UNIVERSITÀ
POLITECNICA
DELLE MARCHE

Facoltà di Ingegneria
Corso di Laurea in Ingegneria Meccanica

Studio e Sviluppo di un Algoritmo per la Compensazione di un Array Circolare di Altoparlanti

Study and Development of an Algorithm for the Compensation of a Circular Array Loudspeakers

Candidato:
Michele Marmorè

Relatore:
Prof.ssa Stefania Cecchi

Correlatore:
Dott.ssa Valeria Bruschi
Sig. Ulises Wilfredo Ruiz Rodes

Anno Accademico 2023-2024

UNIVERSITÀ
POLITECNICA
DELLE MARCHE

Facoltà di Ingegneria
Corso di Laurea in Ingegneria Meccanica

Studio e Sviluppo di un Algoritmo per la Compensazione di un Array Circolare di Altoparlanti

Study and Development of an Algorithm for the Compensation of a Circular Array Loudspeakers

Candidato:
Michele Marmorè

Relatore:
Prof.ssa Stefania Cecchi

Correlatore:
Dott.ssa Valeria Bruschi
Sig. Ulises Wilfredo Ruiz Rodes

Anno Accademico 2023-2024

Università Politecnica delle Marche
Facoltà di Ingegneria
Corso di Laurea in Ingegneria Meccanica
Via Brecce Bianche – 60131 Ancona (AN), Italy

Indice

Elenco delle figure

Capitolo 1

Introduzione

I diffusori acustici sono dispositivi realizzati per la riproduzione di suoni. Il suono è definito come un'onda di pressione composta da una componente principale (frequenza fondamentale), e componenti multiple della fondamentale (frequenze armoniche). Un tono puro è un suono il cui spettro è caratterizzato da una sola frequenza. Se lo spettro contiene componenti armoniche il suono è distorto. La distorsione armonica è un fenomeno che comporta la modifica delle caratteristiche sonore, ad esempio di un diffusore o di uno strumento, poiché ne arricchisce le componenti spettrali. Questo fenomeno non è sempre negativo. Infatti, la combinazione di una fondamentale e alcune armoniche genera la timbrica che è una delle caratteristiche principali degli strumenti musicali. Una chitarra ad esempio non emette toni puri, la corda eccitata produce una frequenza principale ed alcune armoniche che arricchiscono lo spettro. Raramente in editing musicale vengono utilizzati suoni semplici come ad esempio le sinusoidi. In questo caso, quindi, la distorsione è ben voluta. Per quanto riguarda la riproduzione sonora, un diffusore deve essere in grado di riprodurre un suono senza aggiungere componenti armoniche, che andrebbero a modificare lo spettro del segnale originale. Gli altoparlanti elettrodinamici sono i trasduttori elettroacustici più comuni. Questi permettono di trasformare un segnale elettrico proveniente da un amplificatore in onde di pressione acustica. L'altoparlante introduce distorsioni armoniche dovute a componenti reali che lo compongono come motore e sospensioni. A questo fenomeno si aggiunge la distorsione dovuta alla risonanza del volume in cui è inserito. Idealmente il diffusore deve essere costruito in modo da non permettere oscillazioni e vibrazioni delle pareti rigide. Nel caso reale, per motivi fisici, questo fenomeno si verifica aggiungendo una componente di distorsione al sistema. Se si considerano piccoli segnali, il diffusore può essere approssimato idealmente come lineare poiché gli altoparlanti non lavorano a regimi tali da manifestare o introdurre

distorsione e generano onde meccaniche troppo deboli per produrre vibrazioni delle pareti del diffusore. Se il sistema viene alimentato con potenze più elevate, le distorsioni diventano non trascurabili. Generalmente i diffusori non sono utilizzati nelle condizioni per cui è possibile trascurarne la distorsione introdotta, quindi è necessario descriverne il comportamento con relazioni non lineari. I sistemi di questo tipo hanno caratteristiche complesse ed a causa di ciò non possono essere descritti da equazioni lineari. Quando si parla di equazioni è necessario introdurre il concetto di ordine: l'ordine di una equazione algebrica, rappresenta il grado dell'esponente più alto della variabile incognita; nel caso di equazioni differenziali, l'ordine rappresenta il grado della derivata di ordine più alto presente. Un sistema lineare non ha termini incogniti di ordine superiore al primo. Un sistema non lineare invece può essere descritto con equazioni algebriche e differenziali di ordine superiore al primo, ognuno dei quali identifica una risposta armonica. Idealmente è possibile descrivere infinite armoniche, tuttavia, per i sistemi reali descrivere un ordine infinito di non linearità equivale ad un costo computazionale di pari grandezza. Quindi per contenere la complessità numerica, l'ordine di rappresentazione va limitato. Un ulteriore concetto da introdurre è quello della memoria. Un sistema, sia lineare che non lineare, può contenere memoria, ciò significa che l'uscita generata non è dovuta solo all'ingresso corrente ma è influenzata dagli ingressi precedenti. Quando il sistema è non lineare, questa dipendenza dagli stati precedenti assume una complessità aggiuntiva, poiché l'interazione tra gli input passati e presenti avviene in modo non lineare. Questo lavoro di tesi è incentrato sulla caratterizzazione di un diffusore composto da un array circolare di altoparlanti e sulla compensazione della parte lineare e non lineare dello stesso.

Capitolo 2

Stato dell'arte

2.1 Tecniche di modellazione dei sistemi sonori non lineari

Per analizzare le caratteristiche di un diffusore è necessario misurarne la risposta. Questa è molto importante al fine di valutare le distorsioni introdotte dal sistema. Successivamente si sceglie un modello in grado di descriverne nel miglior modo possibile il comportamento. Il modello è l'insieme delle relazioni matematiche che descrivono il funzionamento del sistema incognito che in questo caso è il diffusore. Questo deve essere più fedele possibile al sistema reale, poiché conoscerne le caratteristiche permette di prevederne, per ogni eccitazione in ingresso, lo stato futuro. Inoltre, la caratterizzazione del sistema consente di lavorare in "offline" senza la necessità di disporre del sistema reale. Nella teoria dei sistemi si possono distinguere tre approcci:

- white-box: la costruzione del modello viene fatta sulla base delle equazioni dello spazio di stato del sistema. L'approccio "white-box" prevede la conoscenza di fenomeni interni al sistema per la modellazione. Questa caratteristica permette di ottenere un ottimo modello a discapito di una notevole complessità numerica. Inoltre per sistemi complessi risulta molto complicato dedurre le equazioni che descrivono la relazione ingresso-uscita. Un esempio di modello white-box è quello che si ricava partendo dalla conoscenza del circuito elettrico che descrive il dispositivo da caratterizzare;

- black-box: in questo approccio, il sistema incognito viene descritto dal suo comportamento esterno. Il modello viene costruito misurando l'uscita generata da un ingresso noto senza avere alcuna conoscenza interna del sistema. Questa

tecnica consente un approccio più semplice, tuttavia, l'interpretazione del modello risulta più complicata;

- grey-box: questo approccio si trova a metà strada tra quello black-box e quello white-box. Questa tecnica viene utilizzata quando non si conoscono tutte le leggi che regolano l'uscita, quindi viene creato un modello di riferimento che descrive una parte del comportamento del sistema da identificare (white-box) e integrato la caratterizzazione con la relazione ingresso-uscita (black-box).

Un sistema fisico viene rappresentato mediante un modello matematico sotto forma di un sistema dinamico. Questo è classificato lineare solo se le equazioni differenziali che lo descrivono sono lineari. In caso contrario, il sistema è non lineare. Inoltre i sistemi possono essere divisi in tempo-variante e tempo-invariante. Questa caratteristica descrive il comportamento dei sistemi nel tempo. Un ingresso noto in un sistema tempo-invariante produce la stessa uscita in qualunque momento, in caso contrario il sistema è tempo-variante. In genere per la caratterizzazione dei sistemi LTI (lineari-tempo-invariante) si preferisce un approccio più semplice come quello black-box in modo da avere una buona identificazione mantenendo un costo computazionale basso. In questo caso basta misurare il comportamento del sistema misurando la risposta all'impulso unitario per conoscere tutte le caratteristiche della black-box che identifica il sistema lineare. Al contrario, per sistemi non lineari diventa più complicato. Per contenere la complessità numerica, l'identificazione della parte non lineare va limitata ad un ordine specifico che nel caso in questione è tre. Inoltre, si devono separare i blocchi di non linearità statica dalla risposta lineare dinamica. Così facendo si ottiene un modello meno generalizzato ma comunque fedele al sistema reale originale. In questo capitolo verranno analizzati i metodi di modellazione per i sistemi non lineari presenti in letteratura. Alcuni esempi sono: lo sviluppo in serie di Volterra [1], il modello di Hammerstein [4], il modello di Wiener [3] e il modello Wiener-Hammerstein [3]. Successivamente saranno descritti nella sezione 2.3 i metodi per la compensazione dei modelli sopra citati.

2.1.1 Serie di Volterra

La teoria della serie di Volterra fu sviluppata dal matematico Vito Volterra nel 1900, il quale introdusse un'estensione dell'espansione delle serie di Taylor ai funzionali. Un funzionale prende una funzione come input e restituisce un numero reale o complesso come output. Un esempio è la funzione che associa un valore di output reale a un continuum di valori di input passati [5]. L'uso delle serie di Volterra fu introdotto nella teoria dei sistemi non lineari da Norbert Wiener negli anni '40 [6]. L'approccio fornisce una spiegazione coerente del comportamento del sistema non lineare e fornisce una caratterizzazione di un sistema sia nel dominio del tempo che nel dominio della frequenza. Inoltre, utilizzando un'estensione multidimensionale della trasformata di Laplace e un set di regole di combinazione, prevede la risposta non lineare globale di sistemi costituiti da blocchi non lineari interconnessi con addizione, moltiplicazione, cascata e feedback. Le equazioni differenziali possono anche essere utilizzate come punto di partenza con l'approccio Volterra/Wiener e questo è stato precedentemente utilizzato per modellare il comportamento non lineare degli altoparlanti [7] [8]. In generale, il comportamento dei sistemi non lineari è ben descritto da tale approccio per le non linearità regolari e moderate [36]. Distorsioni di questo tipo sono prodotte da non linearità che non introducono discontinuità e sono possono essere analizzate in modo più semplice, ad esempio con funzioni polinomiali o con l'utilizzo della serie di Volterra. Tuttavia, quest'ultima non descrive bene le distorsioni irregolari come il Rub & Buzz dell'altoparlante e le non linearità estreme come il clipping dell'amplificatore [36]. L'approccio introdotto risulta valido per distorsioni regolari inferiori al 10% [36]. La teoria di Volterra-Wiener è un'estensione della teoria dei sistemi lineari. Per questi, l'uscita in t è una somma ponderata dei valori di ingresso. Di seguito è riportata l'equazione che descrive un sistema lineare:

$$y(t) = \int_{-\infty}^{\infty} h(\tau)x(t-\tau)\,d\tau. \tag{2.1}$$

In questo caso la $h(t)$ rappresenta la risposta di Volterra del primo ordine. L'idea di Wiener è quella di descrivere l'uscita non lineare del sistema come somma di funzioni integrali:

$$
\begin{aligned}
y(t) &= \sum_{n=1}^{\infty} y_n(t) \\
&= \int_{-\infty}^{\infty} h_1(\tau)x(t-\tau)\,d\tau \\
&\quad + \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} h_2(\tau_1,\tau_2)x(t-\tau_1)x(t-\tau_2)\,d\tau_1\,d\tau_2 \\
&\quad + \dots \\
&\quad + \int_{-\infty}^{\infty}\dots\int_{-\infty}^{\infty} h_n(\tau_1,\tau_2,\dots,\tau_n)x(t-\tau_1)x(t-\tau_2)\dots x(t-\tau_n)\,d\tau_1\,d\tau_2\dots d\tau_n.
\end{aligned}
\tag{2.2}
$$

Nello specifico nella formula (2.2) è riportata la serie di Volterra nella quale il primo termine rappresenta la risposta lineare (coincide infatti con la formula (2.1)), e tutti gli altri termini indicano le n armoniche. Le componenti $h(n)$ sono detti kernel di Volterra e indicano le risposte impulsive dell'ordine non lineare $n > 1$ corrispondente. Per mantenere l'onere computazionale accettabile la serie di Volterra deve essere troncata all'ordine di caratterizzazione che si vuole eseguire. Ad esempio per costruire un modello non lineare del terzo ordine si avrà la somma di tre equazioni.

2.1.2 Modello di Wiener

Il modello di Wiener consente di rappresentare un sistema dinamico a blocchi come cascata di un blocco lineare dinamico seguito da un blocco non lineare statico [9]. In Figura 2.1 è illustrato lo schema a blocchi della struttura di Wiener, dove la $G(s)$ rappresenta la funzione di trasferimento nel dominio della trasformata di Laplace relativa al blocco lineare e $f(x)$ rappresenta la parte non lineare statica del modello.

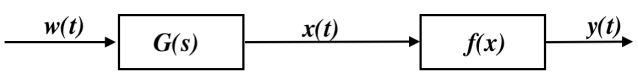


Figura 2.1: Modello di Wiener a blocchi.

Il segnale intermedio $x(t)$, dato dal segnale di ingresso $w(t)$ filtrato con la parte lineare $G(s)$ è definito come segue:

$$x(t) = G(s) \cdot w(t) \tag{2.3}$$

$$y(t) = f(x(t)) \tag{2.4}$$

Come mostrato nella formula (2.4) l'uscita $y(t)$ è data dall'applicazione della funzione non lineare al segnale intermedio. Il segnale $x(n)$ che collega i due blocchi non è direttamente misurabile, e la parte non lineare, essendo in cascata, rende più complessa l'identificazione tramite la misura ingresso uscita. La struttura del modello di Wiener implica che la parte con memoria sia tutta rappresentata nel blocco lineare e quella non lineare sia senza memoria. La procedura di identificazione è composta da due fasi. La prima è quello del calcolo della parte lineare, la seconda è l'applicazione di una non linearità che consenta una buona fedeltà del modello al sistema reale di riferimento. In letteratura esistono diversi metodi di caratterizzazione con Wiener. Uno dei più efficaci è il metodo dei quadrati [37].

2.1.3 Modello di Hammerstein

Un modello non lineare a blocchi si dice di Hammerstein quando è costituito da un blocco non lineare statico seguito da un blocco lineare dinamico. Lo schema di principio è illustrato in Figura 2.2.

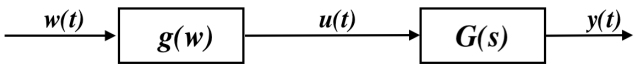


Figura 2.2: Modello di Hammerstein a blocchi.

Rispetto al modello di Wiener, la non linearità $g(w)$ è inserita prima del blocco lineare, la relazione ingresso-uscita del modello è il seguente:

$$y(t) = G(s) \cdot u(t), \tag{2.5}$$

dove *u(t)* è definito come segue:

$$u(t) = g(w(t). \tag{2.6}$$

Come per Wiener, anche in questo caso i segnali misurabili sono $w(t)$ e $y(t)$, mentre $u(n)$ non è direttamente misurabile. Quindi generalmente non è possibile distinguere i due blocchi separatamente. Per fare ciò si possono utilizzare tecniche diverse. In letteratura sono presenti alcuni metodi di caratterizzazione che utilizzano il modello di Hammerstein. Un esempio è l'approccio indicato in [10] mostrato in Figura 2.3.

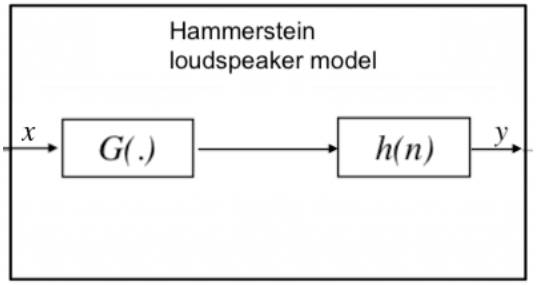


Figura 2.3: Modello di Hammerstein dell'approccio[10].

