basa quindi su di un feedback schematizzato come in figura.

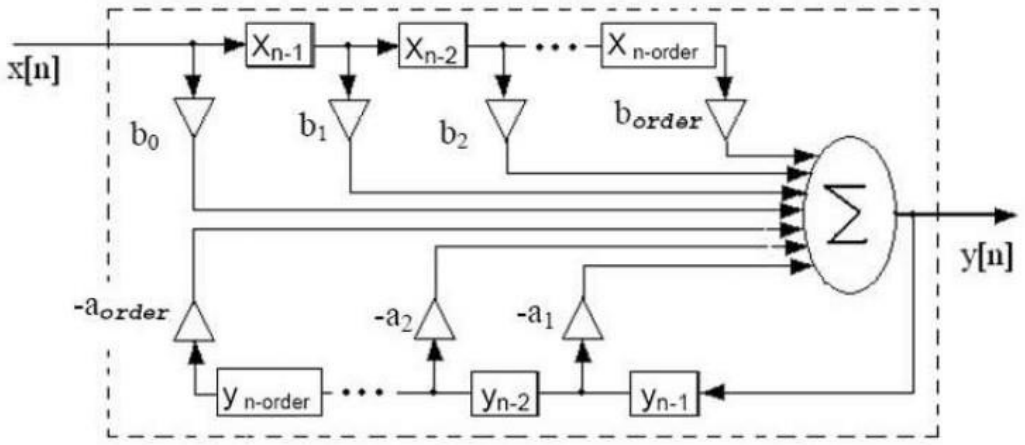


Figura 3: struttura di un filtro IIR in forma canonica

I parametri dell'algoritmo sono i seguenti:

- **x:** segnale d'ingresso.
- **Q:** punto di lavoro; controlla la linearità della funzione di trasferimento per ingressi a bassa ampiezza, risultando maggiormente lineare quanto più è negativo il suo valore.
- **gain:** quantità della distorsione. (> 0)
- **dist:** carattere della distorsione; un valore maggiore darà un clipping più accentuato. (> 0)
- **rh:** polo del filtro passa alto; $|rh| < 1$, solitamente ≈ 1.
- **rl:** polo del filtro passa basso; simula la capacitanza di un amplificatore valvolare; $|rl| < 1$.
- **mix:** quantità di suono processato; $0 < mix < 1$ dove 0 equivale al segnale d'ingresso mentre 1 equivale al segnale processato.

Inoltre è stato aggiunto un ulteriore parametro durante la fase di progettazione detto **dry/wet:** svolge sostanzialmente la stessa funzione di mix, posto però stavolta dopo i filtri, consentendoci di regolare in uscita la quantità di segnale d'ingresso e di segnale processato e filtrato.

1.5 Funzionamento dell'algoritmo

Si espone l'implementazione dell'algoritmo proposta nel testo di riferimento [18] in codice Matlab [19], linguaggio di programmazione per il calcolo numerico e al contempo anche un ambiente per l'analisi matematica e statistica. È necessario premettere che il segnale audio viene elaborato digitalmente sotto forma di un vettore di valori decimali, rappresentato nel codice da **x** per l'ingresso, **y** per l'uscita.

```
function y=tube_dist(x,gain,Q,dist,rh,rl,mix)

q=x*gain/max(abs(x));                  %Normalization
```

In primo luogo viene eseguita una normalizzazione per cui i valori dei campioni all'interno del vettore vengono moltiplicati per il guadagno e divisi per il valore massimo assoluto tra di essi, così da rientrare tutti in un range compreso tra **-gain** e **gain**.

```
if Q==0
    z=q./(1-exp(-dist*q));
        for i=1:length(q)                  %Test because of the
            if q(i)==Q                     %transfer function's
            z(i)=1/dist;                   %0/0 value in Q
            end
        end
else
    z=(q-Q)./(1-exp(-dist*(q-Q)))+(Q/(1-exp(dist*Q)));
    for i=1:length(q)
        if q(i)==Q
           z(i)=1/dist+Q/(1-exp(dist*Q));
        end
    end
end
```

Come detto in precedenza la funzione richiede come condizioni che $Q \neq 0$ e $x \neq Q$, ma l'algoritmo appunto è in grado di realizzare l'elaborazione anche quando queste vengono meno. Infatti è posto un *if* che verifica se il punto di lavoro è pari a zero: in tal caso il segnale viene calcolato mediante la funzione $f(x) = \frac{x}{1-e^{-dist(x)}}$, altrimenti si procede con la sua forma $f(x) = \frac{x-Q}{1-e^{-dist(x-Q)}} + \frac{Q}{1-e^{dist(Q)}}$. In entrambe le situazioni si procede poi attraverso un ciclo *for* a controllare che all'interno del frame normalizzato non vi siano valori che coincidono con quello di **Q**: se così fosse si sostituisce l'i-esimo indice del vettore del frame elaborato con il risultato di $z = \frac{1}{dist}$ se $Q = 0$, altrimenti di $z = \frac{1}{dist} + \frac{Q}{1-e^{dist(Q)}}$.

```
y=mix*z*max(abs(x))/max(abs(z))+(1-mix)*x;
y=y*max (abs (x) ) /max(abs (y)) ;

y=filter( [1 -2 1], [1 -2*rh rh^2] ,y);    %HP filter
y=filter( [1-rl], [1 -rl] ,y);             %LP filter
```

A questo punto, le restanti operazioni da svolgere sono il missaggio e il filtraggio. Nella prima input e output vengono moltiplicati rispettivamente per **1-mix** e **mix** ed infine sommati tra loro, così da ottenere un segnale che sia dato dalla combinazione dei due.
Questo stesso segnale viene poi fatto passare attraverso i filtri passa alto e passa basso, che vanno di fatto a rimuoverne alcune frequenze. Per realizzare tali filtri in Matlab è usata la funzione *filter*, la quale richiede come argomenti una coppia di vettori che contengono rispettivamente i coefficienti al numeratore e al denominatore della relativa funzione di trasferimento razionale. Il valore dei coefficienti dipende dalla posizione dei poli del filtro **rh** e **rl**.
Il passaggio finale, non presente in quanto aggiunto in seguito, è quello di **dry/wet**, che funziona esattamente come il missaggio ma questa volta tra segnale d'ingresso e segnale elaborato e filtrato.

Nella figura successiva un diagramma di flusso che rappresenta graficamente la successione logica delle operazioni compiute all'interno dell'algoritmo in questione.