In questo caso la parte non lineare statica viene modellata con un polinomio g di grado N ed il blocco lineare dinamico è caratterizzato con un filtro FIR (Finite Impulsive Response) indicato con $h[n]$. In seguito è indicata l'espressione matematica del polinomio che modella il blocco non lineare come segue:

$$g(x) = \sum_{n=1}^{N} c_n \psi_n(x) = \boldsymbol{\psi}^T(x)\mathbf{c}. \tag{2.7}$$

In particolare x rappresenta l'ingresso al sistema di Hammerstein, $\boldsymbol{\psi}^T(x)$ è il vettore delle funzioni di base di x e il termine $\boldsymbol{c}$ è il vettore dei coefficienti che moltiplicano le componenti non lineari. Oltre al modello di Hammerstein in forma polinomiale è bene introdurre quello illustrato in Figura 2.4 che prende il nome di "Modello di Hammerstein generalizzato".

Il modello di Hammerstein generalizzato introdotto da Antonin Novak in [34] è stato utilizzato con successo negli ultimi anni in molte applicazioni fisiche per descrivere il comportamento di un sistema non lineare in fase di test. Il vantaggio principale di tale modello non lineare è la sua capacità di modellare in modo efficiente sistemi

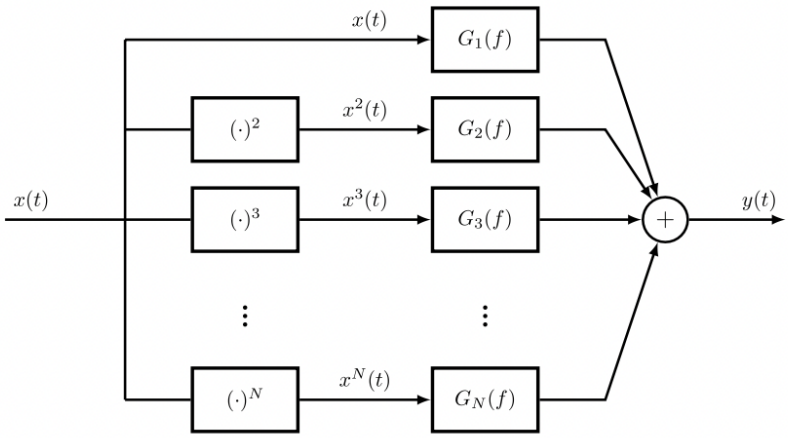


Figura 2.4: Modello di Hammerstein generalizzato.

non lineari mantenendo bassi i costi computazionali. D'altro canto, questo modello non può prevedere comportamenti non lineari complicati come quello isteretico. A tal proposito, Antonin novak propone un'estensione del modello di Hammerstein generalizzato a un modello con input non lineari non polinomiali che consente di modellare sistemi non lineari più complicati con un buon accordo tra il modello e il sistema non lineare isteretico in fase di test [11].

2.1.4 Modello Wiener-Hammerstein

Un sistema dinamico a blocchi si dice di Wiener-Hammerstein se è composto dalla cascata di un blocco lineare dinamico, uno non lineare statico ed infine uno lineare dinamico come mostrato in Figura 2.5.

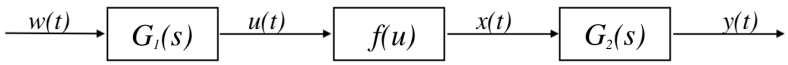


Figura 2.5: Modello di Wiener-Hammerstein.

In questo caso i blocchi $G_1(s)$ e $G_2(s)$ rappresentano la risposta lineare dinamica del modello mentre il blocco $f(u)$ modella la parte non lineare statica. Come nei casi precedenti anche qui gli unici parametri direttamente misurabili sono l'ingresso e l'uscita, rispettivamente *w(t)* e *y(t)*. Rispetto all'approccio Wiener e Hammerstein in questo caso i segnali incogniti sono due. Questa caratteristica rende il modello

Wiener-Hammerstein più complesso a livello di identificazione [35]. Alcuni metodi di identificazione per modelli Wiener-Hammerstein sono indicati in [12] [13].

2.2 Procedura di identificazione dei modelli non lineari

Nella sezione precedente sono stati indicati i modelli di caratterizzazione dei sistemi non lineari. La procedura di identificazione permette di passare dal sistema fisico al modello desiderato. Per quando riguarda l'ambito audio, in letteratura sono presenti diversi metodi per l'identificazione dei modelli sopra citati.

2.2.1 Identificazione basata sulla cross-correlazione

Per la costruzione del modello di Volterra è necessario riuscire ad identificare i nuclei (o kernel) di Volterra. A tale scopo si possono utilizzare delle tecniche di minimizzazione dell'errore quadratico medio (MSE), calcolato comparando l'uscita del sistema reale con quella del modello in fase di identificazione. Il calcolo può essere effettuato con la tecnica della cross-correlazione [14]. Per fare ciò, in accordo con la teoria di Lee-Schetzen(L-S) [15], viene fornito un ingresso di tipo rumore bianco gaussiano al sistema che si sta identificando e si cerca di minimizzare l'errore quadratico medio. Tuttavia per poter applicare L-S è necessario che l'uscita del sistema sia sotto forma di somma di termini ortogonali. La serie di Volterra non soddisfa questa condizione, quindi l'identificazione viene fatta con i kernel di Wiener che verranno poi trasformati nella forma dei kernel di Volterra. Nell'articolo [16] è indicato un metodo di identificazione che utilizza questo approccio di trasformazione.

2.2.2 Identificazione basata sul segnale "Swept-Sine"

Un segnale swept-sine, o chirp, è un segnale ampiamente utilizzato per caratterizzare una risposta in frequenza di un sistema lineare. Una tecnica per l'analisi di sistemi non lineari basata su un segnale swept-sine esponenziale è stata presentata da Farina nel 2000 [17] [18]. Questa tecnica consente una separazione delle risposte all'impulso per ogni ordine di distorsione armonica. Dal 2000, il metodo è stato utilizzato in molte applicazioni nei campi dell'audio e dell'acustica, ad esempio per le misurazioni della risposta all'impulso della stanza [19] [20], per la misurazione delle

funzioni di trasferimento correlate alla testa e delle risposte all'impulso correlate alla testa [21] [22], nell'acustica non lineare [23] [24] e in molte altre applicazioni in acustica. La necessità di sincronizzare il segnale sinusoidale esponenziale per scopi di identificazione del sistema non lineare è stata presentata per la prima volta in [25] e il background matematico per l'identificazione di sistemi non lineari, utilizzando il segnale sinusoidale esponenziale è stato quindi fornito in [26]. Antonin Novak ha perfezionato un metodo di identificazione che utilizza lo Swept-Sine. Il metodo si basa sulla convoluzione non lineare [27]. In ingresso al sistema viene conferito un segnale (chirp) definito in (2.8) composto da una frequenza istantanea esponenziale e consente la caratterizzazione di un sistema non lineare in termini di distorsione armonica a diversi ordini.

$$x(t) = \sin\left\{2\pi f_1 L\left[\exp\left(\frac{t}{L}\right) - 1\right]\right\} \tag{2.8}$$

dove L è definito come segue:

$$L = \frac{1}{f_1} \cdot \text{round}\left[\frac{\hat{T} f_1}{\ln\left(\frac{f_2}{f_1}\right)}\right] \tag{2.9}$$

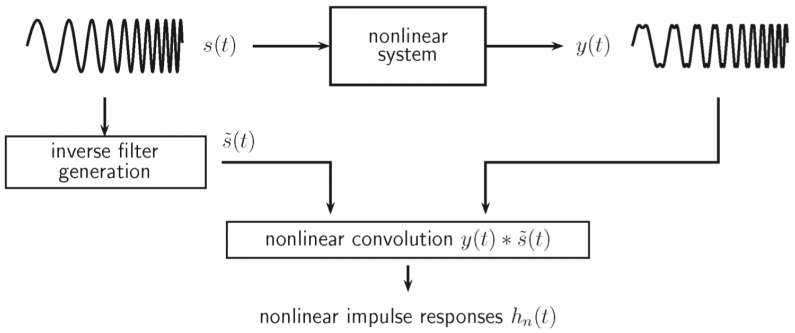


Figura 2.6: Diagramma a blocchi del processo di convoluzione non lineare.

La Figura 2.6 mostra il diagramma a blocchi del sistema di misura. In ingresso al sistema non lineare viene conferito uno swept-sine *s(t)* che passando per il NLS incognito produrrà un'uscita *y(t)* distorta. La *y(t)* viene quindi salvata per poter effettuare il processo di convoluzione non lineare. Il segnale $\tilde{s}(t)$ è derivato invertendo *s(t)* in modo che la convoluzione tra $\tilde{s}(t)$ e *s(t)* produca una delta di dirac $\delta(t)$.

Convoluendo quindi *y(t)* con $\tilde{s}(t)$ si ottiene la seguente espressione:

$$y(t) * \tilde{s}(t) = \sum_{m=1}^{\infty} h_m(t + \Delta t_m) \tag{2.10}$$

dove $h_m(t)$ rappresenta le risposte impulsive di ordine superiore e Δt_m rappresenta il ritardo temporale tra la fondamentale e la m-esima risposta impulsiva armonica. Poiché la risposta impulsiva non lineare consiste in un insieme di risposte impulsive di ordine superiore che sono spostate nel tempo, ogni risposta impulsiva parziale può essere separata l'una dall'altra, come illustrato nella Figura 2.7.

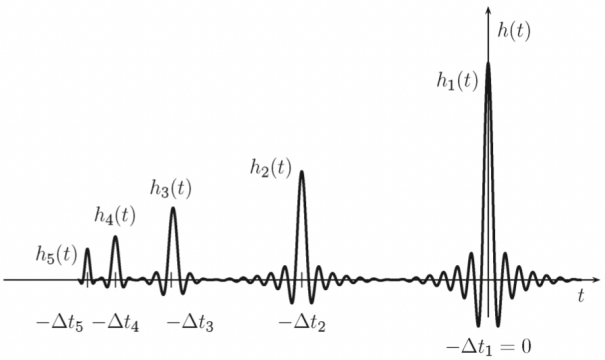


Figura 2.7: Risultato del processo di convoluzione non lineare.

Idealmente si possono calcolare infinite risposte impulsive relative alle armoniche successive alla fondamentale, tuttavia per motivi fisici e di complessità numerica è buona norma troncare l'ordine di non linearità identificato. Una volta identificate tutte le risposte impulsive inerenti alle armoniche successive è possibile costruire il modello di Hammerstein strutturato come illustrato nella Figura 2.4.

2.3 Tecniche di compensazione dei modelli

Dopo aver indicato i metodi che permettono di caratterizzare i sistemi non lineari e le tecniche di identificazione per raggiungere tali strutture, si pone ora il problema della compensazione. Essere in grado di diminuire la distorsione prodotta dagli altoparlanti e dal diffusore completo permette di raggiungere prestazioni migliori a parità di potenza per quanto riguarda il livello di pressione sonora ed inoltre consente

una riproduzione sonora migliorata che contiene una percentuale minore di artefatti dovuti alle caratteristiche non lineari dei diffusori stessi. In questa sezione verranno indicate alcune tecniche che consentono la correzione delle distorsioni dei sistemi non lineari. In tale ambito possiamo distinguere due macro-categorie: a ciclo aperto (open loop) ed a ciclo chiuso (closed loop). In entrambi i casi il metodo adottato consiste nella pre-distorsione del segnale che viene conferito in ingresso al diffusore. Questo avviene anteponendo al sistema da correggere un filtro con caratteristiche non lineari, in grado di produrre un segnale che, dato in ingresso al diffusore, generi un'uscita non distorta.

2.3.1 Modello ad anello chiuso

Il modello ad anello chiuso è illustrato in Figura 2.8. Come indicato in [28] le grandezze misurabili per un sistema feedback (closed loop) sono:

- Spostamento;
- Velocità;
- Corrente;
- Pressione.

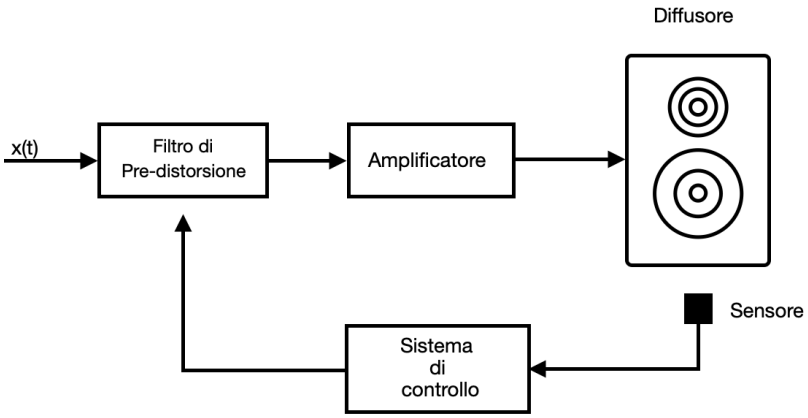


Figura 2.8: Schema a blocchi di un sistema closed loop.

I segnali misurati tramite opportuni sensori, corrispondenti alle grandezze indicate, servono ad implementare un filtro adattivo non lineare che ha il compito di pre-

distorcere il segnale con il modello non lineare inverso cosicché l'uscita prodotta risulti il più possibile fedele all'ingresso *x(t)*. Il blocco non lineare può essere costruito con una rete neurale ricorsiva, tuttavia le dimensioni e la complessità computazionale dei modelli di questo tipo rendono difficile l'implementazione in real-time. Perciò i modelli di tipo Hammerstein, Wiener e Volterra risultano più idonei. Un algoritmo adattivo feedback in tempo reale riesce a tenere conto dei cambiamenti del diffusore, ad esempio, in relazione all'usura o a variazioni fisiche della risposta dovute ad agenti esterni. Tuttavia il sistema diventa più complesso sia a livello hardware che software poiché l'integrazione di uno o più sensori implica una costruzione più complessa del diffusore stesso, e di fatto l'adattamento real-time richiede un costo computazionale più elevato. Inoltre, come il diffusore, anche i sensori saranno soggetti ad usura e cambiamenti dovuti ad agenti esterni (come polvere ed umidità) che, compromettendone il funzionamento, andrebbero ad alterare il risultato della compensazione che può risultare inefficace.

2.3.2 Modello ad anello aperto

Lo schema a blocchi di un sistema open loop è mostrato in Figura 2.9. Come si può notare, in questo caso non è presente nessun ramo di retroazione. Per poter effettuare una compensazione di un sistema ad anello aperto non è necessario l'impiego costante di sensori. Questa caratteristica rende il sistema più semplice da implementare a livello hardware e di costo computazionale [28].

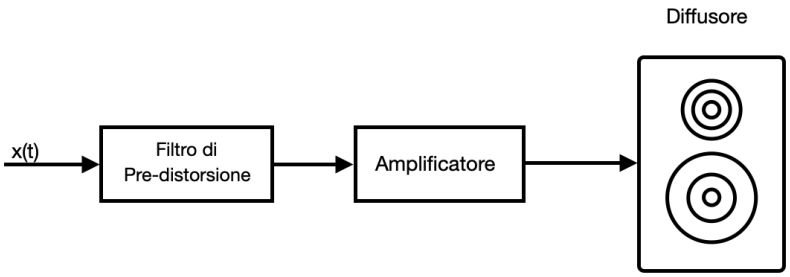


Figura 2.9: Schema a blocchi di un sistema open loop.

L'adattamento del filtro di pre-distorsione viene fatto offline e non in real time, quindi il ciclo di iterazione viene effettuato in riferimento al modello precedentemente

misurato. In questo caso, è quindi fondamentale disporre di una struttura che descriva fedelmente il comportamento non lineare del sistema reale. Rispetto al caso ad anello chiuso in real time, la compensazione, poiché effettuata una sola volta e poi applicata al diffusore, non terrà conto dell'usura dovuta alla degradazione degli altoparlanti (come perdita delle rigidità delle sospensioni) e della meccanica del diffusore (come perdita di rigidità delle pareti o di tensione delle viti di fissaggio dei trasduttori). In alcuni casi è possibile costruire un modello inverso apposito che consenta di compensare il comportamento non lineare del sistema reale. Come per il caso a ciclo chiuso si possono utilizzare, per il filtro di pre-distorsione, strutture come la serie di Volterra ed i modelli di Hammerstein e Wiener.

2.3.3 Struttura del filtro non lineare inverso

Il filtro inverso per la compensazione può avere diverse strutture. In questo paragrafo sono indicati i modelli di Volterra, Hammerstein e Wiener per la pre-distorsione.

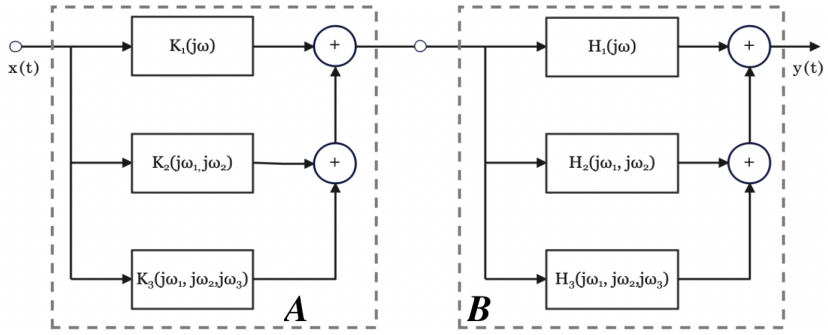


Figura 2.10: Schema a blocchi del pre-distorsore di Volterra

Nella Figura 2.10 è illustrato lo schema a blocchi del pre-distorsore con serie di Volterra. In particolare il riquadro ***A*** mostra il blocco che effettua la compensazione, mentre il riquadro ***B*** indica il modello sistema reale. In questo esempio è stato utilizzato un pre-distorsore di Volterra per la compensazione di un sistema con la stessa struttura, tuttavia non è una regola fissa. Il metodo descritto risulta molto complesso a livello computazionale, perciò si preferisce utilizzare modelli più semplici per la riduzione della distorsione, in particolar modo se l'algoritmo deve essere implementato su sistemi embedded come ad esempio DSP. I modelli di Wiener ed Hammerstein risultano meno onerosi dal punto computazionale. Per la compensazione dei sistemi

non lineari descritti con Wiener o Hammerstein, sono preferibili strutture dello stesso tipo. Possono essere considerati due sistemi di identificazione e compensazione: il sistema Wiener-Hammerstein, ed il sistema Hammerstein-Wiener.

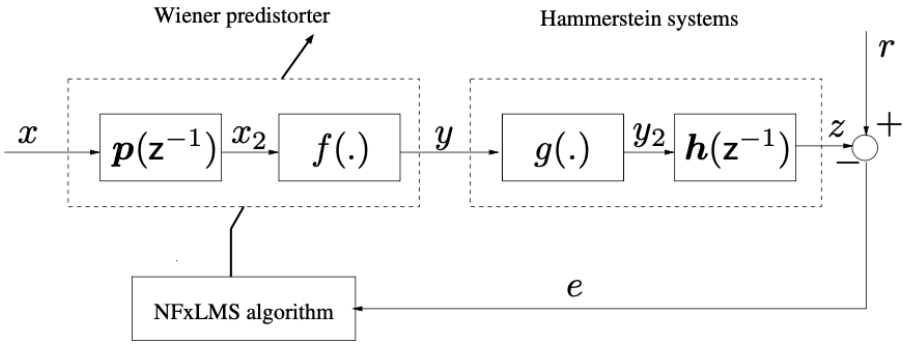


Figura 2.11: Schema a blocchi con pre-distorsore di Wiener

La Figura 2.11 mostra una struttura Wiener-Hammerstein dove il primo viene utilizzato come modello inverso del secondo[29]. L'utilizzo di un modello di questo tipo risulta più semplice. Generalmente i segnali accessibili sono solo l'ingresso x e l'uscita z, quindi un sistema come quello indicato risulta meno complesso poiché le relazioni che legano i segnali nascosti da quelli misurabili sono lineari. In questo lavoro di tesi verrà utilizzato il modello di Wiener come filtro non lineare inverso per compensare un array circolare di altoparlanti identificato dal modello di Hammerstein full-band [33].

Capitolo 3

Algoritmo implementato

L'obiettivo del lavoro di tesi è quello di compensare le distorsioni introdotte da un diffusore composto da woofer e tweeter configurato come un array di altoparlanti circolare. La caratterizzazione è fatta con il modello di Hammerstein con una struttura semplificata, con l'obiettivo di contenere la complessità numerica del modello così da avere una struttura inversa che permetta l'implementazione su piattaforma embedded, che in questo caso è il DSP di Analog Device "adau1701". Per l'implementazione, prima la parte non lineare è stata caratterizzata con una struttura polinomiale e la parte lineare con un filtro IIR (Infinite Impulsive Response) . In seguito dopo le valutazioni fatte nella sezione 3.4.2, è stato deciso di sostituire il blocco IIR con uno FIR per la parte lineare e mantenuta la struttura polinomiale per la parte non lineare.

3.1 Compensazione basata sul modello di Wiener

In questo lavoro di tesi è stato utilizzato un modello di Wiener per la compensazione di un sistema non lineare [29]. L' algoritmo proposto punta all'implementazione di un pre-distorsore di Wiener composto da una parte lineare di tipo IIR $p(\boldsymbol{z^{-1}})$ con in cascata un polinomio $f(.)$. Cosi come per il pre-distorsore anche il modello di Hammerstein è stato inizialmente modellato come cascata di un polinomio $g(.)$ e un filtro IIR $h(\boldsymbol{z^{-1}})$. L'adattamento è fatto attraverso l'algoritmo Non Linear Filtered X Least Mean Square (NL-FxLMS). Nella Figura 3.1, è illustrato lo schema a blocchi del sistema completo Wiener-Hammerstein.

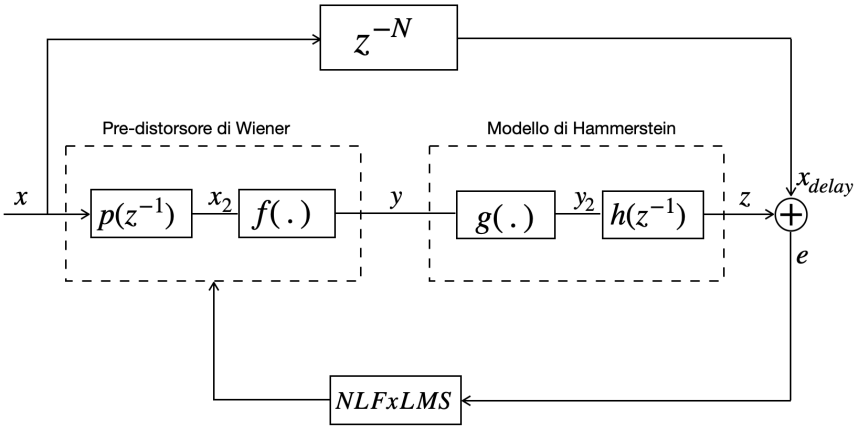


Figura 3.1: Schema a blocchi del sistema di riferimento implementato nell'algoritmo

La scelta iniziale di utilizzare Filtri IIR sia per la caratterizzazione che per la pre-distorsione è dettata dalla necessità di dover implementare il filtraggio su DSP. Lavorare con gli IIR rende possibile l'utilizzo di filtri ridotti a parità di prestazioni, quindi richiede meno risorse di calcolo e comporta una minore latenza tra ingresso ed uscita del sistema. Questo vantaggio si ha a discapito di una risposta in fase e quindi ritardo di gruppo non lineare che può generare effetti dannosi. Osservando lo schema a blocchi riportato in Figura 3.1, possiamo distinguere i seguenti segnali:

- x: rappresenta l'ingresso al pre-distorsore di Wiener ossia il segnale originale;
- x_2: grandezza intermedia del blocco Wiener, si ottiene filtrando l'ingresso con il filtro IIR adattivo;
- y: indica il segnale pre-distorto in uscita dal blocco di Wiener ed in ingresso al diffusore modellato con Hammerstein:
- y_2: grandezza intermedia del blocco Hammerstein, si ottiene applicando all'ingresso y il polinomio $g(.)$;
- z: è il segnale in uscita al sistema compensato che dovrebbe essere quanto più fedele a x una volta raggiunta la convergenza dell'algoritmo.

La definizione matematica del modello a bocchi di Figura 3.1 è discussa nella sezione 3.1.1.

3.1.1 Struttura del modello Wiener-Hammerstein

Seguendo a ritroso il flusso uscita-ingresso dello schema illustrato nella Figura 3.1, e implementando campione per campione possiamo definire la z come segue:

$$\begin{aligned} z(n) &= h(z^{-1})y_2(n) = \frac{B(z^{-1})}{1 - A(z^{-1})}y_2(n) \\ &= \sum_{m=0}^{m_b} b_m y_2(n-m) + \sum_{m=1}^{m_a} a_m z(n-m), \end{aligned} \tag{3.1}$$

dove la $h(z^-1)$ indica il filtro IIR della parte lineare del modello di Hammerstein, ed A e B sono i coefficienti che compongono il numeratore ed il denominatore di h e sono definiti come segue:

$$\begin{aligned} A(z^-1) &= \sum_{m=0}^{m_a} a_m(z^{-m}) \\ B(z^-1) &= \sum_{m=0}^{m_b} b_m(z^{-m}). \end{aligned} \tag{3.2}$$

Il segnale $y_2(n)$ intermedio è definito come:

$$\begin{aligned} y_2(n) &= g_1 y(n) + g_2 y^2(n) + \cdots + g_{m_g} y^m(n) \\ &= \boldsymbol{\theta}_g^T * \boldsymbol{y}(n). \end{aligned} \tag{3.3}$$

dove i vettori $\boldsymbol{\theta}_g^T$ e $\boldsymbol{y}(n)$ sono composti rispettivamente dai coefficienti del polinomio $\boldsymbol{g(.)}$ e dall'elevamento a potenza di $y(n)$ corrispondente all'ordine del polinomio come segue:

$$\boldsymbol{\theta}_g^T = (g_1, g_2, g_3, \ldots, g_{m_g})^T \tag{3.4}$$

$$\boldsymbol{y}(n) = (y(n), y^2(n), y^3(n), \ldots, y^{m_g}(n))^T. \tag{3.5}$$

Come si può notare la struttura proposta, come in [29], non è un modello generalizzato, ma composto da un polinomio e un filtro IIR. Perciò in fase di modellazione non si può utilizzare il modello introdotto da Antonin Novak ma un altro. La struttura

di Hammerstein scelta per l'identificazione è quella proposta in [33] nella forma full-band. Il blocco Wiener che modella il pre-distorsore è cosi strutturato: partendo dall'uscita si può definire il segnale *y(n)* come segue:

$$\begin{aligned} y(n) &= f_1(n)x_2(n) + f_2(n)x_2^2(n) + \cdots + f_{m_f}x_2^{m_f}(n) \\ &= \boldsymbol{\theta}_f^T * \boldsymbol{x_2}(n). \end{aligned} \tag{3.6}$$

Come per il caso precedente $\boldsymbol{\theta}_f^T$ e $\boldsymbol{x_2}(n)$ sono vettori che contengono rispettivamente i coefficienti del polinomio della parte non lineare del pre-distorsore ed il segnale $x_2(n)$ elevato a potenza dell'ordine corrispondente a m_f:

$$\boldsymbol{\theta}_f^T = (f_1(n), f_2(n), f_3(n), \ldots, f_{m_f}(n))^T \tag{3.7}$$

$$\boldsymbol{x_2}(n) = (x_2(n), x_2^2(n), x_2^3(n), \ldots, x_2^{m_f}(n))^T. \tag{3.8}$$

Il segnale intermedio $x_2(n)$ del pre-distorsore è invece dato dal filtraggio dell'ingresso con il filtro adattivo lineare del modello di Wiener:

$$\begin{aligned} x_2(n) &= p(z^{-1})x(n) = \frac{D(z^{-1})}{1 - C(z^{-1})}x(n) \\ &= \sum_{m=0}^{m_d} d_m x(n-m) + \sum_{m=1}^{m_c} c_m x_2(n-m). \end{aligned} \tag{3.9}$$

dove i polinomi D e C rappresentano rispettivamente i coefficienti del numeratore e denominatore del filtro IIR del modello di Wiener. Per comodità definiamo un parametro $\boldsymbol{\theta}$ definito come segue:

$$\begin{aligned}\boldsymbol{\theta} &= (\boldsymbol{\theta_f}\boldsymbol{\theta_c}\boldsymbol{\theta_d}) \\ \boldsymbol{\theta_f} &= (f_1 f_2 \dots f_{m_f}) \\ \boldsymbol{\theta_c} &= (c_1 c_2 \dots c_{m_c}) \\ \boldsymbol{\theta_d} &= (d_1 d_2 \dots d_{m_d}).\end{aligned} \tag{3.10}$$

Questo contiene i coefficienti del pre-distorsore che dovranno essere aggiornati con l'algoritmo NL-FxLMS nella procedura di adattamento. L'errore $\boldsymbol{e}$ è calcolato sottraendo al segnale $x_{delay}(n)$ definito in (3.11) *z(n)*:

$$x_{delay}(n) = x(n-\tau) + v(n). \tag{3.11}$$

Il primo termine dell'equazione (3.11) indica l'ingresso opportunamente ritardato per l'allineamento con l'uscita, mentre *v(n)* rappresenta lo zero-mean Additive White Gaussian Noise (AWGN).

3.1.2 Algoritmo NL-FxLMS

L'algoritmo NL-FxLMS è ottenuto applicando il metodo del gradiente stocastico [30], ed è utilizzato stimare i parametri dei vettori θ. La formula per l'aggiornamento dei coefficienti è la seguente:

$$\boldsymbol{\theta}(n+1) = \boldsymbol{\theta}(n) - \frac{\mu}{2} * \boldsymbol{\Delta}^T(n). \tag{3.12}$$

dove il termine μ indica lo *step-size*: una costante minore di 1 opportunamente scelta; ed il termine $\boldsymbol{\Delta}^T(n)$ è il vettore gradiente ed è definito come segue:

$$\boldsymbol{\Delta}^T(n) = \frac{de^2(n)}{d\boldsymbol{\theta}(n)} = -2e(n) * \frac{dz(n)}{d\boldsymbol{\theta}(n)}. \tag{3.13}$$

La complessità numerica deriva dal calcolo del termine $\frac{dz(n)}{d\boldsymbol{\theta}(n)}$. La *z(n)* è data dall'equazione 3.1 e derivando per $\boldsymbol{\theta}(n)$ si ottiene :

$$\frac{dz(n)}{d\boldsymbol{\theta}(n)} = \sum_{m=0}^{m_b} b_m \frac{dy_2(n-m)}{d\boldsymbol{\theta}(n)} + \sum_{m=1}^{m_a} a_m \frac{dz(n-m)}{d\boldsymbol{\theta}(n)}. \tag{3.14}$$

Assumendo uno *step-size* abbastanza piccolo da permettere che i coefficienti θ varino lentamente [29] [30] [31] [32], si possono considerare le seguenti approssimazioni:

$$\frac{dy_2(n-m)}{d\boldsymbol{\theta}(n)} \approx \frac{dy_2(n-m)}{d\boldsymbol{\theta}(n-m)}, \quad m = 0, 1, \ldots, m_b, \tag{3.15}$$

$$\frac{dz(n-m)}{d\boldsymbol{\theta}(n)} \approx \frac{dz(n-m)}{d\boldsymbol{\theta}(n-m)}, \quad m = 1, 2, \ldots, m_a. \tag{3.16}$$

Sostituendo queste ultime alla (3.14) si ottiene:

$$\frac{dz(n)}{d\boldsymbol{\theta}(n)} = \hat{\boldsymbol{h}}(z^{-1}) * \frac{dy_2(n)}{d\boldsymbol{\theta}(n)}. \tag{3.17}$$