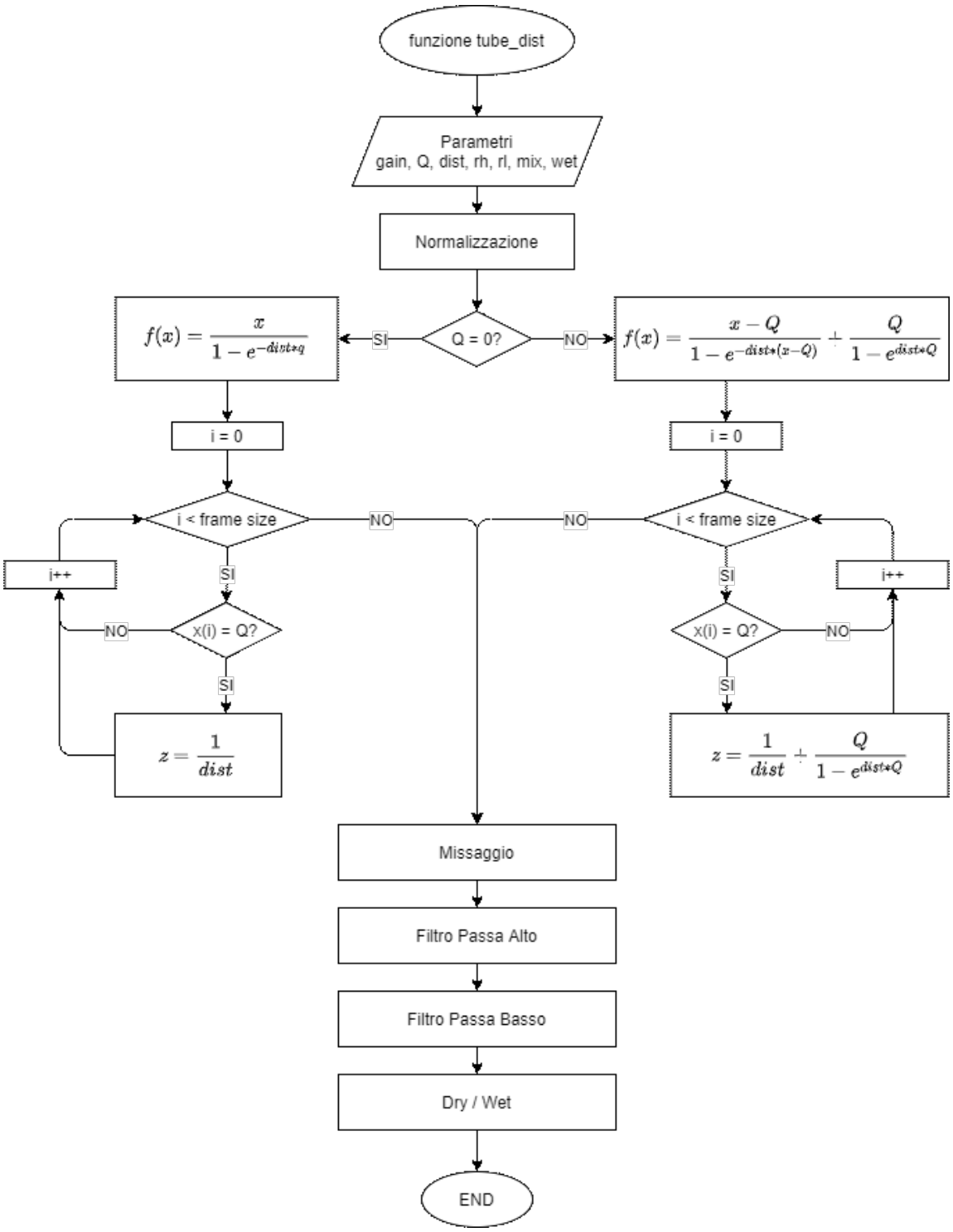


Figura 4: flowchart dell'algoritmo

2. Tools

2.1 NU-Tech

NU-Tech è una piattaforma di Digital Signal Processing (DSP) sviluppata dalla Leaff Engineering basata su di un'architettura a PlugIn, tale da consentire di testare e applicare algoritmi complessi per l'elaborazione dei segnali mediante un PC e una scheda audio. Lo sviluppatore avrà pertanto la possibilità di scrivere i propri NUTS (NU-Tech Satellites) in codice sorgente C/C++ così che una volta compilati possano essere posti nell'interfaccia grafica del programma ed utilizzati mandandovi in ingresso un segnale audio.

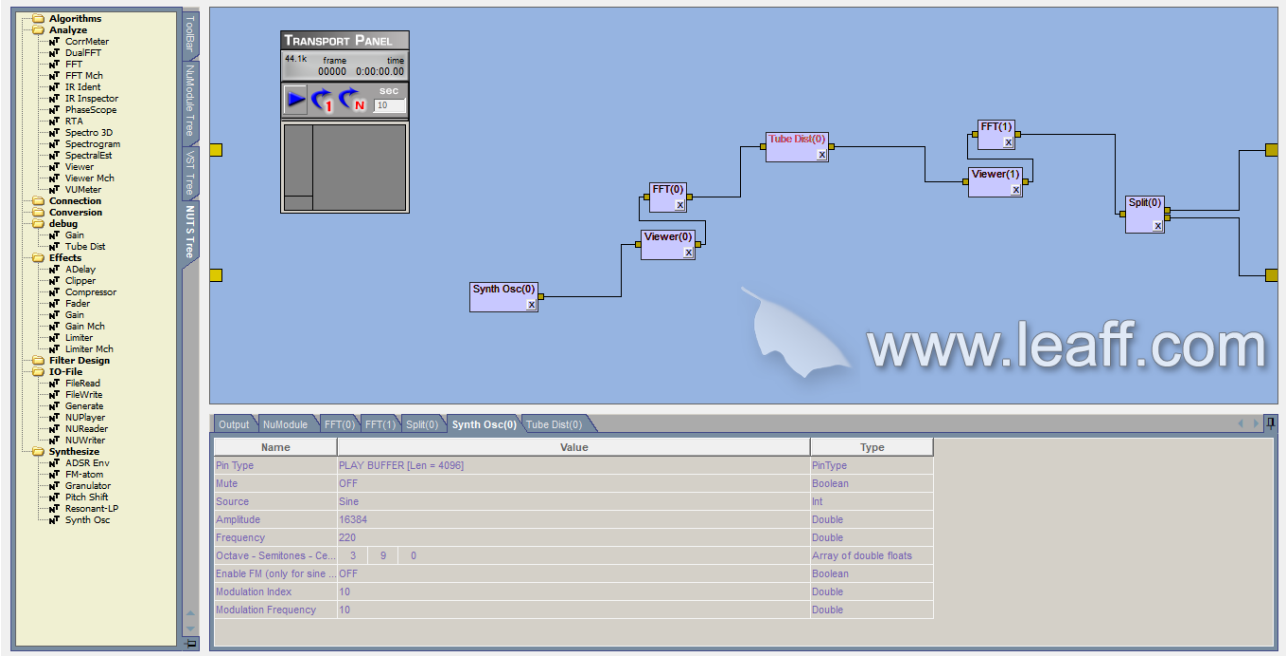


Figura 5: GUI di NU-Tech in un esempio di funzionamento

Prima di proseguire nell'elencarne le caratteristiche più specifiche, è bene fare un breve discorso generale relativo ai PlugIn: in informatica con questo termine si intendono solitamente delle componenti software non autonome, che vengono pertanto integrate all'interno di altri programmi al fine di estenderne le funzionalità originarie; in particolare nel campo delle applicazioni audio per computer, questi elementi saranno adibiti all'elaborazione dei segnali digitali o anche alla sintesi sonora. L'aspetto più interessante riguarda le infinite possibilità di customizzazione che tale struttura offre: infatti, oltre ai PlugIn sviluppati dalle principali aziende nel mercato dei software DSP, è possibile scrivere il proprio codice per realizzare la funzione che si desidera incorporare all'interno della piattaforma.

Come già accennato, all'interno del programma utilizzato per questo progetto, potremo dunque implementare dei nuovi elementi per l'elaborazione dei segnali, i quali verranno detti NUTS. Questi condividono comunque le principali caratteristiche dei plugin: infatti si basano su di uno standard DLL, cioè libreria a collegamento dinamico, tale che le funzioni contenute in essa non risultino caricate interamente in fase di compilazione, bensì direttamente in fase di esecuzione. Possono essere dotati di un'interfaccia grafica personalizzabile nella quale si modificano i parametri in tempo reale, anche se

non strettamente necessaria in quanto la stessa operazione è realizzabile attraverso un pannello integrato nel programma. Riguardo il passaggio di informazioni in entrata ed in uscita dal NUTS, sono presenti delle porte dette pin: queste sono settabili a seconda del tipo di dato trasferito, a patto che risulti una coerenza di tipo tra gli elementi messi in comunicazione tra di loro, che dovranno pertanto scambiarsi un segnale rigorosamente dello stesso tipo.

Sono ora esposte le principali funzionalità della piattaforma NU-Tech. In primo luogo, il suo fine è quello di elaborare in tempo reale un segnale audio mandato in ingresso al computer tramite scheda audio. Pertanto per gestire il flusso del segnale, il programma si basa su una logica a buffer. Relativamente a questi ultimi si osserva quindi che l'elaborazione realizzata avviene "a frame": l'input infatti non potrà essere trattato interamente, né tantomeno campione per campione, richiedendo pertanto che i suoi valori vengano suddivisi in più frame, ciascuno dei quali verrà copiato in un buffer di ingresso, processato ed infine scritto in un buffer di uscita. La dimensione di questi elementi influenza in particolare le prestazioni del programma, in quanto se il frame size è eccessivamente grande la CPU impiega più tempo a trasferire i dati incrementando la latenza, mentre se invece è troppo piccolo vi saranno un gran numero di trasferimenti tra CPU e memoria che richiederanno un maggior sforzo computazionale.

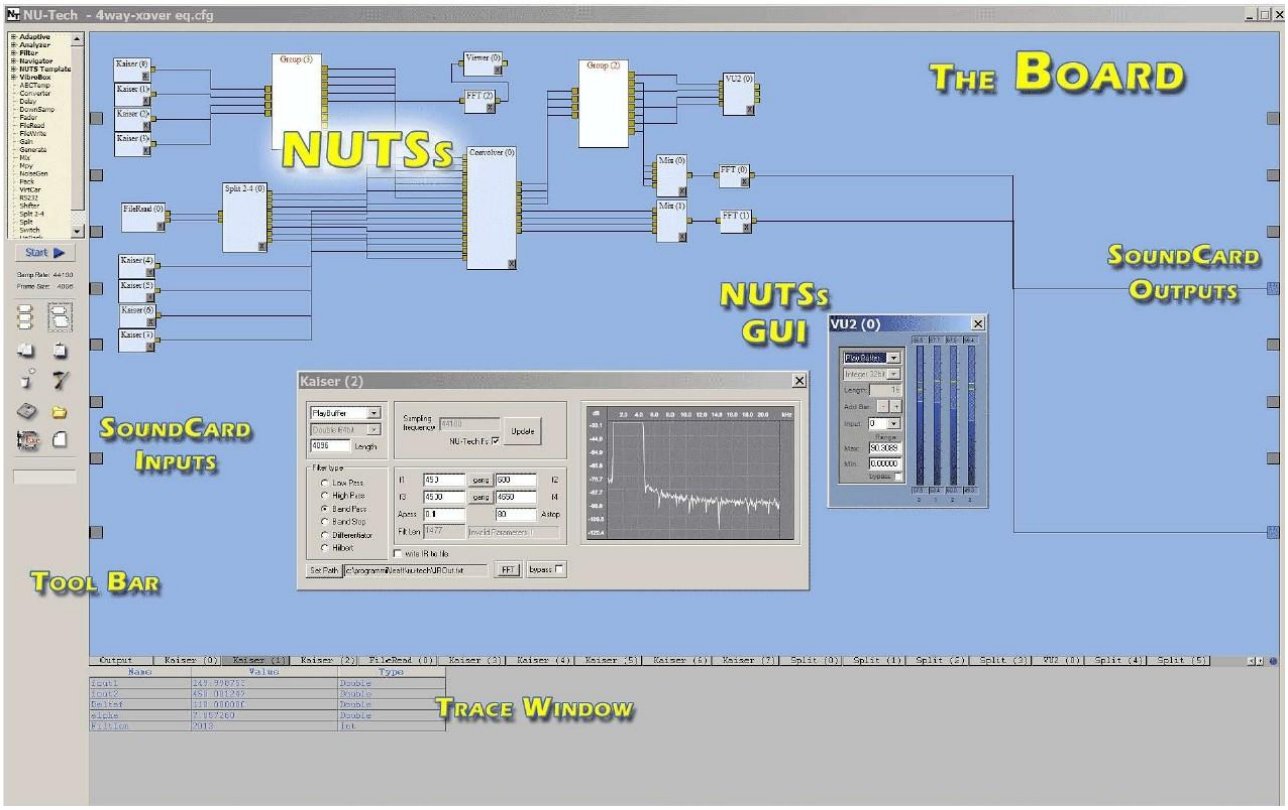


Figura 6: elementi dell'interfaccia grafica di NU-Tech

Il software ha una struttura dell'interfaccia grafica molto intuitiva che semplifica le operazioni e in figura ne sono evidenziato gli elementi principali: il tutto si basa prevalentemente sulla Board, sulla quale possiamo posizionare e collegare tra loro i NUTS. Questi possono essere selezionati dalla toolbar a sinistra e trascinati sulla Board, dove appariranno sotto forma di blocchi con il rispettivo nome e dei pin input e output, il cui numero e tipo di dato sarà settabile; ricordando il criterio di coerenza dei tipi di dato, è possibile quindi collegare più NUTS tra di loro creando una vera e propria rete.

NU-Tech dispone di diverse modalità: le prime due, ASIO e DirectSound, si basano su interfacce software ai driver hardware che consentono pertanto di utilizzare ingressi e uscite del computer anche in assenza di schede audio esterne. Infatti agli estremi della Board compaiono dei pin collegati direttamente a microfono e altoparlanti, che sono però assenti nella modalità Offline: questa infatti funziona senza il supporto dell'hardware, offrendo quindi la possibilità di modificare la struttura del buffer e in particolare la sua dimensione. Infine si hanno le modalità Trigger (adatta all'elaborazione di segnali video in cui la sincronizzazione con il segnale audio è essenziale) e Remote Control (che consente di connettersi ad un dispositivo presente sulla stessa rete).

Cliccando su di un NUTS, si possono visualizzare e modificare i valori dei suoi parametri costruttivi o attraverso la finestra con l'interfaccia grafica realizzata appositamente per esso, oppure in maniera più semplice attraverso il RealTime Watch, pannello posto in basso che ci consente di realizzare la stessa operazione senza doverci preoccupare della programmazione di una GUI. Infine il Transport Panel, il quale consente di avviare e interrompere lo stream dei dati.

2.2. Librerie Intel

Le Intel Signal Processing Libraries [20] sono librerie di funzioni per l'elaborazione dei segnali sviluppate dalla Intel, risultando pertanto ottimizzate per computer basati su architettura Intel: consentono di realizzare operazioni complicate in maniera efficiente come ad esempio elaborazione di vettori e matrici anche complessi, finestrature, trasformate di Fourier continue e discrete, convoluzioni, correlazioni, filtraggi e così via. Nascono nel 1995 come NSP (Native Signal Processing) con l'intento di consentire a tutti i PC con processori Intel di far girare dei codici time-sensitive indipendentemente dal sistema operativo: in questo modo non era più strettamente necessario avere dei microprocessori appositi per l'elaborazione dei segnali (ovvero i DSP) ma bastava un unico potente processore centrale in grado pertanto di realizzare grazie a tali librerie di lavorare in tempo reale su audio e video. Dal 2007 in poi verranno sostituite dalle IPP (Integrated Performance Primitives).

Una particolarità che contraddistingue tali librerie è la diversa definizione dei tipi di dato: infatti avranno tutti il prefisso Ipp seguito dal numero di byte e una lettera per identificarlo, ad esempio il tipo double diventerà Ipp64f, dove f indica tra l'altro che il valore è memorizzato in virgola mobile. Una novità è rappresentata dai tipi di dato che contengono numeri complessi, contraddistinti dal suffisso c (ad esempio Ipp64fc): in verità però altro non sono che delle struct contenenti due valori, uno per la componente reale e uno per quella immaginaria accessibili singolarmente attraverso l'operatore punto.

Anche le funzioni stesse dovranno indicare nel loro nome il tipo di dato che restituiscono in uscita utilizzando la sintassi appena descritta: ad esempio l'operazione tale da sommare una costante ad ogni elemento di un vettore, posti entrambi di tipo double, sarà ippsAddC_64f. La funzione appena descritta però è tale da memorizzare il vettore risultante su di un nuovo array: nel caso in cui si desideri che l'operazione venga svolta direttamente sul vettore in ingresso, si deve aggiungere il suffisso _I che sta appunto per "in place".

3. Implementazione MATLAB

In seguito è stato implementato uno script Matlab per testare il comportamento della funzione. Lo script accetta in ingresso un segnale audio di vario tipo (come ad esempio un rumore bianco, un tono puro, una sovrapposizione di armoniche, una traccia audio) e lo processa suddividendolo in frame, per replicare lo stesso comportamento a buffer presente in NU-Tech.

```
[x, Fs] = audioread(filename);
info = audioinfo(filename)
t = 0:seconds(1/Fs):seconds(info.Duration);
t = t(1:end-1);
plot(t,x)
xlabel('Time')
ylabel('Audio Signal')

L=length(t);
n = 2^nextpow2(L);
Y=fft(x,n);
f = Fs*(0:(n/2))/n;
P = abs(Y/n);
plot(f,P(1:n/2+1))
xlabel('Frequency')
ylabel('|P(f)|')

gain=5; Q=-1; dist=11;
rh=0.995; rl=0.5; mix=1;
len=126;

z=buffer(x,len);
buff_size=length(z);
frame=zeros(len,1);
w=zeros(len,buff_size);

for i=1:buff_size
    frame(1:len)=z(1:len,i);
    temp = tube_dist(frame,gain,Q,dist,rh,rl,mix);
    w(1:end,i)=temp;
end
y=reshape(w,[],1);

audiowrite(filename,y,Fs);
[y] = audioread(filename);
plot(t,y)
xlabel('Time')
ylabel('Audio Signal')

Y=fft(y,n);
f = Fs*(0:(n/2))/n;
P = abs(Y/n);
plot(f,P(1:n/2+1))
xlabel('Frequency')
ylabel('|P(f)|')
sound(y,Fs)
```

Il comando plot consente realizzare dei grafici del segnale prima e dopo l'elaborazione, sia nel dominio del tempo sia in particolare nel dominio della frequenza attraverso una Trasformata di Fourier (FFT): in tal modo si potranno osservare le armoniche superiori aggiunte dalla distorsione asimmetrica.
Come anticipato, lo script dovrà elaborare il segnale d'ingresso in frame e per farlo utilizzerà il comando buffer [21]: questo infatti scompone il vettore x dell'input in una matrice il cui numero di righe sarà pari al parametro len settabile, così che ogni colonna rappresenti un frame di dimensione len appunto. Una volta fatto ciò, attraverso un ciclo for che va da 0 a buff_size (cioè il numero di colonne della matrice), per ogni iterazione si andranno a leggere i valori contenuti nell'i-esima colonna, copiandoli in un vettore di lunghezza len e passandolo infine alla funzione tube_dist; il risultato verrà quindi "appoggiato" in un vettore temp così che questo possa essere infine copiato nell'i-esima colonna della matrice del buffer di uscita. Al termine del ciclo il comando reshape [22] pone tutti gli elementi della matrice in un unico vettore di uscita y che rappresenta pertanto il segnale elaborato: si conclude salvandolo in memoria, ricaricandolo e riproducendolo.