La $\hat{\boldsymbol{h}}(z^{-1})$ è la stima della parte lineare del modello di Hammerstein ed è nota, quindi l'attenzione si sposta sul temine $\frac{dy_2(n)}{d\boldsymbol{\theta}(n)}$. Considerando le equazioni (3.3) e (3.5), possiamo sviluppare la derivata come indicato di seguito:

$$\begin{aligned} \frac{dy_2(n)}{d\boldsymbol{\theta}(n)} &= \boldsymbol{\theta}_g^T \frac{dy(n)}{d\boldsymbol{\theta}(n)} \\ &= \boldsymbol{\theta}_g^T \frac{dy(n)}{dy(n)} \frac{dy(n)}{d\boldsymbol{\theta}(n)} \\ &\approx s_1(n) \frac{dy(n)}{d\boldsymbol{\theta}(n)}. \end{aligned} \tag{3.18}$$

Il termine $s_1(n)$ introdotto può essere ricavato come segue:

$$s_1(n) = \hat{\theta}_g^T \frac{d\mathbf{y}(n)}{dy(n)} = \hat{\theta}_g^T \begin{pmatrix} 1 \\ 2y(n) \\ \vdots \\ m_g y^{m_g - 1}(n) \end{pmatrix}. \tag{3.19}$$

In questo caso il termine $\hat{\boldsymbol{\theta}}_{\boldsymbol{g}}$ sta ad indicare il vettore che contiene i coefficienti stimati del polinomio che identifica la non linearità del modello di Hammerstein. Poiché il vettore $\boldsymbol{\theta}$ è definito come indicato nella formula (3.10), possiamo riscrivere

la formula(3.17) considerando le derivate parziali degli elementi di $\boldsymbol{\theta}$:

$$\begin{aligned}\frac{dz(n)}{d\boldsymbol{\theta}(n)} &= \hat{\boldsymbol{h}}(z^{-1}) * s_1(n) * \frac{dy(n)}{d\boldsymbol{\theta}(n)} \\ &= \hat{\boldsymbol{h}}(z^{-1}) * s_1(n) * \left(\frac{\partial y(n)}{\partial \boldsymbol{\theta}_f(n)} \frac{\partial y(n)}{\partial \boldsymbol{\theta}_d(n)} \frac{\partial y(n)}{\partial \boldsymbol{\theta}_c(n)} \right).\end{aligned} \tag{3.20}$$

In questo caso il termine $\frac{dy(n)}{d\boldsymbol{\theta}_f(n)}$ può essere facilmente ricavato considerando le equazioni (3.6) (3.8) e (3.9), come riportato di seguito:

$$\frac{\partial y(n)}{\partial \boldsymbol{\theta}_f(n)} = \boldsymbol{x_2}^T(n) = \begin{pmatrix} \boldsymbol{p}(n.\boldsymbol{z}\hat{}[-1]x(n)) \\ [\boldsymbol{p}(n.\boldsymbol{z}\hat{}[-1]x(n))]^2 \\ \vdots \\ [\boldsymbol{p}(n.\boldsymbol{z}\hat{}[-1]x(n))]^{m_f} \end{pmatrix}. \tag{3.21}$$

Anche i termini $\frac{dy(n)}{d\boldsymbol{\theta}_d(n)}$ e $\frac{dy(n)}{d\boldsymbol{\theta}_c(n)}$ possono essere ricavati questa volta con qualche passaggio in più, poiché non abbiamo termini direttamente derivanti da $\boldsymbol{\theta_d}$ e $\boldsymbol{\theta_c}$. Dobbiamo introdurre un nuovo termine ausiliario che permette di ridurre la complessità del calcolo:

$$\begin{aligned}\frac{\partial y(n)}{\partial \boldsymbol{\theta}_d(n)} &= \boldsymbol{\theta}_f^T(n) \frac{\partial \boldsymbol{x_2}(n)}{\partial x_2(n)} \frac{\partial x_2(n)}{\partial \boldsymbol{\theta}_d(n)} = s_2(n) \frac{\partial x_2(n)}{\partial \boldsymbol{\theta}_d(n)} \\ \frac{\partial y(n)}{\partial \boldsymbol{\theta}_c(n)} &= \boldsymbol{\theta}_f^T(n) \frac{\partial \boldsymbol{x_2(n)}}{\partial x_2(n)} \frac{\partial x_2(n)}{\partial \boldsymbol{\theta}_c(n)} = s_2(n) \frac{\partial x_2(n)}{\partial \boldsymbol{\theta}_c(n)}.\end{aligned} \tag{3.22}$$

$s_2(n)$ è definito come segue:

$$s_2(n) = \boldsymbol{\theta}_f^T(n) \frac{\partial \boldsymbol{x_2}(n)}{\partial x_2(n)} = \boldsymbol{\theta}_f^T(n) \begin{pmatrix} 1 \\ 2x_2(n) \\ \vdots \\ m_f x_2^{m_f-1}(n) \end{pmatrix}. \tag{3.23}$$

Rimane quindi di calcolare i termini $\frac{\partial x_2(n)}{\partial \boldsymbol{\theta}_d(n)}$ e $\frac{\partial x_2(n)}{\partial \boldsymbol{\theta}_c(n)}$. Differenziando i due lati dell'equazione 3.9 rispettivamente per $d_k(n)$ e $c_k(n)$ si ottiene:

$$\begin{aligned}\frac{\partial x_2(n)}{\partial d_k(n)} &= x(n-k) + \sum_{m=1}^{m_c} c_m(n)\frac{\partial x_2(n-m)}{\partial d_k(n)} \\ \frac{\partial x_2(n)}{\partial c_k(n)} &= x_2(n-k) + \sum_{m=1}^{m_c} c_m(n)\frac{\partial x_2(n-m)}{\partial c_k(n)}.\end{aligned} \tag{3.24}$$

Assumendo che $\boldsymbol{\theta}$ vari lentamente e applicando le derivate parziali alla (3.24), possiamo definire le seguenti equazioni:

$$\frac{\partial x_2(n)}{\partial \boldsymbol{\theta}_d(n)} = \begin{pmatrix} \frac{\partial x_2(n)}{\partial d_0(n)} \\ \frac{\partial x_2(n)}{\partial d_1(n)} \\ \vdots \\ \frac{\partial x_2(n)}{\partial d_{m_d}(n)} \end{pmatrix}^T \approx \begin{pmatrix} \frac{1}{1-C(n,\boldsymbol{z^{-1}})}x(n) \\ \frac{\boldsymbol{z}^{-1}}{1-C(n,\boldsymbol{z^{-1}})}x(n) \\ \vdots \\ \frac{\boldsymbol{z}^{-m_d}}{1-C(n,\boldsymbol{z^{-1}})}x(n) \end{pmatrix}^T \tag{3.25}$$

$$\frac{\partial x_2(n)}{\partial \boldsymbol{\theta}_c(n)} = \begin{pmatrix} \frac{\partial x_2(n)}{\partial c_1(n)} \\ \frac{\partial x_2(n)}{\partial c_2(n)} \\ \vdots \\ \frac{\partial x_2(n)}{\partial c_{m_c}(n)} \end{pmatrix}^T \approx \begin{pmatrix} \frac{\boldsymbol{z}^{-1}}{1-C(n,\boldsymbol{z^{-1}})}[\boldsymbol{p}(n,\boldsymbol{z}^{-1})x(n)] \\ \frac{\boldsymbol{z}^{-2}}{1-C(n,\boldsymbol{z^{-1}})}[\boldsymbol{p}(n,\boldsymbol{z}^{-1})x(n)] \\ \vdots \\ \frac{\boldsymbol{z}^{-m_c}}{1-C(n,\boldsymbol{z^{-1}})}[\boldsymbol{p}(n,\boldsymbol{z}^{-1})x(n)] \end{pmatrix}^T. \tag{3.26}$$

Quindi dopo aver derivato tutti i termini è ora nota l'equazione (3.20) e quindi si dispone di tutti gli elementi per il calcolo del gradiente $\boldsymbol{\Delta}(n)$.

3.2 Modifiche al modello di riferimento

Nella sezione precedente è stato ben illustrato il modello di riferimento [29], considerato inizialmente per l'algoritmo di compensazione. Tuttavia, dopo alcuni test sono state effettuate delle modifiche. Il motivo è che per modelli con complessità ridotta si raggiungono buoni risultati in termini di errore quadratico medio, mentre se la complessità del sistema aumenta, il modello di riferimento non garantisce prestazioni accettabili. Con tale scopo il modello modificato è illustrato il Figura 3.2.

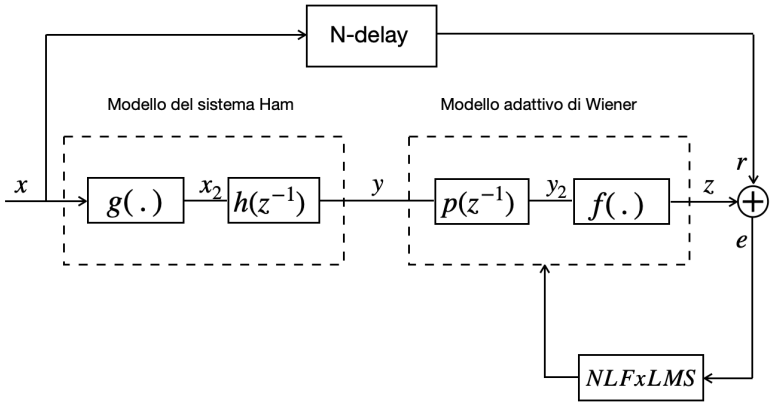


Figura 3.2: Schema a blocchi del modello modificato per l'implementazione con sistemi più complessi

Come si può notare, il modello di Hammerstein è considerato precedente al pre-distorsore. Questo comporta una modifica alle formule utilizzate per l'aggiornamento dei coefficienti dei blocchi del modello di Wiener. La scelta di modificare il modello è dettata dal fatto che con questa configurazione, si tiene conto del segnale già distorto per il calcolo del pre-distorsore, determinando una buona robustezza del sistema. Nello schema originale, il filtraggio per modello del diffusore è fatto dopo quello di Wiener, e di fatto alcuni coefficienti sono aggiornati senza considerare il sistema distorcente. Per sistemi semplici come ad esempio: tangente iperbolica o quello simulato in [29]; l'algoritmo ha buone prestazioni, tuttavia se si considera un modello più complesso si verifica la divergenza. Seguendo lo schema a blocchi illustrato in Figura 3.2 vengono definiti i cambiamenti nelle formule sopra citate nella sezione 3.1. In questo caso l'ingresso al pre-distorsore è indicato con il termine $\boldsymbol{y}$, cioè l'uscita del

sistema di Hammerstein generata dal segnale di ingresso $\boldsymbol{x}$. I segnali da considerare sono quindi:

$$x_2(n) = g_1 x(n) + g_2 x(^2n) + \cdots + g_{m_g} x^{m_g}(n) \tag{3.27}$$

$$y(n) = \frac{B(z^{-1})}{1 - A(z^{-1})} x_2(n) \tag{3.28}$$

$$y_2(n) = f_1(n) y(n) + f_2(n) y^2(n) + \cdots + f_{m_f} y^{m_f}(n) \tag{3.29}$$

$$z(n) = \frac{D(z^{-1})}{1 - C(z^{-1})} y_2(n). \tag{3.30}$$

Sostituendo queste ultime alle equazioni indicate nella sezione "Algoritmo NL-FxLMS" si possono facilmente ricavare i termini incogniti necessari per il calcolo dei coefficienti del pre-distorsore. Le modifiche effettuate al modello permettono di poter raggiungere la convergenza dell'algoritmo con sistemi la cui complessità è maggiore come quello considerato nel lavoro di tesi. Ovviamente, l'approccio descritto è utilizzato solo per l'adattamento del pre-distorsore poiché, essendo i sistemi di tipo causali, non posso effettuare una compensazione dopo il diffusore bensì, come previsto, l'ingresso $\boldsymbol{x}$ al diffusore viene pre-distorto in modo che l'uscita del sistema completo $\boldsymbol{z}$ risulti approssimabile all'ingresso $\boldsymbol{x}$.

3.3 Implementazione con filtri FIR

Fino ad ora per le parti lineari sia del modello dello speaker che del pre-distorsore, sono stati considerati come filtri IIR. Questi consentono di effettuare filtraggi notevoli con un numero ridotto di coefficienti. L'aspetto negativo è però il rischio di instabilità e il ritardo di fase e di gruppo non lineari. Inoltre, come descritto in seguito, per un corretto funzionamento dell'algoritmo è necessario che il filtro IIR sia sufficientemente lungo per poter invertire correttamente la risposta dell'altoparlante reale. Poiché il DSP utilizzato permette l'implementazione di filtraggi con filtri FIR relativamente lunghi, si è deciso di passare ad una implementazione di questo tipo. Data la differenza matematica tra i due tipi di filtraggio sono necessarie modifiche alle equazioni dell'algoritmo NL-FxLMS. I blocchi $\boldsymbol{h(z^{-1})}$ e $\boldsymbol{p(z^{-1})}$ ora FIR sono definiti

come segue:

$$\boldsymbol{h}(\boldsymbol{z}^{-1}) = \boldsymbol{B}(\boldsymbol{z}^{-1}) \tag{3.31}$$

$$\boldsymbol{p}(\boldsymbol{z}^{-1}) = \boldsymbol{D}(\boldsymbol{z}^{-1}). \tag{3.32}$$

I segnali considerati diventano quindi:

$$x_2(n) = g_1 x(n) + g_2 x(^2n) + \cdots + g_{m_g} x^{m_g}(n) \tag{3.33}$$

$$y(n) = B(z^{-1}) x_2(n) \tag{3.34}$$

$$y_2(n) = f_1(n) y(n) + f_2(n) y^2(n) + \cdots + f_{m_f} y^{m_f}(n) \tag{3.35}$$

$$z(n) = D(z^{-1}) y_2(n). \tag{3.36}$$

Sostituendo (3.33),(3.34),(3.35) e (3.36) alle equazioni dell'algoritmo NL-FxLMS e ponendo molta attenzione al calcolo delle derivate parziali, otteniamo tutti gli elementi per l'adattamento del modello di Wiener. Poiché i filtri hanno solo i coefficienti del numeratore, non è necessario considerare il termine $\boldsymbol{\theta_c}$. È possibile definire quindi la derivata di $\boldsymbol{z(n)}$ rispetto a $\boldsymbol{\theta}$ come segue:

$$\begin{aligned} \frac{dz(n)}{d\boldsymbol{\theta}(n)} &= \hat{\boldsymbol{h}}(z^{-1}) * s_1(n) * \frac{dy(n)}{d\boldsymbol{\theta}(n)} \\ &= \hat{\boldsymbol{h}}(z^{-1}) * s_1(n) * \left(\frac{dy(n)}{d\boldsymbol{\theta}_f(n)} \frac{dy(n)}{d\boldsymbol{\theta}_d(n)} \right). \end{aligned} \tag{3.37}$$

In questo caso la $\hat{\boldsymbol{h}}(z^{-1})$ è definita in (3.31) ed il termine moltiplicativo $s_1(n)$ si calcola come indicato in (3.19). Per quanto riguarda le derivate $\frac{dy(n)}{d\boldsymbol{\theta}_f(n)}$ e $\frac{dy(n)}{d\boldsymbol{\theta}_d(n)}$ il calcolo del primo termine rimane come illustrato nell'equazione (3.21), la derivata

rispetto a $\boldsymbol{\theta}_d$ invece è definita come segue:

$$\frac{\partial y(n)}{\partial \boldsymbol{\theta}_d(n)} = \boldsymbol{\theta}_f^T(n)\frac{\partial \boldsymbol{x_2}(n)}{\partial x_2(n)}\frac{\partial x_2(n)}{\partial \boldsymbol{\theta}_d(n)} = s_2(n)\frac{\partial x_2(n)}{\partial \boldsymbol{\theta}_d(n)}' \tag{3.38}$$

$$,\frac{\partial x_2(n)}{\partial \boldsymbol{\theta}_d(n)} = \begin{pmatrix} \frac{\partial x_2(n)}{\partial d_0(n)} \\ \frac{\partial x_2(n)}{\partial d_1(n)} \\ \vdots \\ \frac{\partial x_2(n)}{\partial d_{m_d}(n)} \end{pmatrix}^T \approx \begin{pmatrix} x(n) \\ \boldsymbol{z}^{-1}x(n) \\ \vdots \\ \boldsymbol{z}^{-m_d}x(n) \end{pmatrix}^T . \tag{3.39}$$

Nel paragrafo seguente è illustrata l'implementazione in Matlab di entrambi gli approcci con filtro IIR e FIR per la parte lineare e equazioni polinomiali per quella non lineare.