4. Implementazione NU-Tech

Il codice sorgente per l'implementazione di un NUTS richiede una struttura precisa, basata in primo luogo sull'oggetto PlugIn che viene dichiarato con i suoi membri e metodi in un file header e successivamente implementato in un file cpp separato. A livello di struttura logica lo schema classico utilizzato è il seguente:

- **Costruttore:** è chiamato in corrispondenza del posizionamento del NUTS sulla Board ed è addetto alla creazione dell'oggetto e inizializzazione delle variabili.
- **Init:** è chiamata in corrispondenza dell'avvio dello streaming del segnale; è buona norma allocare memoria per gli array dinamici in questa fase.
- **Process:** è la funzione che compie la vera e proprio elaborazione del segnale, richiamata per ogni frame passato in ingresso.
- **Delete:** è chiamata in corrispondenza del termine dello streaming del segnale; andrà pertanto a disallocare lo spazio di memoria riservato precedentemente agli array dinamici.
- **Distruttore:** è infine chiamato in corrispondenza dell'eliminazione del NUTS dalla Board e va a liberare la memoria occupata dall'intero oggetto PlugIn.

Si approfondirà ora nel dettaglio l'implementazione della sola Process, ma circa le restanti funzioni è necessario sapere che le variabili con cui lavoreranno saranno le seguenti, definite così nell'header:

```
int FrameSize;
double gain, dist, Q, rh, rl, mix, wet;
double* temp1, * temp2, * q, * z;
double* taps_hpf, * taps_lpf, * pDly_hpf, * pDly_lpf;
```

I parametri della funzione introdotti in precedenza (gain, dist, etc.) sono stati implementati in maniera tale da poterne modificare i valori in tempo reale attraverso l'RT Watch e inoltre memorizzarli quando si salverà una configurazione di NUTS sotto forma di un file XML; nella riga successiva si trovano invece gli array dinamici definiti mediante puntatori che vengono utilizzati durante la Process per realizzare l'elaborazione del frame del segnale conservato sotto forma di vettore all'interno del buffer. A questo proposito si osserva che FrameSize è una variabile intera a cui si fa ricorso in fase di allocazione di memoria di tali array, in quanto verrà valorizzato con il comando `CBFunction(this, NUTS_GET_FS_SR, 0, (LPVOID)AUDIOPROC)` che restituisce la dimensione del buffer, consentendo pertanto di inizializzare i suddetti alla sua stessa lunghezza. È bene ricordare che tale valore può essere modificato solamente andando a modificare la dimensione stessa del buffer dalle impostazioni di NU-Tech avviato in modalità Offline: infatti nella modalità DirectSound verrà usata la specifica di base del driver audio, e di conseguenza il tipo di dato PlayBuffer [23] trarrà le sue definizioni a partire da questo, risultando in una dimensione del frame pari a 4096 campioni.
Per gli elementi taps_hpf e taps_lpf occorre invece un discorso a parte, in quanto rappresentano i vettori di valori da passare in ingresso rispettivamente ai filtri passa alto e passa basso, i quali dipenderanno dai poli rh e rl: sono stati quindi implementati in maniera tale che, nel caso in cui venissero modificati i valori di queste ultime due variabili attraverso la RT Watch, andranno a modificarsi di conseguenza anche i valori all'interno dei vettori, sia durante lo streaming che prima.

Sempre relativamente ai filtri si hanno infine pDly_hpf e pDly_lpf, i quali conservano i valori della delay line: sono definiti come array dinamici ma in realtà la loro dimensione è pari all'ordine dei coefficienti del filtro, pertanto saranno composti rispettivamente da due elementi e da uno solo.

Di seguito i passaggi contenuti all'interno di Process, equivalenti alla funzione tube_dist di Matlab:

```
int __stdcall PlugIn::LEPlugin_Process(PinType
**Input,PinType **Output,LPVOID ExtraInfo)
{
 double* InputData = ((double*)Input[0]->DataBuffer);
 double* OutputData = ((double*)Output[0]->DataBuffer);

 ippsDivC_64f_I(CONV_VALUE, InputData, FrameSize)
```

Dopo il primo passaggio in cui si assegnano ai puntatori InputData e OutputData gli indirizzi di memoria in cui si trovano i DataBuffer di ingresso ed uscita, si applica un'operazione che consente di rendere decimali i valori dei campioni del segnale d'ingresso. Infatti in file audio con codifica a 16 bit ogni sample ha un valore compreso tra -2^{15} e 2^{15}; pertanto al fine di poter compiere correttamente le successive operazioni e soprattutto testare che i valori ottenuti risultino uguali a quelli calcolati in Matlab, sarà necessario realizzare una divisione dei singoli valori del buffer d'ingresso per una costante, che in questo caso è stata definita come `#define CONV_VALUE 32768` cioè 2^{15}. La funzione utilizzata è quindi IppsDivC_64f_I [24], la quale fa parte delle librerie Intel e svolge appunto l'operazione sopra descritta, accettando come argomenti la costante CONV_VALUE, il puntatore all'array InputData e la dimensione di quest'ultimo data da FrameSize. La stessa operazione tra l'altro è realizzata in maniera inversa al termine del tutto, moltiplicando pertanto per CONV_VALUE tutti i valori decimali nel buffer d'uscita indicato dal puntatore OuputData.

```
 ippsMulC_64f(InputData, gain, q, FrameSize);
 ippsMaxAbs_64f(InputData, FrameSize, temp1);
 ippsDivC_64f_I(temp1[0], q, FrameSize); /*NORMALIZATION*/

 if (Q == 0)
 {
       ippsMulC_64f(q, -dist, temp1, FrameSize);
       ippsExp_64f_I(temp1, FrameSize);
       ippsSubCRev_64f_I(1, temp1, FrameSize);
       ippsDiv_64f(temp1, q, z, FrameSize);

       for (int i = 0; i < FrameSize; i++)
       {
             if (q[i] == Q)
             {
                   z[i] = 1 / dist;
             }
       }
 }
 else
 {
       ippsSubC_64f(q, Q, temp1, FrameSize);
       ippsMulC_64f(temp1, -dist, temp2, FrameSize);
       ippsExp_64f_I(temp2, FrameSize);
       ippsSubCRev_64f_I(1, temp2, FrameSize);
```

```
            ippsDiv_64f(temp2, temp1, z, FrameSize);
            ippsAddC_64f_I(Q /(1-exp(dist * Q)), z, FrameSize);

            for (int i = 0; i < FrameSize; i++)
            {
                  if (q[i] == Q)
                  {
                        z[i] = 1 /dist + Q /(1 - exp(dist * Q));
                  }
            }
      }
      ippsMaxAbs_64f(InputData, FrameSize, temp1);
      ippsMaxAbs_64f(z, FrameSize, temp2);
      ippsMulC_64f_I(mix, z, FrameSize);
      ippsMulC_64f(z,temp1[0]/temp2[0], OutputData, FrameSize);
      ippsMulC_64f(InputData, 1 - mix, temp1, FrameSize);
      ippsAdd_64f_I(temp1, OutputData, FrameSize);

      ippsMaxAbs_64f(InputData, FrameSize, temp1);
      ippsMaxAbs_64f(OutputData, FrameSize, temp2);
      ippsMulC_64f_I(temp1[0]/temp2[0], OutputData, FrameSize);
```

Le funzioni della libreria Intel consentono di realizzare in maniera veloce ed efficace delle operazioni matematiche su vettori e matrici, in questo caso ricalcando gli stessi passaggi visti in Matlab per l'emulazione della distorsione asimmetrica tipica degli amplificatori valvolari. In particolare si ritrova l'equazione $f(x) = \frac{x-Q}{1-e^{-dist(x-Q)}} + \frac{Q}{1-e^{dist(Q)}}$, ma per realizzarla occorrono stavolta svariate funzioni. Per prima cosa viene sottratto ad ogni elemento del frame in ingresso (che nel frattempo si è spostato sull'array dinamico q a seguito della normalizzazione) il punto di lavoro Q, copiando il risultato nell'array temp1; i suoi elementi verranno poi moltiplicati per la costante -dist e copiati a loro volta in temp2; qui si realizzano due funzioni in place, una per calcolarne l'esponenziale e una per andare a sottrarre ad 1 ogni suo elemento. Pertanto in temp1 e temp2 si hanno rispettivamente il numeratore e il denominatore della frazione a sinistra dell'equazione: questa corrisponde ad una divisione tra due vettori realizzata da IppsDiv_64f, il cui risultato è copiato nell'array z. A questo punto, grazie all'inclusione della libreria math.h, si realizza l'ultimo passaggio rappresentato dalla somma tra z e la costante $\frac{Q}{1-e^{dist(Q)}}$ calcolata direttamente come Q/(1-exp(dist*Q)).