3.4 Implementazione dei codici Matlab

Dopo aver completato la procedura di studio dell'algoritmo si è passati alla scrittura del codice in Matlab per l'implementazione. Dapprima l'approccio utilizzato è quello mostrato in [29] e il codice è testato con sistemi noti semplici. Quindi, sono state implementate le modifiche sopra citate per garantire una maggior robustezza dell'algoritmo. Infine è stato implementato l'approccio FIR utilizzando gli stessi sistemi noti del caso IIR.

3.4.1 Implementazione con filtri IIR

Il primo step è quello di inizializzare le variabili costanti che indicano il numero di coefficienti utilizzati per i modelli, con i valori voluti come indicato nel codice seguente:

```
%%%% ordini dei filtri ed eventuale ritardo %%%%
M = 0; %ritardo
mc = 4; %ordine del den di p(z)
md = 4; %ordine del nma di p(z)
mf = 9; %ordine del polinomio di wiener
mg = 3; %ordine del polinomio di hammerstein
```

Listing 3.1: Parametri del filtro

Nel Codice 3.1 si possono cogliere vari elementi: per il filtro IIR di Wiener è stato scelto un ordine quattro sia per il numeratore che per il denominatore, mentre per quanto riguarda l'ordine del polinomio è stato utilizzato un ordine più alto pari a nove. Questo è dovuto al fatto che, poiché in genere non si conosce la struttura del sistema per cui si effettua la compensazione, è bene considerare un polinomio di ordine più alto. Il termine **M** indica il ritardo da specificare per l'allineamento del segnale originale con quello filtrato al momento del calcolo dell'errore. Un altro elemento da evidenziare è lo *step-size*. Quest'ultimo è un elemento fondamentale per l'aggiornamento dei coefficienti, ed in questo caso per l'algoritmo implementato è utilizzato senza alcuna operazione di normalizzazione rispetto alla potenza del segnale. Di seguito è riportata la porzione di codice che implementa l'inizializzazione dei coefficienti dei filtri.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% inizializzo i coefficienti dei filtri %%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%Modello Di Hammerstein%%%
num_hz = [0.8 0.77 0.5 0.2 ];                %H(z)
den_hz = [1 0.58 0.42 0.06 ];
tetaG= [ 1 0.5 0.25  ];                      %g(.)

%%%Modello Di Wiener%%%
num_dn = [ 1 zeros(1,md)];                   %P(z)
den_dn = [1 -zeros(1,mc)];
tetaF=[ 1 zeros(1,mf-1)];                    %f(.)

%%%Coefficienti teta del filtro di Wiener%
tetaD=[ 1 zeros(1,md)];                      %D(z)

tetaC=zeros(1,mc);                           %C(z)
```

Listing 3.2: Inizializzazione dei coeficienti dei filtri

Come si può notare nel codice Codice 3.2 il modello di Hammerstein è stato inizializzato con coefficienti noti, indicati nelle variabili **num-hz** e **den-hz**, risulta quindi come segue:

$$z(n) = \frac{0.8 + 0.77z^{-1} + 0.5z^{-3} + 0.2z^{-3}}{1 + 0.58z^{-1} + 0.42z^{-2} + 0.06z^{-3}} y_2(n). \tag{3.40}$$

La variabile **tetaG** invece è il vettore delle costanti moltiplicative del polinomio di Hammerstein ed ha la seguente forma:

$$y_2(n) = y(n) + 0.5y^2(n) + 0.25y^3(n). \tag{3.41}$$

Ponendo l'attenzione alla porzione dedicata alla parte Wiener, si nota che il numeratore del filtro lineare è inizializzato come vettore di zeri, il denominatore e la variabile delle costanti del polinomio sono inizializzati come vettori di zeri lunghi ***mc*** per **den-dn** e ***mf-1*** per **tetaF** con il primo valore pari ad uno. Le variabili **tetaD** e **tetaC** sono i vettori che conterranno i coefficienti aggiornati del filtro lineare del pre-distorsore e sono inizializzati allo stesso modo di **num-dn** e **den-dn**. Nella Figura **??** sono riportare le operazioni di filtraggio effettuate al primo ciclo, quando tutti i filtri sono inizializzati. La porzione di codice implementa direttamente il modello modificato e quindi compaiono più segnali rispetto allo schema a blocchi originale, necessari per le operazioni inerenti al NL-FxLMS.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% fltraggio iniziale %%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%

delX=M;% delay per l'allineamento dei segnali

delayXR=zeros(1,delX)';%creazione del vettore di zeri per il ritardo

xDel=[delayXR;xR(1:end-delX)];% segnale di ingresso ritardato

y2 = nonlinear_system(xR, tetaG);%filtraggio per g(.)

xZ=filter(num_hz,den_hz,y2);%filtraggio per h(z)

x2=filter(num_dn,den_dn,xZ);%filtraggio per f(.)

y = nonlinear_system(x2, tetaF);%filtraggio per p(z)
```

Listing 3.3: Operazioni di filtraggio pre-elaborazione

Le prime tre righe del codice calcolano il segnale originale ritardato necessario per l'allineamento con quello filtrato nel calcolo dell'errore. La procedura avviene

semplicemente inserendo **M** zeri in testa al vettore **xR**.

```
%%%%%%%%%%%%%%%%%%%%%%
%%% N di iterazioni %%%
%%%%%%%%%%%%%%%%%%%%%%

for i=1:200 % epoche

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% inizializzazione dei ritardi per i filtraggi nell'LMS%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    tic
    del3=zeros(md+1,length(den_dn));
    del4=zeros(mc,length(den_dn));
    DEL_FILTH=zeros(1,max(length(num_hz),length(den_hz))-1);
    del_dzF=zeros(1,max(length(num_hz),length(den_hz))-1);
    del_dzC=zeros(1,max(length(num_hz),length(den_hz))-1);
    del_dzD=zeros(1,max(length(num_hz),length(den_hz))-1);
    DEL_FILTp=zeros(1,max(length(num_dn),length(den_dn))-1);

    tetaF=[ 1 zeros(1,mf-1)];
    tetaD=[ 1 zeros(1,md)];
    tetaC=zeros(1,mc);

    num_dn = tetaD;
    den_dn = [1,-tetaC];
```

Listing 3.4: Codice per l'inizializzazione dei ritardi per il filtraggio

Il `Codice 3.4` illustra la porzione di codice che implementa il numero di iterazioni da effettuare per l'adattamento del sistema e l'inizializzazione dei ritardi utili in fase di filtraggio all'interno dell'algoritmo di LMS. Quest'ultimo lavora sample-by-sample, quindi poiché tutti i filtraggi sono fatti con il comando del Matlab "*filter*", è necessario costruire un vettore di ritardi in modo tale che nel filtraggio del campione corrente si tenga conto di quelli precedenti. Inoltre vengono nuovamente inizializzati i vettori dei coefficienti dei filtri e impostati il numeratore ed il denominatore del filtro lineare di Wiener. Questa procedura non è utile nella prima iterazione ma dalle successive. Di seguito nel `Codice 3.5` è mostrato il calcolo dei coefficienti $\boldsymbol{s}_1$ e $\boldsymbol{s}_2$ definiti nelle

equazioni (3.19),(3.23).

```
%%%%%%%%%%%%%%%
%%% NL-FxLMS %%%
%%%%%%%%%%%%%%%
for n=1:N

    for k1=1:mg    % calcolo S1

        s1mat(k1)=(k1*y(n)^(k1-1));

    end

    S1= tetaG*s1mat';

    for k2=1:mf % calcolo S2

        s2mat(k2)=(k2*(x2(n)^(k2-1)));

    end

    S2= tetaF*s2mat';
```

Listing 3.5: Codice per il calcolo dei coefficienti $\boldsymbol{s}_1$ e $\boldsymbol{s}_2$

In particolare in Matlab sono state separate le operazioni calcolando dapprima, con un ciclo iterativo, il vettore che contiene le derivate fino all'ordine $\boldsymbol{m}_g$-esimo e $\boldsymbol{m}_f$-esimo del segnale (***s1mat*** e ***s2mat***), e successivamente la moltiplicazione per il vettore delle costanti del polinomio corrispondente (***S1*** e ***S2***).

I segnali $y(n)$ e $x_2(n)$ sono calcolati come indicato nel `Codice 3.3`. Il prossimo step è calcolare le derivate parziali $\frac{\partial x_2(n)}{\partial \boldsymbol{\theta}_c(n)}$ e $\frac{\partial x_2(n)}{\partial \boldsymbol{\theta}_d(n)}$ con il `Codice 3.6` mostrato di seguito.

```
for k3=1:md+1 %calcolo dx2(n)/dOd(n)

    num =[zeros(1,k3-1),1];

    del0=max(length(num),length(den_dn))-1;

    [x_der(k3),del3(k3,1:del0)]= filter(num,den_dn, xZ(n),del3(k3,1:
    del0));
end
```

```
9
10 for k4=1:mc %calcolo dx2(n)/dOc(n)
11
12     num =[zeros(1,k4),1];
13
14     del1=max(length(num),length(den_dn))-1;
15
16     [x_der2(k4),del4(k4,1:del1)]= filter(num,den_dn,x2(n),del4(k4,1:
       del1));
17
18 end
19
20 for j=1:mf     % calcolo X2 vettore
21
22     X2(j)= x2(n)^j;
23
24 end
```

Listing 3.6: Codice per il calcolo derivate parziali $\frac{\partial x_2(n)}{\partial \boldsymbol{\theta}_c(n)}$ e $\frac{\partial x_2(n)}{\partial \boldsymbol{\theta}_d(n)}$

Come nel precedente, anche in questo caso le operazioni sono state separate per semplicità di implementazione. Si possono notare alcune variabili ausiliarie come ad esempio *num*. Quest'ultima indica il vettore dei ritardi, raggiunge la lunghezza massima pari all'ordine del denominatore del filtro di Wiener. Le variabili *del0* e *del1* sono costanti che indicano la lunghezza massima del ritardo da applicare in fase di filtraggio. *del0* e *del1* sono da tenere conto per non commettere errori di filtraggio nella riga di codice successiva. Queste variabili sono definite all'interno dei cicli *for* poiché devono tenere conto della lunghezza del *num*, che aumenta progressivamente da uno a $m_d + 1$ o m_c ad ogni iterazione. L'ultimo termine incognito $\boldsymbol{X2}$, definito in nella formula (3.21), è ricavato nelle ultime tre righe del `Codice 3.6` e corrisponde alla derivata $\frac{\partial y(n)}{\partial \boldsymbol{\theta}_f(n)}$. Una volta ottenute le equazioni delle derivate parziali $\frac{\partial x_2(n)}{\partial \boldsymbol{\theta}_c(n)}$ e $\frac{\partial x_2(n)}{\partial \boldsymbol{\theta}_d(n)}$ è possibile calcolare i termini $\frac{\partial y(n)}{\partial \boldsymbol{\theta}_d(n)}$ e $\frac{\partial y(n)}{\partial \boldsymbol{\theta}_c(n)}$ come illustrato nella porzione di `Codice 3.7`. Questo implementa le equazioni indicate in (3.22) dove $s_2(n)$ corrisponde a $\boldsymbol{S2}$ ed i termini $\frac{\partial x_2(n)}{\partial \boldsymbol{\theta}_c(n)}$ e $\frac{\partial x_2(n)}{\partial \boldsymbol{\theta}_d(n)}$ corrispondono rispettivamente a $\boldsymbol{x\_der2}$ e $\boldsymbol{x\_der}$.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Calcolo delle derivate parziali di y(n) %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

dY_dTetaf = X2;
dY_dTetac = S2 * x_der2;
dY_dTetad = S2 * x_der;
```

Listing 3.7: Codice per il calcolo derivate parziali $\frac{\partial y(n)}{\partial \boldsymbol{\theta}_d(n)}$ e $\frac{\partial y(n)}{\partial \boldsymbol{\theta}_c(n)}$

Una volta ottenute le derivate parziali posso ricavare il termine $\frac{dz(n)}{d\boldsymbol{\theta}(n)}$ come illustrato nell'equazione (3.20). Anche in questo caso per semplicità di scrittura le operazioni sono state separate.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Costruzione del vettore delle derivate parziali %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

dzpF(1,:) = S1 * dY_dTetaf.';
dzpC(1,:) = S1 * dY_dTetac.';
dzpD(1,:) = S1 * dY_dTetad.';

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Filtro di dzp per ottenere dz  %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[dzF(1,:),del_dzF] = filter(num_hz, den_hz, dzpF(1,:),del_dzF);
[dzC(1,:),del_dzC] = filter(num_hz, den_hz, dzpC(1,:),del_dzC);
[dzD(1,:),del_dzD] = filter(num_hz, den_hz, dzpD(1,:),del_dzD);
```

Listing 3.8: Codice per il calcolo della derivata $\frac{dz(n)}{d\boldsymbol{\theta}(n)}$

La porzione di codice nominata "Costruzione del vettore delle derivate" implementa la moltiplicazione tra il termine $s_1(n)$ con le singole derivate $\frac{\partial y(n)}{\partial \boldsymbol{\theta}_f(n)}$, $\frac{\partial y(n)}{\partial \boldsymbol{\theta}_d(n)}$ e $\frac{\partial y(n)}{\partial \boldsymbol{\theta}_c(n)}$. Per semplicità, i termini sono considerati separatamente quindi al momento del calcolo di Δ e per l'aggiornamento dei coefficienti avremo tre equazioni. Il codice nominato "Filtraggio dei vettori delle derivate" esegue il filtraggio delle singole componenti per il filtro lineare di Hammerstein. Poiché anche in questo caso l'operazione viene fatta

con singoli campioni, devo considerare i ritardi per la funzione *filter* rispettivamente *del_dzF*, *del_dzC* e *del_dzD*.

```
% Aggiornamento dei coefficienti %


mu(n) = muOG;

e(n)=xDel(n)-y(n); % Calcolo dell'errore

% Calcolo dei vettori gradiente

deltaF(1,:)= -2*e(n).* dzpF(1,:)'; % vettore gradiente di F
deltaC(1,:)= -2*e(n).* dzpC(1,:)'; % vettore gradiente di C
deltaD(1,:)= -2*e(n).* dzpD(1,:)'; % vettore gradiente di D

% Aggiornamento Coefficienti del Filtro adattivo

tetaF=tetaF-((0.5*mu(n)).* deltaF(1,:)); %Aggiornamento dei
    coerrifienti  tetaF
tetaD=tetaD-((0.5*mu(n)).* deltaD(1,:));%Aggiornamento dei
    coerrifienti  tetaD
tetaC=tetaC-((0.5*mu(n)).* deltaC(1,:));%Aggiornamento dei
    coerrifienti  tetaC
```

Listing 3.9: Codice per il calcolo dell'errore e aggiornamento dei coefficienti

Nel `Codice 3.9` viene riportata la procedura di aggiornamento dei coefficienti del filtro e del polinomio del pre-distorsore. Si parte dall'aggiornamento del ***"mu"*** (*step-size*): in questo caso il parametro è considerato costante tuttavia, è possibile normalizzarlo rispetto alla potenza del segnale per evitare cambiamenti troppo elevati in fase di aggiornamento. Di seguito viene calcolato l'errore puntuale per il campione n-esimo. È bene notare che "**e**" è calcolato sottraendo l'uscita del sistema all'ingresso ritardato cosi da compensare ritardi dovuti al filtraggio e garantire l'allineamento dei segnali. Quindi si procede con il calcolo del gradiente (effettuato per i singoli vettori) ed infine all'aggiornamento dei coefficienti come illustrato nella porzione "Aggiornamento Coefficienti del filtro adattivo". Dopo aver calcolato ($\boldsymbol{\theta_f}$ $\boldsymbol{\theta_c}$ $\boldsymbol{\theta_d}$) aggiornati, è necessario impostare i nuovi valori del numeratore (***num_dn***), ed del

denominatore (***den_dn***) cosi da effettuare il filtraggio del segnale sia per la nuova parte lineare che non lineare e quindi ripetere l'iterazione per tutta la lunghezza del segnale. Il `Codice 3.10` implementa questa procedura.

```
% Settaggio dei coefficienti aggiornati dei filtri %

num_dn = tetaD; %settaggio del numeratore di p(z^(-1)) con i
    coefficienti aggiornati
den_dn = [1,-tetaC]; %settaggio del denominatore di p(z^(-1)) con i
    coefficienti aggiornati

% Filtraggio per il filtro ed il polinomio aggiornati
x2=filter(num_dn,den_dn,xZ); %filtraggio per la parte lineare del pre-
    distorsore
y = nonlinear_system(x2, tetaF); %filtraggio per la parte non lineare
    del pre-distorsore
```

Listing 3.10: Codice per la costruzione del $\boldsymbol{p}(\boldsymbol{z}^{-1})$aggiornato e filtraggio dei segnali per l'iterazione successiva.