4.1 Filtri

```
//    HPF
IppsIIRState_64f* pIIRState_Hpf = NULL;
ippsIIRInit_64f(&pIIRState_Hpf, taps_hpf, 2, pDly_hpf, pBuf_Hpf);
ippsIIR_64f_I(OutputData, FrameSize, pIIRState_Hpf);
ippsIIRGetDlyLine_64f(pIIRState_Hpf, pDly_hpf);

//    LPF
IppsIIRState_64f* pIIRState_Lpf = NULL;
ippsIIRInit_64f(&pIIRState_Lpf, taps_lpf, 1, pDly_lpf, pBuf_Lpf);
ippsIIR_64f_I(OutputData, FrameSize, pIIRState_Lpf);
ippsIIRGetDlyLine_64f(pIIRState_Lpf, pDly_lpf);

//    DRY/WET
ippsMulC_64f(OutputData, wet, temp1, FrameSize);
ippsMulC_64f(InputData, 1 - wet, temp2, FrameSize);
ippsAdd_64f(temp1, temp2, OutputData, FrameSize);

ippsMulC_64f_I(CONV_VALUE, OutputData, FrameSize);

return COMPLETED;
}
```

Relativamente all'implementazione tramite librerie Intel dei filtri si illustrano ora le componenti e i passaggi necessari da compiere prima di realizzare l'operazione in sé: infatti questo elemento si baserà su una struttura detta State, che dovrà essere inizializzata su di un buffer esterno. Prima ancora è necessario anche definire tale buffer, che per questioni di ottimizzazione dello spazio è stato dichiarato nell'header e allocato in memoria durante la fase di Init del NUTS, avendone preventivamente calcolato attraverso la funzione ippsIIRGetStateSize [25] la dimensione, che dipende dall'ordine dei coefficienti della funzione di trasferimento.

Solo una volta fatto ciò è possibile chiamare all'interno del Process la funzione ippsIIRInit [26], che vuole come argomenti i puntatori alla struttura state, al buffer e all'array dei "taps": questo era stato introdotto in precedenza nelle dichiarazioni e altro non è che il vettore contenente i coefficienti della funzione di trasferimento razionale che definisce il filtro IIR. Tale vettore avrà $2 * (ordine + 1)$ elementi in quanto i valori al numeratore e al denominatore sono posti al suo interno consecutivamente: praticamente, data una funzione di ordine 1, se in precedenza nel codice Matlab venivano passati due vettori distinti $[A_0, A_1]$ e $[B_0, B_1]$ come argomenti della funzione filter, stavolta saranno memorizzati in taps direttamente come $[A_0, A_{1,} B_0, B_1]$.

Come già osservato, i valori dei coefficienti dipendono dal posizionamento dei poli rh e rl nel filtro, quindi in fase di allocazione in memoria di tale array dinamico è stato tenuto conto della possibilità di poter modificare questi ultimi in tempo reale attraverso l'RT Watch: infatti in SetRHValue -funzione che gestisce l'aggiornamento del valore della variabile in questione- è stata posta un'ulteriore condizione, la quale verifica se l'array è diverso da zero. In questo modo si impedisce che il comando memcpy acceda ad una posizione di memoria riservata, che non è stata ancora allocata per contenere l'array taps_hpf.

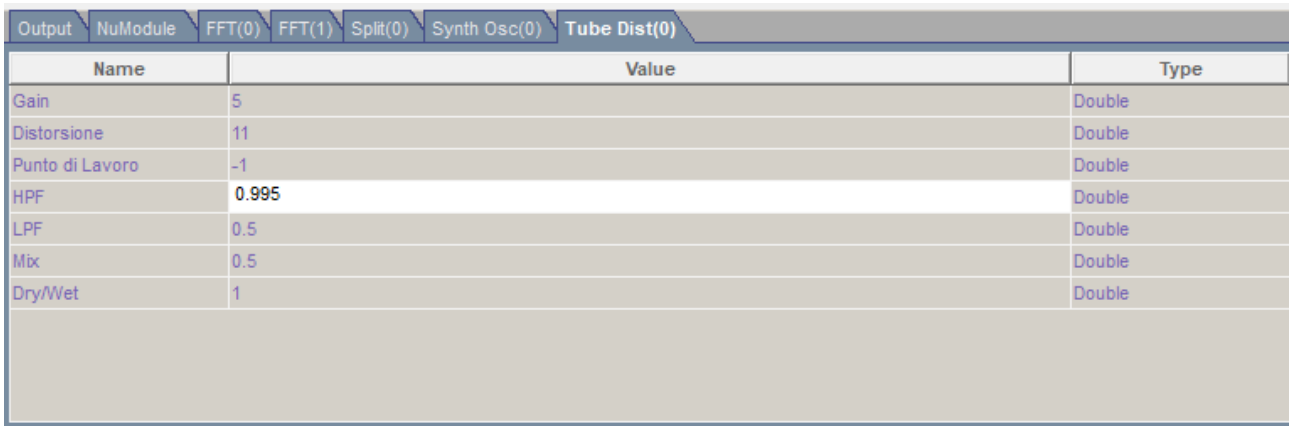


Figura 7: esempio di modifica del valore della variabile rh mediante RT Watch

Resta da realizzare un ultimo fondamentale passaggio, ovvero l'implementazione di una delay line per i filtri: infatti così come sono stati descritti finora risulteranno tali da generare delle incongruenze nelle uscite. NU-Tech si basa su di un'elaborazione a tempo reale frame per frame, in cui il vettore del segnale d'ingresso è suddiviso in più blocchi di stessa dimensione che verranno processati singolarmente: questo approccio però collide con i filtri IIR che sono pensati per lavorare solitamente su di un intero array di informazione audio considerandone i valori per ogni campione, i quali pertanto se messi in questa situazione elaboreranno i blocchi consecutivi come se fossero indipendenti tra di loro. La conseguenza nell'output è che al passaggio tra un frame e l'altro si verificheranno dei picchi con cadenza costante, ben visibili nei grafici nel dominio del tempo e in particolare distintamente udibili, producendo un effetto decisamente sgradito.
La possibile soluzione per realizzare una convoluzione a blocchi nei filtri a risposta finita FIR è data dai metodi di overlap&add e overlap&save, in cui la suddivisione in frame prevede una sovrapposizione tale da avere dei campioni in comune tra i blocchi consecutivi, mentre nelle uscite tali valori verranno rispettivamente sommati o trascurati ricomponendo pertanto un unico vettore [27].

Quello che occorre fare invece nei filtri IIR -sfruttando le funzioni messe a disposizione dalle librerie Intel- sarà l'aggiunta di una "memoria" tale che il filtraggio di un certo frame tenga conto delle condizioni iniziali poste proprio dai risultati del filtraggio del frame immediatamente precedente, ovvero una delay line.
La sua implementazione è in realtà piuttosto semplice: in primo luogo si dovrà effettuare la chiamata alla funzione ippsIIR, la quale prendendo come argomenti il puntatore alla struttura State del filtro e il puntatore all'array del frame, realizza appunto un filtraggio di quest'ultimo. In seguito verrà chiamata la funzione ippsIIRGetDelayLine [28], il cui ruolo è quello di calcolare i valori della linea di ritardo associata allo stato del filtro appena utilizzato e quindi memorizzarli in un array dinamico, la cui dimensione è pari all'ordine dei coefficienti della funzione di trasferimento del filtro. Questi verranno utilizzati solamente nella fase di filtraggio del frame successivo: infatti nelle operazioni di inizializzazione del filtro oltre agli argomenti citati in precedenza, verrà introdotto anche il puntatore all'array con i valori della delay line del frame precedente, così da porre le condizioni di partenza per l'elaborazione e risultando complessivamente in un filtraggio a blocchi efficiente.

5. Test su diversi tipi di segnali

In questo capitolo si valuterà l'algoritmo in azione in NU-Tech con segnali di varia natura per osservare il suo comportamento nel dominio del tempo e della frequenza. Si hanno pertanto grafici in cui i diversi ingressi saranno tutti elaborati con i parametri fissati ai seguenti valori:

- **Q:** -1
- **gain:** 5
- **dist:** 11
- **rh:** 0.995
- **rl:** 0.5
- **mix:** 1
- **wet:** 1

5.1 Rumore Bianco

Il rumore bianco è un segnale casuale caratterizzato da una distribuzione uniforme dell'energia nello spettro delle frequenze. In particolare è stato usato uno stesso segnale generato dalla funzione randn in Matlab al fine di verificare la conformità dei risultati ottenuti dalla funzione in entrambi i programmi, potendo inoltre osservare come i grafici siano pressoché identici.

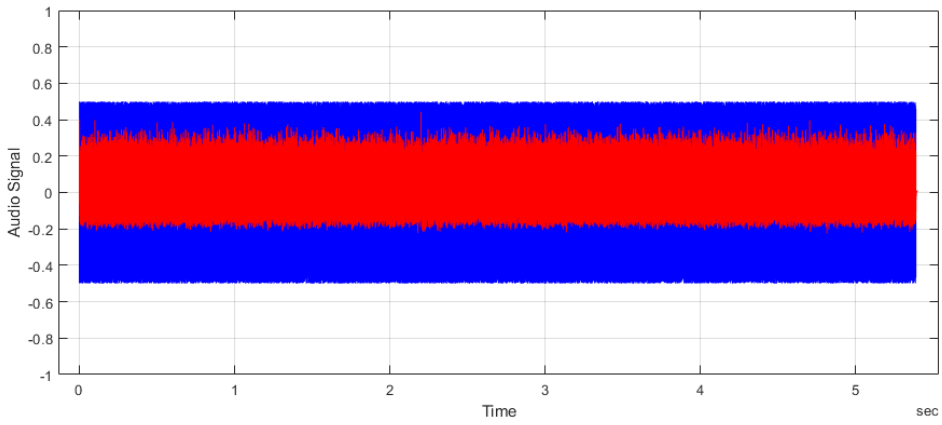


Figura 8: grafico nel dominio del tempo del rumore bianco, in Matlab. In blu l'input, in rosso l'output

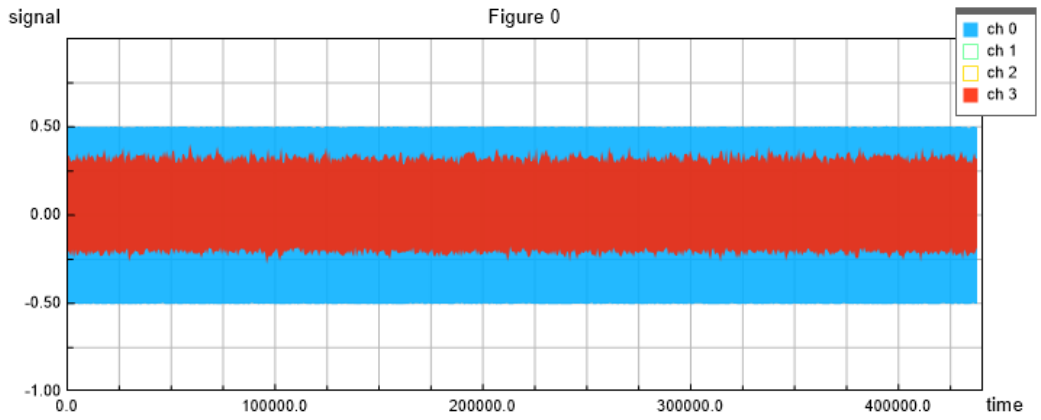


Figura 9: grafico nel dominio del tempo del rumore bianco, in NU-Tech. In blu l'input, in rosso l'output

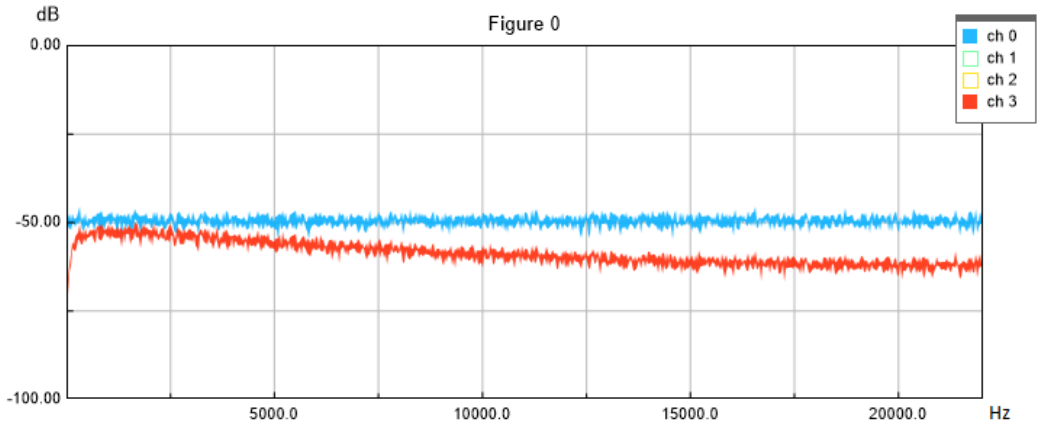


Figura 10: grafico nel dominio della frequenza del rumore bianco, in NU-Tech. In blu l'input, in rosso l'output

Anche se la distorsione introdotta è difficile da percepire sia ascoltando il segnale ottenuto che analizzando il grafico nel tempo, nello spettro è invece ben visibile l'andamento in frequenza del modello usato. In particolare si avrà un netto taglio di frequenze basse intorno ai 10-20 Hz dovuto all'effetto del filtro passa alto, mentre la riduzione dell'intensità delle frequenze alte a partire da 5000 Hz in poi generata dal filtro passa basso sarà meno accentuata ma comunque presente.

5.2 Sinusoide

Usando in ingresso un tono puro, ovvero un suono composto da una singola frequenza, si è in grado di osservare come la distorsione dell'onda sinusoidale aggiunga componenti armoniche superiori in uscita; inoltre nel dominio del tempo si verifica il clipping generato dal plugin.

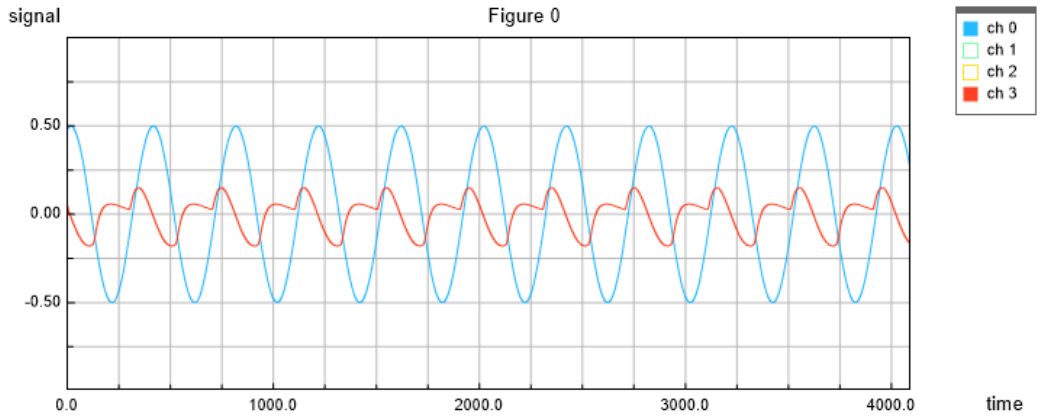


Figura 11: grafico nel dominio del tempo di una sinusoide a 110Hz. In blu l'input, in rosso l'output

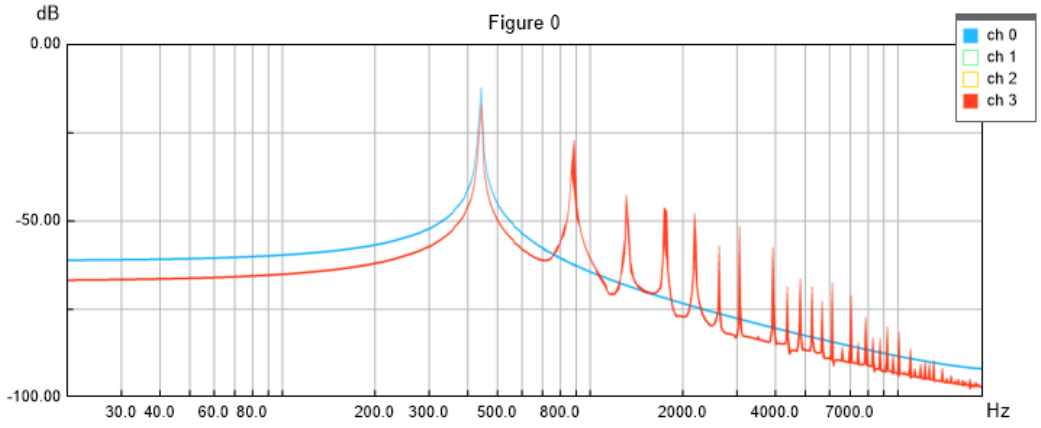


Figura 12: grafico nel dominio della frequenza in scala logaritmica di una sinusoide a 440Hz. In blu l'input, in rosso l'output

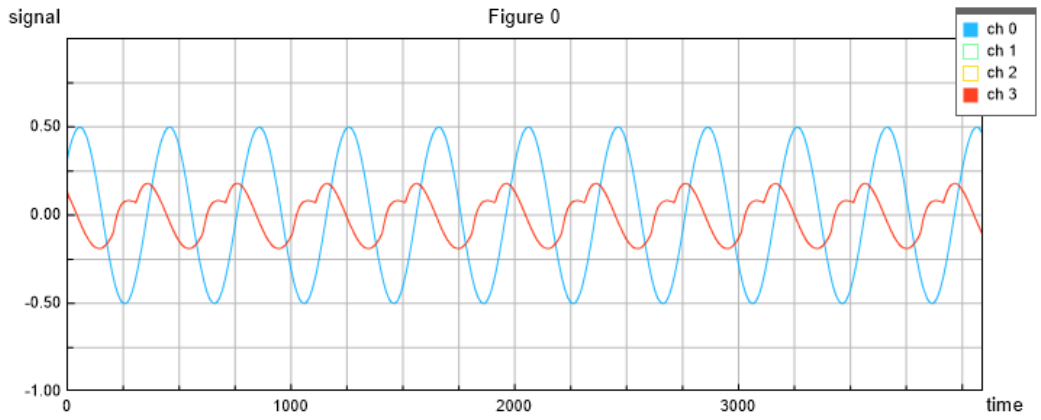


Figura 13: grafico nel dominio del tempo di una sinusoide a 110Hz con i parametri settati come segue: gain 10, dist 25, Q -7. In blu l'input, in rosso l'output