Per i filtraggi con le parti non lineari è stata realizzata una funzione Matlab che effettua iil calcolo del polinomio di ordine N ed è illustrata nel `Codice 3.11`

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Funzione del sistema non lineare (polinomiale) %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function y_nl = nonlinear_system(x, coeff)
    y_nl = 0;
    for i = 1:length(coeff)
        y_nl = y_nl + coeff(i) * x.^i;
    end
end
```

Listing 3.11: Funzione per il filtraggio non lineare

Per eseguire il filtraggio non lineare, è sufficiente chiamare la funzione "nonlinear_system" passandogli il segnale da filtrare ed i coefficienti del polinomio desiderato.

3.4.2 Implementazione con filtri FIR

Dopo aver testato il funzionamento dell'algoritmo con filtri di tipo IIR è stato deciso di ottimizzare il modello stimando la parte lineare con filtri FIR. La scelta è stata influenzata dal fatto che l'identificazione del sistema reale è basata sul modello full-band introdotto in [33]. L'algoritmo citato implementa il modello di Hammerstein composto da un polinomio con in cascata un filtro FIR, perciò è stato deciso di utilizzare lo stesso tipo di filtro per il modello di Wiener. Inoltre, per invertire fedelmente il sistema occorre utilizzare un numero di coefficienti elevato anche con filtri IIR rendendo non più così vantaggiosa la scelta di questo tipo di filtri. Per tale ottimizzazione è stato necessario fare dei cambiamenti all'algoritmo. In primo luogo sono stati impostati i nuovi valori di ordine del filtro e del polinomio del pre-distorsore `Codice 3.12`.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ordini dei filtri ed eventuale ritardo %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

M=5;%ritardo
mc=1;%ordine del den di p(z)
md=65;%ordine del nma di p(z)
mf=3;%ordine del polinomio di Wiener
mg=3;%ordine del polinomio di Hammerstein

%step size
muOG=[0.03];
```

Listing 3.12: Scelta dell'ordine del filtro FIR e del polinomio.

Poiché FIR, il dato m_d, che indica l'ordine di $\boldsymbol{p}(\boldsymbol{z}^{-1})$, risulterà più grande rispetto al caso precedente, ed il temine m_c risulta pari ad 1 poiché non sono presenti coefficienti al denominatore. Nel `Codice 3.13` è illustrata l'inizializzazione dei coefficienti Wiener-Hammerstein.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% inizializzo i coefficienti dei filtri %%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%Modello Di Hammerstein%%%
num_hz=FilterCoeff;                                %h(z)
den_hz=1;
tetaG= [ 1 0.5 0.25  ];

%%%Modello Di Wiener%%%
num_dn = [ 1 zeros(1,md-1)];                       %p(z)
den_dn =1;
tetaF=[ 1 zeros(1,mf-1)];

%%%Coefficienti teta del filtro di Wiener%
tetaD=[ 1 zeros(1,md-1)];
tetaC=1;
```

Listing 3.13: Inizializzazione coefficienti dei filtri e dei polinomi per l'implementazione FIR.

Come si nota in questo caso sono presenti 2 vettori di coefficienti per la parte lineare uno per la $\boldsymbol{h}(\boldsymbol{z}^{-1})$ e uno per la $\boldsymbol{p}(\boldsymbol{z}^{-1})$. Entrambi i denominatori sono fissati ad uno. Per quanto riguarda i polinomi $g(.)$ e $f(.)$ non sono previsti cambiamenti. Poiché il numeratore della $\boldsymbol{h}(\boldsymbol{z}^{-1})$ è composto da più coefficienti rispetto al caso IIR, è stato inizializzato con il vettore ***FilterCoeff***, che contiene i tappi del FIR che modella la parte lineare di Hammerstein in questa fase nota. Facendo riferimento all'equazione (3.37) le derivate da calcolare sono due poiché non sono presenti i coefficienti $\boldsymbol{\theta_c}$. Ponendo l'attenzione al `Codice 3.14` si nota che in primo luogo viene effettuato il calcolo dei vettori $\frac{\partial x_2(n)}{\partial \boldsymbol{\theta}_d}(n)$ e $\boldsymbol{x_2}(n)$ per poi ricavare i termini $\frac{dy(n)}{d\boldsymbol{\theta}_f(n)}$ e $\frac{dy(n)}{d\boldsymbol{\theta}_d(n)}$ richiesti per l'equazione (3.37).

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Calcolo delle derivate parziali di y(n) %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for k3=1:md  %calcolo dx2(n)/dOd(n)
    num =[zeros(1,k3-1),1];
    del0=max(length(num),length(den_dn))-1;
```

```
8
9       [x_der(k3),del3(k3,1:del0)]= filter(num,den_dn, xZ(n),del3(k3,1:
        del0));
10  end
11
12
13  for j=1:mf     % calcolo X2 vettore
14
15      X2(j)= x2(n)^j;
16
17  end
18
19  dY_dTetaf = X2;
20  dY_dTetad = S2 * x_der;
```

Listing 3.14: Calcolo derivate parziali nell'implementazione FIR

Rimane quindi l'aggiornamento dei coefficienti dei filtri. Come per l'implementazione IIR anche in questo caso i vettori sono calcolati in due fasi calcolando prima i vettori gradiente **deltaF** e **deltaD**, e poi i vettori ***tetaF*** e ***tetaD*** aggiornati. Quindi viene settato il numeratore del filtro $\boldsymbol{p}(\boldsymbol{z}^{-1})$ **num__dn** per poter effettuare il filtraggio dell'iterazione successiva. Il `Codice 3.15` esegue il calcolo descritto.

```
1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2  % Aggiornamento dei Coefficienti %
3  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
4
5  %Calcolo dei vettori gradiente
6  deltaF(1,:)= -2*e(n).* dzpF(1,:)'; % vettore gradiente di F
7  deltaD(1,:)= -2*e(n).* dzpD(1,:)'; % vettore gradiente di D
8
9  %Calcolo dei nuovi coefficienti
10
11 tetaF=tetaF-((0.5*mu(n)).* deltaF(1,:)); %Aggiornamento dei
       coerrifienti  tetaF
12 tetaD=tetaD-((0.5*mu(n)).* deltaD(1,:)); %Aggiornamento dei
       coerrifienti  tetaD
13
14
```

```
15
16 %Aggiornamento del filtro p(z)
17
18 num_dn = tetaD; %settaggio del numeratore di p(z) con i coefficienti
       aggiornati
19 den_dn = 1;
```

Listing 3.15: Calcolo e aggiornamento dei coefficienti teta.

Capitolo 4

Modellazione dell'Array Circolare

Dopo aver completato la scrittura del codice matlab ed i test con modelli ideali noti, si è passati alla caratterizzazione e compensazione del diffusore reale progettato e realizzato da RUIZ-TECH: Buddy-Sound v1.0. Quest'ultimo è pensato per la diffusione audio a 360° e lo scopo finale è quello di garantire una riproduzione nello spazio con una distorsione quanto più possibile attenuata. Il Buddy-Sound v1.0 è composto da un array circolare di altoparlanti che comprende otto woofer e quattro tweeter intervallati con la sequenza: woofer-tweeter-woofer mostrata in Figura 4.1 replicata quattro volte.

Figura 4.1: Diffusore Buddy-sound v1.0 che riproduce il range 80Hz-20kHz

Le misure sono state effettuate con singola sezione e con più sezioni in funzione,

con lo scopo di testare il funzionamento dell'algoritmo alla presenza di un livello maggiore di distorsioni dovute alle vibrazioni delle pareti del diffusore, ed a potenze diverse per valutare la compensazione delle armoniche introdotte dagli altoparlanti. Queste misure sono state fatte solo a scopo di studio, di fatto l'elaborazione finale è effettuata su una singola sezione a potenza massima e poi replicata per le restanti. Il diffusore in questione integra anche un subwoofer, per la diffusione sonora nel range 30Hz-80Hz, composto da un elemento attivo ed un radiatore passivo. Il subwoofer è lasciato in funzione per ogni misura in modo da testare il corretto funzionamento dell'algoritmo anche per le basse frequenze e per attenuare anche le armoniche introdotte sia dall'altoparlante attivo e passivo sia dal box che deve attutire notevoli vibrazioni in più rispetto alla parte che riproduce lo spettro 80Hz-20kHz.

4.1 Misure in camera semi-anecoica

Il primo passo per poter eseguire l'elaborazione è la caratterizzazione del diffusore. A tale scopo è stato allestito un set-up di misura nella camera semi-anecoica del laboratorio A3lab appartenente al Dipartimento di Ingegneria dell'Informazione dell'Università Politecnica delle Marche, come mostrato in Figura 4.2.

Figura 4.2: Camera semi-anecoica del laboratorio A3lab

La camera è qualificata secondo la norma ISO 3745 e consente di eseguire analisi acustiche in condizioni di campo libero. Le dimensioni della camera sono 9 m x 7 m x 5 m ed è fornita di dispositivi di sospensione, ricircolo dell'aria, scarico di liquidi, utili ad accogliere sistemi da elaborare di vario genere. Il primo step è stato quello di allestire il set-up di misura. Il diffusore Buddy-Sound v1.0 è stato posizionato in prossimità di una parete della camera e sopra un tappeto, quest'ultimo per diminuire le riflessioni dovute al pavimento. Quindi l'area circostante è stata integrata con alcuni coni fonoassorbenti per migliorare le condizioni di misura. Per l'acquisizione è stato utilizzato un microfono Behringer modello B-5 che presenta una risposta circa lineare nel range di frequenze 50Hz-20kHz con range di funzionamento 20Hz-20kHz. In Figura 4.3 è rappresentato il digramma di sensibilità del microfono e la sua risposta in frequenza.

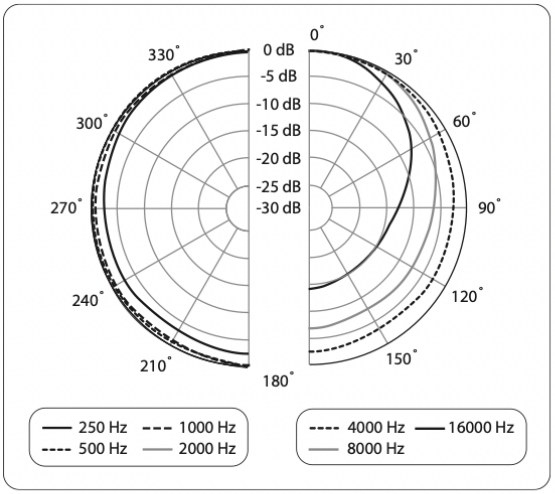
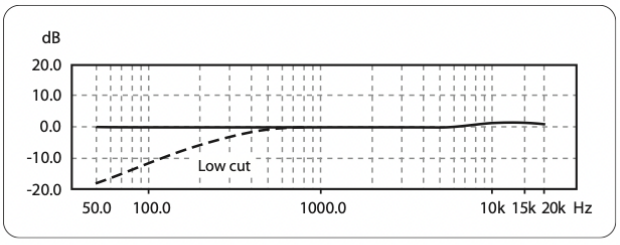


Diagramma polare (omnidirezionale)



Andamento in frequenza (omnidirezionale)

Figura 4.3: Risposta del microfono Behringer B-5

Il microfono è collegato in ingresso ad una scheda audio "Focusrite" modello "scarlet

2i2" illustrata in Figura 4.4, che permette di interfacciarsi al software proprietario NU-Tech di Leaff engeneering che è stato utilizzato per l'acquisizione.

Figura 4.4: Scarlet 2i2 Focusrite

In Figura 4.5 è illustrato il set-up di misura utilizzato per l'acquisizione in camera semi-anecoica.

Figura 4.5: Set-up di misura in camera

Il microfono per l'acquisizione è stato posizionato ad un metro dal centro di una sezione del diffusore, vale a dire allineato ad uno dei quattro tweeter affiancato dai due woofer. La risposta del diffusore in questo caso è acquisita ponendo come ingresso al sistema un segnale del tipo rumore bianco gaussiano generato in NU-Tech e misurando la risposta tramite il microfono con uno schema che verrà illustrato in seguito. Oltre all'uscita dal diffusore, è necessario salvare il rumore bianco originale dato in ingresso al sistema. La risposta misurata e il segnale di riferimento, verranno quindi salvati ed importati in Matlab per eseguire l'identificazione con il modello di Hammerstein full-band. Poiché il sistema di misura introduce una latenza non trascurabile, prima dell'identificazione è necessario allineare i due segnali inserendo un preciso numero di zeri in testa al riferimento. In Figura 4.6 è illustrato lo schema NU-Tech utilizzato per l'acquisizione della risposta del diffusore. Dopo aver calcolato