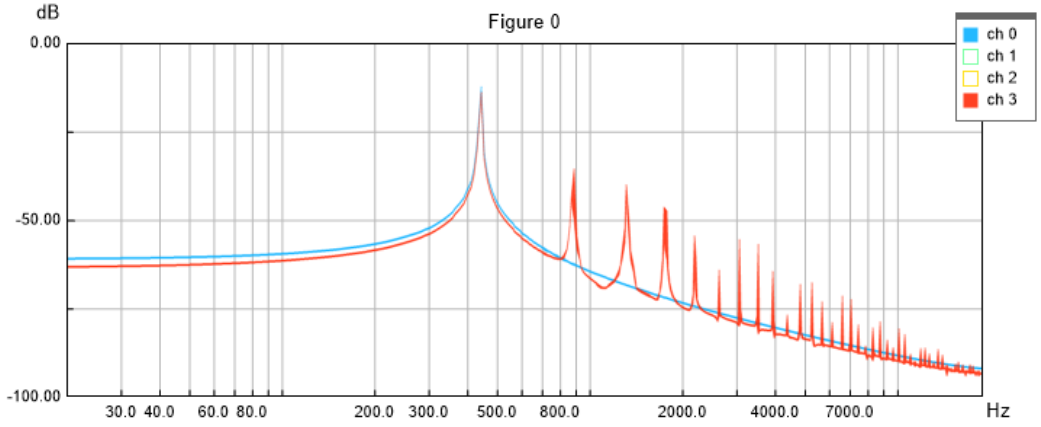


Figura 14: grafico nel dominio della frequenza in scala logaritmica di una sinusoide a 440Hz con i parametri settati come segue: gain 10, dist 25, Q -7. In blu l'input, in rosso l'output

I risultati teorizzati e attesi sono verificati e ben visibili sia nel dominio del tempo che nel dominio della frequenza, in particolare si osserva l'asimmetria del taglio delle ampiezze e la presenza di armoniche superiori di ordine pari e dispari. Si confrontano poi gli effetti ottenuti modificando i valori dei parametri: nel secondo settaggio il punto di lavoro è stato spostato verso sinistra rendendo la funzione di trasferimento più lineare e pertanto, anche a fronte di un aumento di guadagno e distorsione, il clipping è ridotto rispetto al primo esempio. Analogamente nel dominio delle frequenze l'intensità delle armoniche superiori è ridotta, soprattutto sulla seconda armonica.

5.3 Tracce Audio

Infine si utilizza il plugin per la funzione per cui è stato pensato, ovvero la distorsione un suono pulito di chitarra elettrica: si osservano pertanto gli effetti nel dominio del tempo e della frequenza per alcune tracce audio.

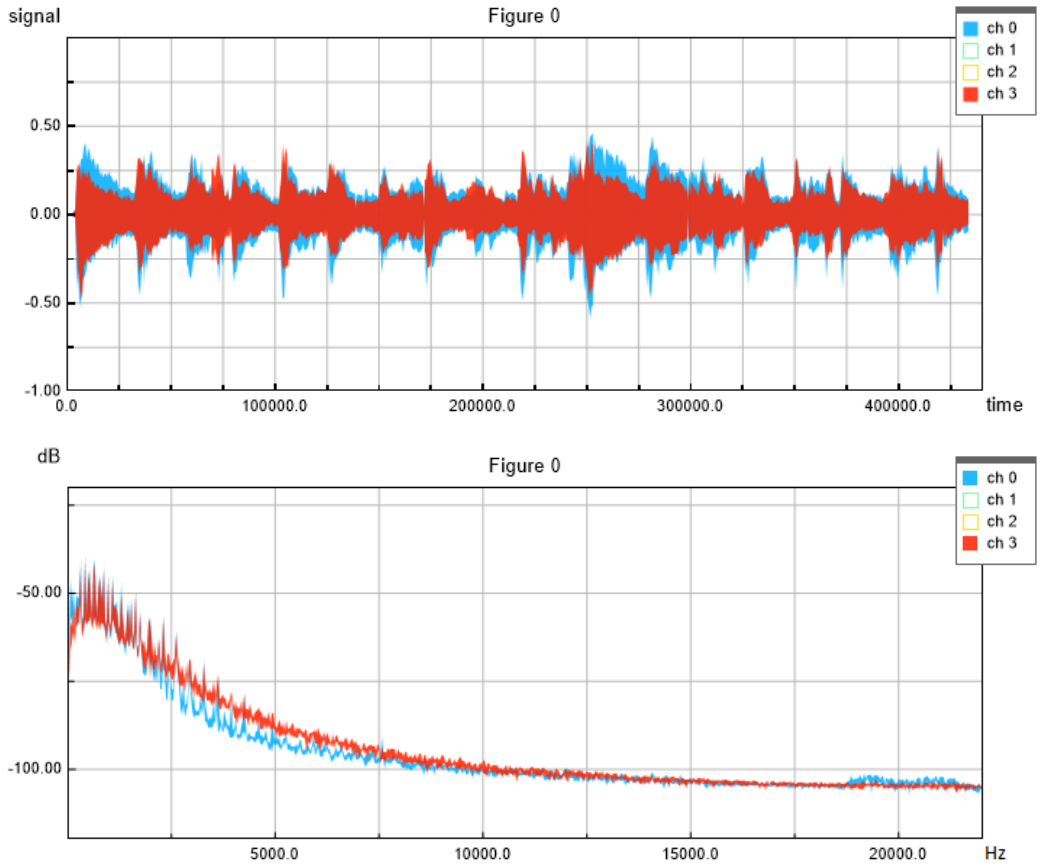
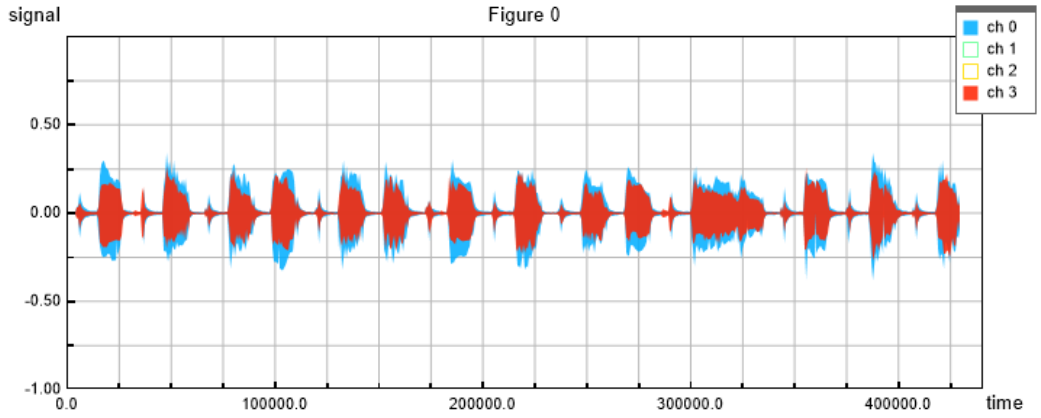


Figura 15-16: grafici nel dominio del tempo e della frequenza di una traccia audio (accordi di chitarra). In blu l'input, in rosso l'output

Il risultato di maggiore interesse in questo caso riguarda lo spettro, in cui intorno ai 2500 – 5000 Hz si verifica un aumento dell'intensità delle frequenze dovuto proprio alle armoniche superiori che caratterizzano la distorsione valvolare.



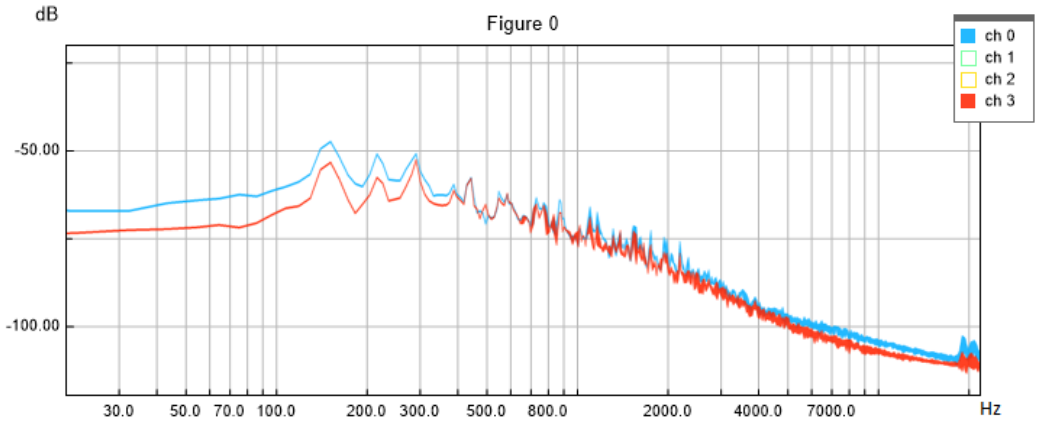


Figura 17-18: grafici nel dominio del tempo e della frequenza in scala logaritmica di una traccia audio (riff di chitarra). In blu l'input, in rosso l'output

Nella seconda traccia viene suonato un minor numero di note all'unisono rispetto alla precedente: infatti nel rock si è soliti ridurre gli accordi alla nota domina++nte, terza e quinta. Quello che ne consegue è uno spettro meno ricco di armoniche superiori come si osserva dal dominio della frequenza, anche se all'ascolto il suono risulta particolarmente efficace. Un risultato inatteso però riguarda il volume, in quanto il livello del segnale di uscita è minore del livello d'ingresso: una possibile soluzione potrebbe essere l'aggiunta di un ulteriore gain o di un compressore in coda al plugin, così da aumentare e livellare l'output.

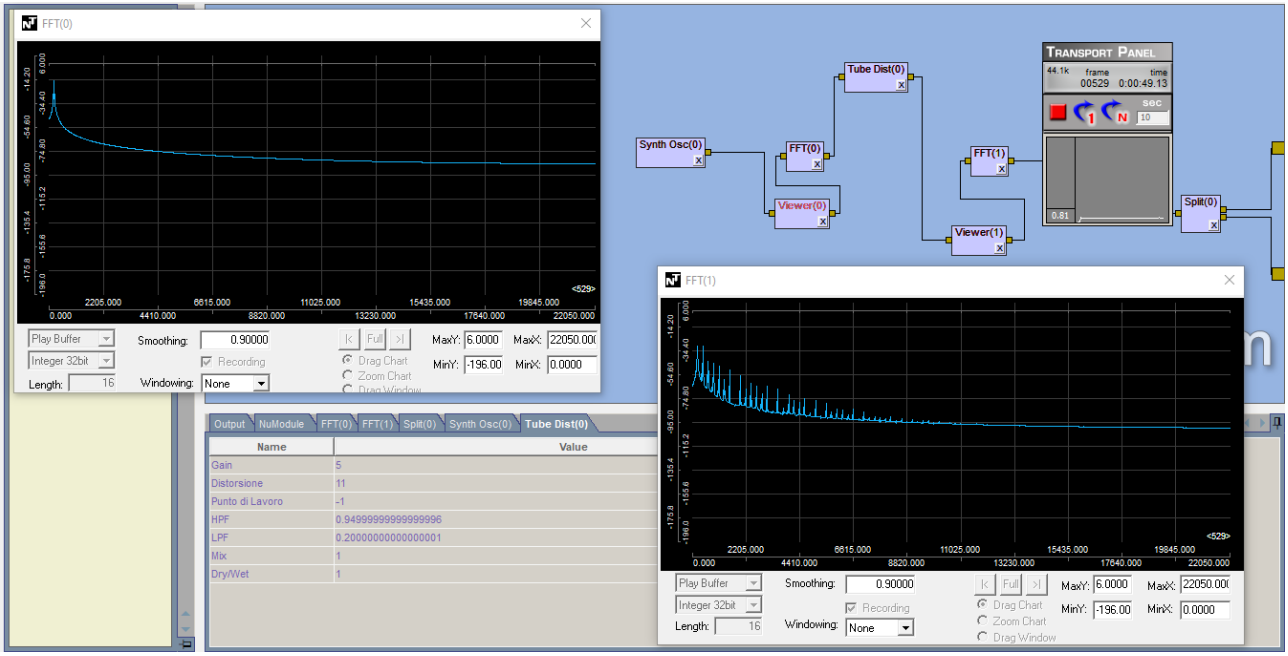


Figura 19: il plugin Tube Dist in funzione su NU-Tech

Conclusioni

Il plugin realizzato è quindi ben funzionante, in quanto come provato dai test realizza una distorsione asimmetrica tipica delle valvole a triodo così come previsto dal modello matematico proposto.

Risulta poi essere utilizzabile in tempo reale, poiché responsivo alle modifiche dei valori dei suoi parametri tramite il pannello dell'RT Watch e di latenza contenuta. Elaborando attraverso esso una registrazione di una chitarra elettrica "clean", ovvero priva di alterazioni, all'ascolto soggettivo il risultato ottenuto è gradevole e si avvicina a quello di modellazioni più complesse della distorsione valvolare. Pur essendo più semplice in confronto ad altri software disponibili sul mercato, il plugin offre ugualmente un'emulazione efficace di un amplificatore valvolare, garantendo inoltre un uso minore di CPU e memoria.
Per sviluppi futuri si può ipotizzare in primo luogo l'implementazione di un'interfaccia grafica ad hoc e una ricompilazione che ne consenta l'utilizzo su tutte le piattaforme di Digital Signal Processing. Inoltre, ricordando che il modello proposto emula il comportamento della sola valvola a triodo, si potrebbe partire da quanto realizzato per implementare un algoritmo tale da emulare l'intero circuito e i vari stadi dell'amplificatore valvolare.

Bibliografia

[1] Leaff Engineering, NU-Tech, 2009-2016: https://www.leaff.it/content.php?id=31

[2] Zolder U., *DAFX: Digital Audio Effects*, John Wiley & Sons Ltd., 2002

[3] Polyphonic*, A Brief History of Electric Guitar Distorsion: https://www.youtube.com/watch?v=iYU90XajYmU*

[4] Navarro Sosa E., Encyclopædia Britannica, Encyclopædia Britannica Inc., 2013: https://www.britannica.com/technology/electron-tube

[5] Zolder U., *DAFX: Digital Audio Effects*, John Wiley & Sons Ltd., 2002, cap. 5.3.2 *Valve Simulation,* pp. 111-112

[6] Araya, T., and A. Suyama. (1996). *Sound Effector Capable of Imparting Plural Sound Effects Like Distortion and Other Effects.*, U.S. Patent No. 5,570,424. Filed Nov. 24, 1993, issued Jun. 4, 1996.

[7] Doidic, M., et al. (1998). *Tube Modeling Programmable Digital Guitar Amplification System.* U.S. Patent No. 5,789,689. Filed Jan. 17, 1997, issued Aug. 4, 1998.

[8] Zolder U., *DAFX: Digital Audio Effects*, John Wiley & Sons Ltd., 2002, cap. 5.3.2 *Valve Simulation,* pp. 118-120

[9] Boyd, S. P. 1985. *Volterra Series: Engineering Fundamentals.* Doctoral Thesis, University of California at Berkeley.

[10] Zolder U., *DAFX: Digital Audio Effects*, John Wiley & Sons Ltd., 2002, cap. 5.3.1 *Basics of Nonlinear Modeling*, pp. 106-107

[11] EECS Department of the University of California at Bekerley, SPICE: http://bwrcs.eecs.berkeley.edu/Classes/IcBook/SPICE/

[12] Karjalainen, M., & Pakarinen, J. (2006). Wave digital simulation of a vacuum-tube amplifier. In *2006 IEEE International Conference on Acoustics Speech and Signal Processing Proceedings* (Vol. 5, pp. V-V). IEEE.

[13] Schmitz, T., & Embrechts, J. J. (2018). Real time emulation of parametric guitar tube amplifier with long short term memory neural network. *arXiv preprint arXiv:1804.07145*.

[14] Shattered Glass Audio, Ace, 2014: https://www.shatteredglassaudio.com/product/103

[15] Klanghelm, *SDRR*, 2011-2020: klanghelm.com/contents/products/SDRR.html

[16] Zolder U., *DAFX: Digital Audio Effects*, John Wiley & Sons Ltd., 2002, cap. 5.3.3 *Overdrive, Distortion and Fuzz,* pp. 121-123

[17] Intel Corporation, *Intel Integrated Performance Primitives Developer Reference, Volume 1: Signal Processing,* pg. 233

[18] Zolder U., *DAFX: Digital Audio Effects*, John Wiley & Sons Ltd., 2002, cap. 5.3.3 *Overdrive, Distortion and Fuzz,* pp. 123-124

[19] The MathWorks Inc., MATLAB, 1994-2020: https://it.mathworks.com/products/matlab.html

[20] Intel Corporation, Intel Integrated Performance Primitives: https://software.intel.com/content/www/us/en/develop/tools/integrated-performance-primitives.html

[21] The MathWorks Inc., voce *buffer* in MATLAB Documentation: https://it.mathworks.com/help/signal/ref/buffer.html

[22] The MathWorks Inc., voce *reshape* in MATLAB Documentation: https://it.mathworks.com/help/matlab/ref/reshape.html

[23] Leaff Engineering, *NUTS Software Development Kit 2.0,* Rev 1.2, 2014, pg. 15

[24] Intel Corporation*, Intel Integrated Performance Primitives Developer Reference, Volume 1: Signal Processing,* pp. 93-94

[25] Intel Corporation, *Intel Integrated Performance Primitives Developer Reference, Volume 1: Signal Processing,* pp. 237-238

[26] Intel Corporation, *Intel Integrated Performance Primitives Developer Reference, Volume 1: Signal Processing,* pp. 234-235

[27] The MathWorks Inc., esempio *Overlap-Add/Save* in MATLAB Documentation: https://it.mathworks.com/help/dsp/ug/overlap-add-save.html

[28] Intel Corporation, *Intel Integrated Performance Primitives Developer Reference, Volume 1: Signal Processing,* pp. 239-240