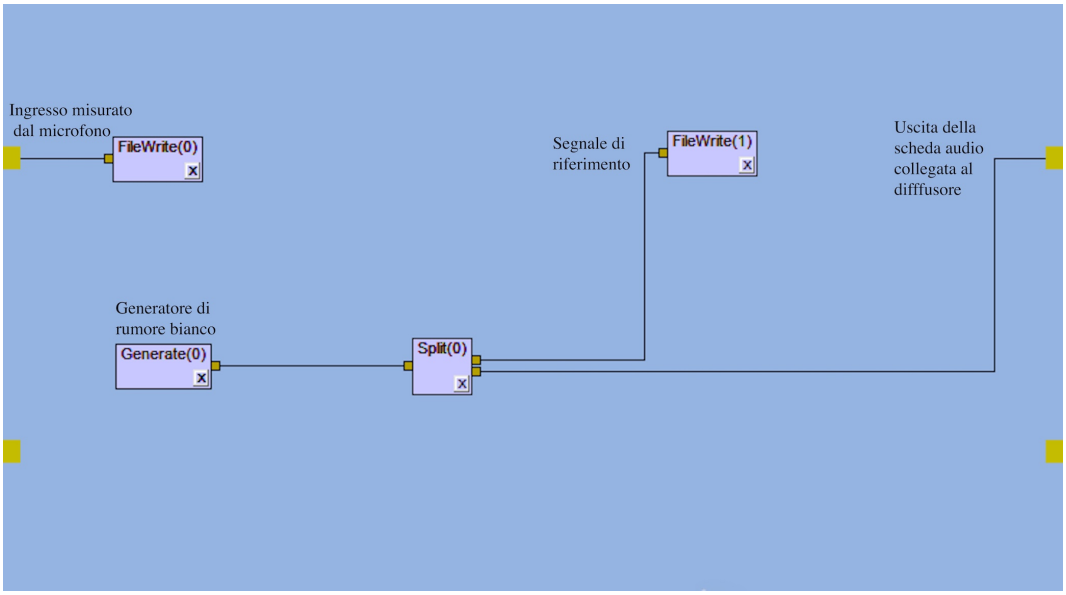


Figura 4.6: Schema NU-Tech per l'acquisizione della risposta

la parte lineare e non lineare del modello è possibile procedere con l'adattamento del pre-distorsore mediante l'algoritmo proposto nel capitolo 3. Con il sistema reale entrano in gioco contributi non nulli di latenza e rumore ambientale dovuti alla misura che ed andranno a modificare le prestazioni dell'algoritmo. Per quanto riguarda il ritardo introdotto è possibile compensarlo impostando il valore M nella porzione `Codice 3.12`, per il rumore additivo invece non è stata effettuata alcuna correzione poiché, essendo all'interno di un ambiente anecoico isolato acusticamente

dall'ambiente esterno, non è presente una quantità di rumore tale da richiedere una compensazione. Nel caso in cui la misura avvenisse in ambiente esterno il rumore risulterebbe non più trascurabile per l'identificazione. Il modello di Hammerstein full-band è composto da un FIR per la parte lineare ed un polinomio per il blocco non lineare. Per completezza, nei test con l'algoritmo IIR, seguendo [29], la parte lineare del modello di Hammerstein, risultante dalla procedura di caratterizzazione, è stata convertita da FIR ad IIR con l'ausilio di due funzioni disponibili in Matlab: prony ed lpc, in modo da avere un sistema computazionalmente più efficiente. Sono state utilizzare entrambe per confrontare la fedeltà della conversione a pari numero di ordine di numeratore e denominatore del filtro. La risposta misurata è lunga 256 campioni. Questo parametro è arbitrario ed è possibile cambiarlo prima della fase di caratterizzazione. Per rendere più snello l'algoritmo è stata inoltre compensata la latenza iniziale dovuta alla misura reale e troncata la coda del filtro lineare, così da accorciare la risposta a 65 campioni. Sia con prony che con lpc, per una buona approssimazione della parte lineare, è necessario un numero di coefficienti del numeratore e del denominatore elevato, quindi è stato deciso di utilizzare il modello di Hammerstein full-band originale accettando un costo computazionale leggermente maggiore in cambio di una buona stabilità e di una risposta in fase lineare. Allo stesso modo, per ottenere dei buoni risultati di inversione della parte lineare con Wiener IIR gli ordini del numeratore e del denominatore (rispettivamente $\boldsymbol{m_d}$ e $\boldsymbol{m_c}$) devono essere elevati, perciò, anche per il pre-distorsore, si è preferito procedere con l'implementazione con blocco lineare FIR. Importato il modello nel codice Matlab è stata avviata la procedura di adattamento. Nello specifico per la compensazione è stato utilizzato un polinomio del terzo ordine in quanto non c'è interesse a compensare non linearità superiori, ed un filtro FIR lungo 65 coefficienti. Il delay è stato appositamente impostato per consentire un corretto allineamento in fase di elaborazione e lo *step-size* migliore è pari al valore 0.03. Il test è stato effettuato anche per altri valori μ ed è stato osservato che: aumentando il valore di questo parametro l'algoritmo inizia a divergere, al contrario la convergenza è raggiunta con valori di errore quadratico medio in ***dB*** più elevati. I coefficienti del blocco lineare e non lineare della parte adattiva sono aggiornati campione per campione per tutta la lunghezza del segnale e l'iterazione è ripetuta per 100 volte. Nella Figura 4.7 è

illustrato l'andamento della risposta impulsiva del diffusore modellata con il metodo indicato in [33].

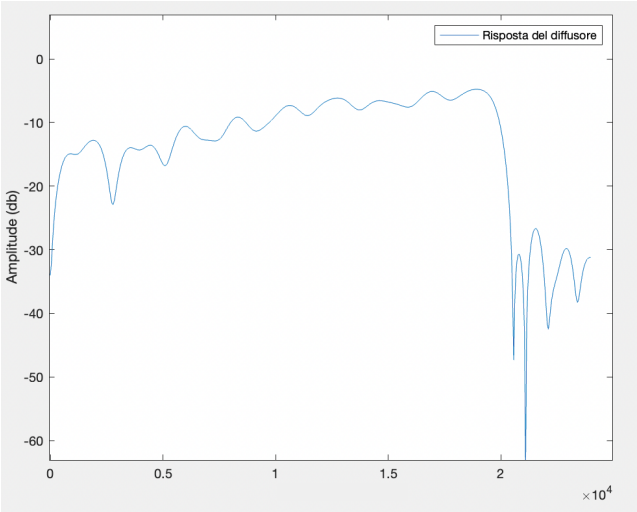


Figura 4.7: Trasformata di Fourier della risposta del modello di Hammerstein del diffusore reale

La convergenza dell'algoritmo è raggiunta con i sistemi: ideale noto e reale misurato. Nel primo caso, l'errore quadratico medio raggiunto è di circa -39dB. I sistemi simulati sono descritti da funzioni semplici come ad esempio la tangente iperbolica. Nel secondo caso l'errore quadratico medio risulta circa -13dB. Questo risultato è notevole considerando il sistema reale che risulta molto più complesso rispetto a quelli simulati. L'andamento dell'errore quadratico medio durante la convergenza dell'algoritmo è mostrato in Figura 4.9, per il sistema ideale, e in Figura 4.8, per il sistema reale.

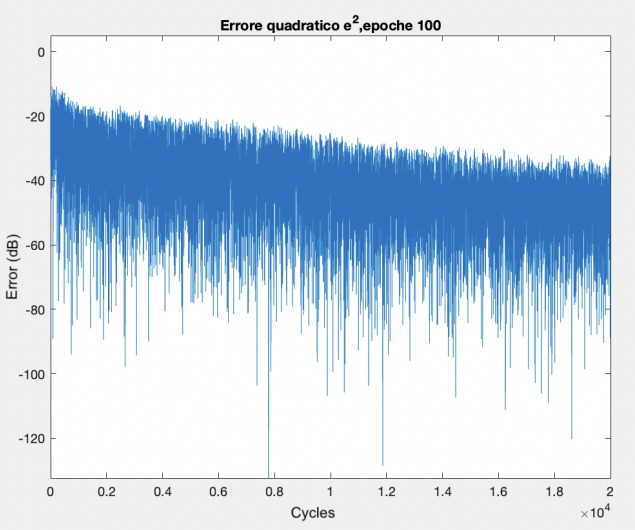


Figura 4.8: Andamento dell'errore con sistema ideale

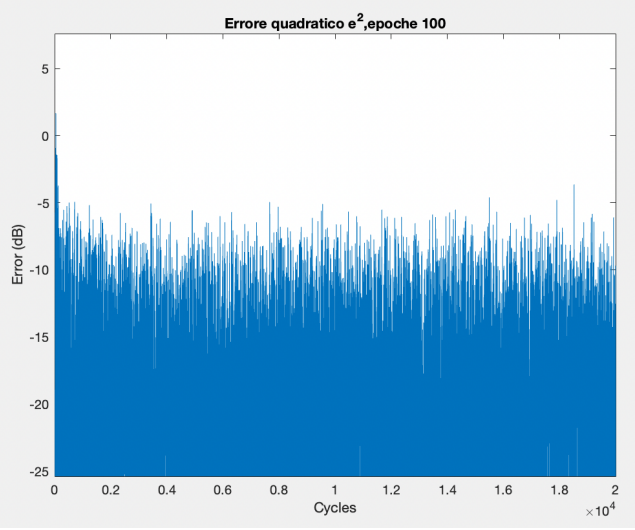


Figura 4.9: Andamento dell'errore quadratico medio con il sistema reale (diffusore Buddy-Sound v1.0)

Dopo aver costruito i blocchi lineare e non lineare del pre-distorsore, è possibile visualizzare la risposta inversa calcolata con la funzione plot di Matlab. Prima della visualizzazione è stata calcolata la trasformata di Fourier dei filtri e quindi plottato l'andamento in frequenza con scala in dB. La Figura 4.10 illustra a confronto le curve che descrivono il modello del diffusore privo di elaborazione ed il modello inverso calcolato con l'algoritmo proposto.

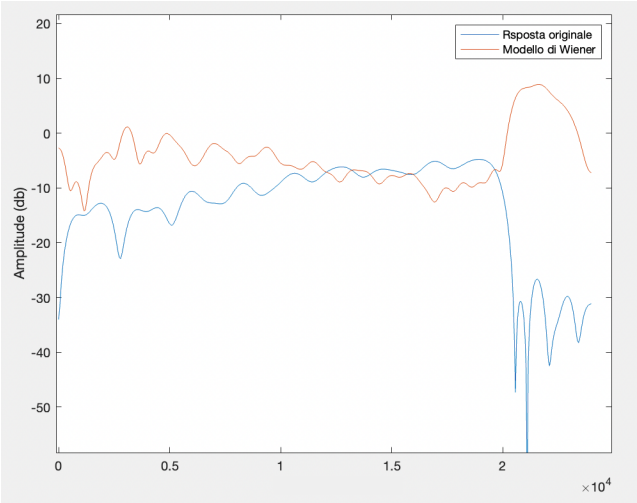


Figura 4.10: Trasformata di Fourier del modello Hammerstein e del pre-distorsore di Wiener puro inerente al sistema reale (diffusore Buddy-Sound v1.0)

Come si può notare dopo 20kHz la curva tende a valori di attenuazione molto alti, questo è dovuto al fatto che il microfono con cui è stata eseguita la misura, ha un range di funzionamento limitato in frequenza (come indicato in Figura 4.3), per cui al di sopra di alcune frequenze non riesce a registrare i segnali sonori e si crea un attenuazione del segnale. Questo particolare non deve essere trascurato in fase di calcolo del pre-distorsore, poiché una grande attenuazione come quella presente, si traduce in una elevata amplificazione e questo aspetto risulta pericoloso nel momento in cui si applica la compensazione al sistema reale generando un alto guadagno a frequenze per cui il diffusore non è progettato, comportando danni irreversibili agli elementi che lo compongono. Per evitare i problemi citati la risposta risultante del pre-distorsore deve essere filtrata con un filtro che limita la risposta nello spettro finale. Nell'immagine 4.11 è illustrata la risposta del sistema Wiener convoluito con il filtro sopra citato per la limitazione in banda. Questo è un passa-banda FIR lungo 1024 campioni e agente dello spettro 20Hz-20kHz, calcolato con la funzione matlab

"*filt*".

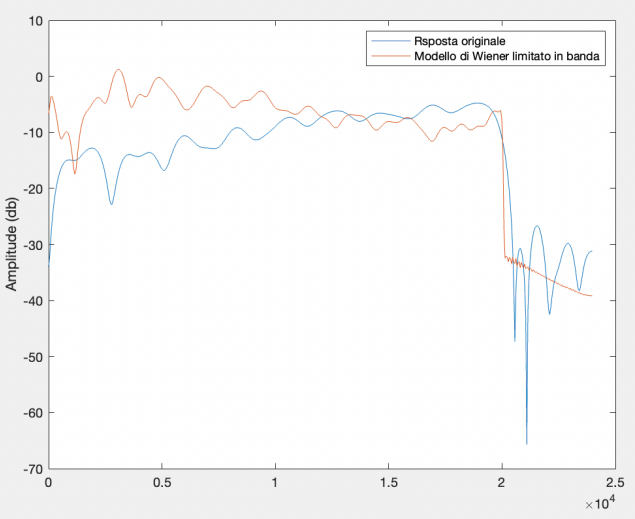


Figura 4.11: Trasformata di Fourier del modello Hammerstein e del pre-distorsore di Wiener limitato in banda inerente al sistema reale (diffusore Buddy-Sound v1.0)

Prima di passare ad i test in camera semi-anecoica, è stato utile filtrare un segnale audio musicale per il modello del diffusore valutando la distorsione presente per apprezzare acusticamente la compensazione. Quindi è stata applicata la compensazione ed eseguito il plot dei tre andamenti a confronto come mostrato in Figura 4.12. Come si può notare, la curva gialla, ottenuta filtrando l'ingresso originale con il sistema di riferimento (Buddy-Sound) ed il pre-distorsore, segue più precisamente l'andamento del segnale originale in blu. Mentre la curva rossa ottenuta filtrando l'ingresso per il modello di Hammerstein che identifica il diffusore è più discostata da quella blu. Quindi è possibile affermare che il filtro di compensazione ha attenuato la distorsione introdotta dal diffusore. Per ultimo è stata valutata la distorsione armonica totale. Questo dato permette di quantificare la distorsione introdotta degli altoparlanti valutando l'uscita prodotta da un tono puro ad una specifica frequenza dato in ingresso. Nello specifico il test è stato effettuato con un segnale sinusoidale ad 1kHz e la distorsione misurata è di circa il 3.1% per il sistema originale e pari al 2.3% nel caso del diffusore compensato. La correzione ottenuta risulta buona e prova ulteriormente il funzionamento dell'algoritmo. Nel capitolo successivo verranno illustrati ed analizzati i risultati della compensazione effettuata al diffusore Buddy-Sound v1.0 reale valutata in camera semi-anecoica.

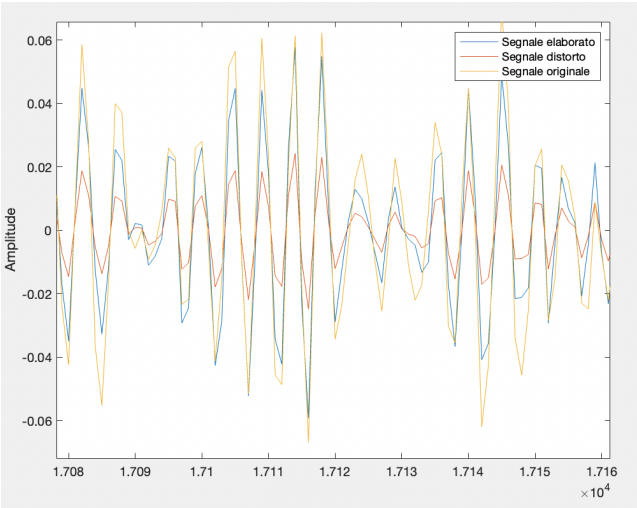


Figura 4.12: Confronto tra il segnale originale quello distorto e quello compensato per il (diffusore Buddy-Sound v1.0)

Capitolo 5

Compensazione dell'Array Circolare: test sperimentali

In questo capitolo verranno illustrate le procedure di implementazione in real-time della compensazione mediante la pre-distorsione del segnale prima di darlo in ingresso al diffusore. In primo luogo è stato realizzato un NUTS (NU-Tech Satellite) apposito nel programma "NU-Tech" per poter valutare la fedeltà dell'elaborazione in tempo reale, con il diffusore Buddy-Sound, misurando la risposta con e senza elaborazione. In secondo luogo è stato realizzato uno schema per l'implementazione del pre-distorsore nel programma "Sigma-Studio" con il fine di caricare l'elaborazione sul DSP "adau1701" integrato nella board di test.

5.1 Implementazione in NU-Tech

Terminati i test offline nella piattaforma Matlab è necessario validare i risultati ottenuti applicando la pre-distorsione al diffusore Buddy-Sound v1.0. A tale scopo è stato utilizzato il software "NU-Tech". Per effettuare il filtraggio in real-time sono necessari due NUTS: uno per il filtraggio della parte lineare di Wiener ed uno per la parte non lineare. Per la prima è possibile utilizzare il NUTS denominato "*filter*". Quest'ultimo permette di effettuare un filtraggio inserendo i parametri di frequenza di taglio, banda passante e lunghezza, oppure importando i coefficienti calcolati esternamente. Per quanto riguarda la parte non lineare è necessario realizzare un NUTS apposito che applichi al frame corrente il polinomio calcolato.

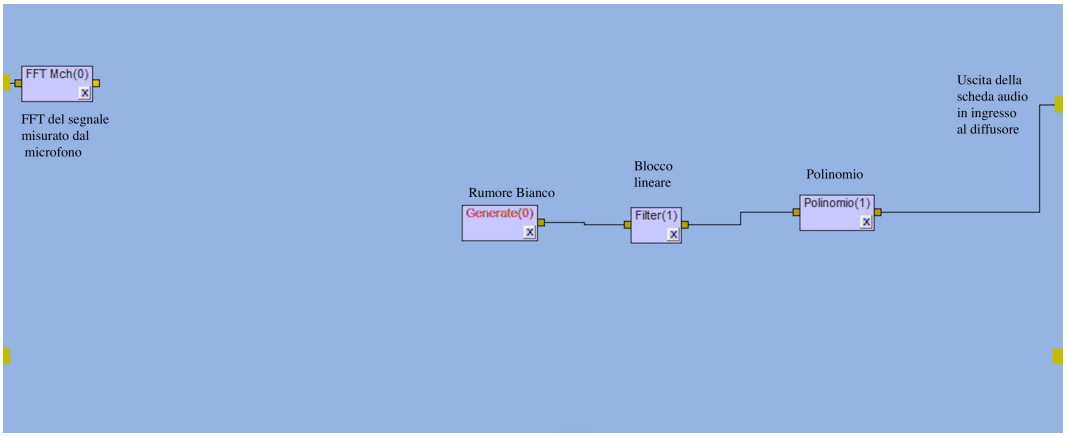


Figura 5.1: Schema NU-Tech che implementa il pre-distorsore di Wiener

In Figura 5.1 è illustrato lo schema che implementa il modello di Wiener. In particolare il flusso procede da sinistra verso destra partendo dal NUTS *Generate* che genera l'ingresso da conferire al pre-distorsore. Quindi in cascata è connesso il *filter* che esegue il filtraggio lineare ed infine il NUTS *Polinomio* che è stato realizzato per l'applicazione della funzione polinomiale. Il segnale pre-distorto viene quindi fornito in ingresso al diffusore mediante la scheda audio ed il risultato dell'elaborazione è acquisito dal microfono e visualizzato in NU-Tech grazie al NUTS *FFTMch*. Il NUTS polinomio è stato realizzato nell'applicazione "*microsoft visual studio*". Questa ide è direttamente interfacciata con il NU-Tech e consente di eseguire il debug dei NUTS realizzati direttamente sul software di Leaff Engineering. Di seguito verrà descritto il codice che implementa il polinomio. Per la realizzazione del NUTS sono necessari di base due file che sono chiamati in questo caso ***plugin.h*** e ***plugin.cpp***. Il `Codice 5.1` mostra la porzione, del ***plugin.h*** dove vengono dichiarate le variabili necessarie per i calcoli all'interno del NUTS. Partendo dall'alto sono presenti alcune variabili di tipo intero: *Framesize, SampleRate* e *N*. In real time non posso lavorare campione per campione poiché richiederebbe costi computazionali molto elevati quindi, a discapito di una maggiore latenza, occorre necessariamente lavorare con frame la quale lunghezza è decisa dalla variabile *Framesize*. *SampleRate* indica la frequenza di campionamento dei segnali che vengono elaborati ed infine *N* l'ordine del polinomio da applicare ai frame. Due variabili di tipo carattere serviranno per acquisire e salvare il path che conterrà i coefficienti moltiplicativi della funzione

polinomiale. La prima contiene il path di defaut, la seconda conterrà il percorso inserito dall'utente nella finestra di interfaccia che verrà descritta inseguito. A è una variabile ausiliaria utilizzata per contenere il coefficiente i-esimo del polinomio nelle operazioni. Per poter attivare o meno la compensazione è necessaria una condizione che se vera attiverà il bypass, e questa è contenuta nella variabile boolean nominata K. Infine sono state dichiarate diverse variabili double ad allocazione dinamica. Queste in particolare sono:

- input_buffer: che contiene il frame di ingresso filtrato per la parte lineare al NUTS polinomio;
- out_buffer: che contiene il frame di uscita del NUTS polinomio. In questo caso se il flag di bypass è attivo out_buffer coinciderà con input_buffer altrimenti conterrà il frame elaborato;
- y_nl: questo vettore contiene il risultato dell'elaborazione che verrà copiato in out_buffer per essere conferito in uscita;
- mul_in_buff: è il vettore che contiene il frame moltiplicato per il coefficiente corrente contenuto in A;
- in_pow: è il vettore che contiene i campioni che compongono il frame elevato a potenza pari all'ordine che si sta calcolando:
- coeff: è il vettore che contiene i coefficienti importati dal file contenuto nel path citato precedentemente.

```
1  private:
2    Ipp64f NUMBIT = 0;
3    int FrameSize,SampleRate;
4    int N;
5    double *coeff;
6    char  save_name_dir[MAX_FILE_NAME_LENGTH];
7    char  save_name_default[ MAX_FILE_NAME_LENGTH];
8    double A;
9    bool K;
10   double*input_buffer,*out_buffer,*y_nl,*mul_in_buff,*Y_out,*in_pow;
```

Listing 5.1: Dichiarazione delle variabili del NUTS

Dopo aver dichiarato tutte le variabili in ***plugin.h*** è possibile passare al ***plugin.cpp***. Per fare un nuovo NUTS è stato utilizzato un template vuoto contenente le sezioni necessarie per la realizzazione. Si parte dall'inizializzazione dei valori delle variabili sopra elencate nella sezione "LEEffect" mostrata nel `Codice` 5.2.

```
PlugIn::PlugIn(InterfaceType _CBFunction,void * _PlugRef,HWND
    ParentDlg): LEEffect(_CBFunction,_PlugRef,ParentDlg)
{
  FrameSize = CBFunction(this,NUTS_GET_FS_SR,0,(LPVOID)AUDIOPROC);
  SampleRate = CBFunction(this,NUTS_GET_FS_SR,1,(LPVOID)AUDIOPROC);
  input_buffer = 0;

  in_pow = 0;
  coeff = 0;
  out_buffer = 0;
  mul_in_buff = 0;
  y_nl = 0;
  N = 3;
  A = 0;
  int dtype = CBFunction(this, NUTS_GETDRIVERTYPE, 0, 0);
  if (dtype == 1)
    NUMBIT = pow(2.0, 31.0);
  else
    NUMBIT = pow(2.0, 15.0);

  memset(save_name_dir, 0, MAX_FILE_NAME_LENGTH * sizeof(char));
  strcpy(save_name_dir, "C:\\Users\\MICH\\Desktop\\POLINOMIO_PROVA\\
    tetaF.dat");

}
```

Listing 5.2: Inizializzazione di tutte le variabili del NUTS

In questa fase i vettori e le variabili sono inizializzate ad un valore fittizio, non sono ancora allocate in memoria. Inoltre è dichiarata una funzione di normalizzazione che verrà utilizzata in seguito. Quindi si passa alla dichiarazione del costruttore nominata "*init*" mostrata nel `Codice` 5.3.

```
void __stdcall PlugIn::LEPlugin_Init()
{
  if (input_buffer == 0)
  {
    input_buffer = ippsMalloc_64f(FrameSize);
    ippsZero_64f(input_buffer, FrameSize);
  }
  if (out_buffer == 0)
  {
    out_buffer = ippsMalloc_64f(FrameSize);
    ippsZero_64f(out_buffer, FrameSize);
  }
  if (y_nl == 0)
  {
    y_nl = ippsMalloc_64f(FrameSize);
    ippsZero_64f(y_nl, FrameSize);
  }
  if (coeff == 0)
  {
    coeff = ippsMalloc_64f(N);
    ippsZero_64f(coeff, N);
  }
  if (mul_in_buff == 0)
  {
    mul_in_buff = ippsMalloc_64f(FrameSize);
    ippsZero_64f(mul_in_buff, FrameSize);
  }
  if (in_pow == 0)
  {
    in_pow = ippsMalloc_64f(FrameSize);
    ippsZero_64f(in_pow, FrameSize);
  }
  read_dat(save_name_dir, coeff, N);
}
```

Listing 5.3: Codice del costruttore per il nuovo NUTS

Questa viene chiamata ogni volta che si trascina il NUTS dall'elenco dei plugin al workspace. Nella *init* è eseguita l'allocazione dei vettori precedentemente descritti

e necessita di particolare attenzione. Infatti quando si stabilisce una lunghezza del vettore bisogna considerare la lunghezza iniziale e la lunghezza massima che si raggiunge nel momento delle elaborazioni che avverranno nella sezione *process*. Allocare un vettore in modo errato può verificare diversi problemi come l'eccezione. Questa anomalia risulta tra l'altro pericolosa in quanto la casella successiva all'ultima allocata contiene l'indirizzo di memoria del vettore stesso e sovrascrivere quell'elemento implica la perdita della posizione di allocazione e quindi rende impossibile de-allocare la variabile in questione. Il ripetersi di questa anomalia potrebbe inoltre generare l'esaurimento della memoria che, per essere liberata, richiede necessariamente il riavvio del PC. Oltre all'allocazione dei vettori, nella *init* è effettuata la procedura di copia degli elementi presenti nel file specificato dal path sopra descritto nella variabile ***coeff***. Dopo il costruttore è stato dichiarato il distruttore che in questo caso prende il nome di "*delete*". La porzione `Codice 5.4` mostra la realizzazione della "*delete*" del NUTS.

```
1  void __stdcall PlugIn::LEPlugin_Delete()
2  {
3    if (input_buffer != 0) {
4      ippsFree(input_buffer);
5      input_buffer = 0;
6    }
7    if (coeff != 0) {
8      ippsFree(coeff);
9      coeff = 0;
10   }
11   if (out_buffer != 0) {
12     ippsFree(out_buffer);
13     out_buffer = 0;
14   }
15   if (y_nl != 0) {
16     ippsFree(y_nl);
17     y_nl = 0;
18   }
19   if (mul_in_buff != 0) {
20     ippsFree(mul_in_buff);
21     mul_in_buff = 0;
22   }
```

```
23   if (in_pow != 0) {
24     ippsFree(in_pow);
25     in_pow = 0;
26   }
27 }
```

Listing 5.4: Codice del distruttore per il nuovo NUTS

In questa parte di codice sono de-allocati i vettori presenti nella *init*. Anche in questo caso occorre attenzione poiché la de-allocazione di variabili non vettorizzate, genera errore nella procedura di funzionamento quando si elimina il NUTS dal workspace, ma non determina errori nella fase di compilazione. Dopo aver terminato le procedure di inizializzazione, allocazione e de-allocazione delle variabili è possibile passare alla sezione ***process***. Questa contiene l'elaborazione che avviene in loop quando è premuto il pulsante di avvio nel NU-Tech. Il `Codice 5.5` riportato di seguito effettua il filtraggio del frame del segnale di ingresso per il polinomio desiderato.

```
1  int __stdcall PlugIn::LEPlugin_Process(PinType **Input,PinType **
       Output,LPVOID ExtraInfo)
2  {
3    ippsCopy_64f((double*)(*Input[0]).DataBuffer, input_buffer,
       FrameSize);
4
5    if (K == false) {
6      ippsMulC_64f_I(1.0 / NUMBIT, input_buffer, FrameSize);
7
8      for (int i = 0; i < N; i++) {
9
10       A = coeff[i];
11       ippsPowx_64f_A26(input_buffer, i + 1, in_pow, FrameSize);
12
13       ippsMulC_64f(in_pow, A, mul_in_buff, FrameSize);
14
15       ippsAdd_64f_I(mul_in_buff, y_nl, FrameSize);
16     }
17     ippsCopy_64f(y_nl, out_buffer, FrameSize);
18     ippsZero_64f(y_nl, FrameSize);
19     ippsMulC_64f_I(-NUMBIT, out_buffer, FrameSize);
```

```
20      ippsCopy_64f(out_buffer, (double*)(*Output[0]).DataBuffer,
        FrameSize);
21    }
22    else {
23      ippsCopy_64f(input_buffer, (double*)(*Output[0]).DataBuffer,
        FrameSize);
24    }
25    return COMPLETED;
26  }
```

Listing 5.5: Codice del distruttore per il nuovo NUTS

Il primo passo è quello di copiare nell'input_buffer il frame ricevuto in ingresso mediante la funzione *ippsCopy*, che effettua la copia di *Framesize* valori dal DataBuffer all'input_buffer che sarà utilizzato per i calcoli successivi. Quindi è inserita la condizione di controllo sul flag del bypass contenuto in **K**: nel caso in cui il bypass risulti ***true*** mediante *ippsCopy* setto l'uscita del NUTS uguale all'ingresso. Se il bypass è nella condizione "*false*", viene effettuata la normalizzazione del segnale in ingresso cosi da evitare grandi numeri in fase di calcolo e si procede con l'elaborazione. Il secondo passo è quello di applicare al frame corrente la non linearità data dal polinomio del pre-distorsore. Ciò è fatto mediante un ciclo iterativo che viene ripetuto un numero di volte pari all'ordine del polinomio stesso ***N***. La variabile ***A*** è caricata con la costante del coefficiente polinomiale dell'ordine i-esimo, quindi viene effettuato l'elevamento a potenza del frame con l'indice "i+1". Questo si deve al fatto che l'indice "*i*" va da 0 ad N-1 mentre l'elevazione a potenza in questo caso deve essere fatta da 1 ad N. La funzione utilizzata per il calcolo è *ippsPowx*. Questa effettua l'elevazione a potenza di tutti gli elementi del vettore di ingresso e li salva in *in_pow*. Una volta ottenuto il segnale *in_pow* è possibile effettuare la moltiplicazione di ogni elemento del frame corrente per la costante ***A***. L'operazione è fatta grazie alla funzione *ippsMulC*, che effettua la moltiplicazione di tutti gli elementi del vettore di ingresso (*in_pow*) per una costante(***A***) salvando il risultato nel vettore *mul_in_buff*. Dopo aver moltiplicato il frame per ***A*** è possibile calcolare la funzione polinomiale avendo tutti i termini noti. Si esegue l'addizione dei termini del polinomio con la funzione *ippsAdd_I*. Quest'ultima rispetto alle altre è fatta in-place poiché ad ogni iterazione devo sommare il valore riferito all'ordine corrente a quelli precedenti. Il

terzo passo riguarda il settaggio dell'uscita del NUTS. In primo luogo viene copiata l'uscita dell'elaborazione nell' *out_buffer*, quindi il vettore *y_nl* viene azzerato per accogliere l'elaborazione del frame successivo. È importante che *y_nl* sia un vettore di zeri quando inizia un nuovo ciclo di elaborazione sul frame, altrimenti all'iterazione successiva, i campioni calcolati verranno sommati a quelli del frame precedente. Per concludere è effettuata la de-normalizzazione del frame e settata l'uscita del NUTS copiando il contenuto di *out_buffer* in *DataBuffer Output*.

Per rendere possibile l'utilizzo del NUTS con diversi polinomi è stato deciso di realizzare un'interfaccia utente che permetta di variare i parametri di elaborazione senza dover accedere al codice sorgente. Tale procedura è realizzata nella sezione LERTwatch di interfaccia utente del NUTS. Il codice per la realizzazione dell'interfaccia utente è riportato di seguito:

```
1  void __stdcall PlugIn::LERTWatchInit()
2  {
3    WatchType NewWatch;
4
5    memset(&NewWatch, 0, sizeof(WatchType));
6    NewWatch.EnableWrite = true;
7    NewWatch.LenByte = sizeof(double);
8    NewWatch.TypeVar = WTC_DOUBLE;
9    NewWatch.IDVar = ORDER;
10   sprintf(NewWatch.VarName, "N_order");
11   CBFunction(this, NUTS_ADDRTWATCH, 0, (LPVOID)&NewWatch);
12
13   WatchType NewWatch1;
14
15   memset(&NewWatch1, 0, sizeof(WatchType));
16   NewWatch1.EnableWrite = true;
17   NewWatch1.LenByte = 256 * sizeof(char);
18   NewWatch1.TypeVar = WTC_LPCHAR;
19   NewWatch1.IDVar = COEFF;
20   sprintf_s(NewWatch1.VarName, MAXCARDEBUGPLUGIN, "Coefficienti del
       polinomio");
21   CBFunction(this, NUTS_ADDRTWATCH, TRUE, &NewWatch1);
22
23   WatchType NewWatch2;
24
```

```
25    memset(&NewWatch2, 0, sizeof(WatchType));
26    NewWatch2.EnableWrite = true;
27    NewWatch2.LenByte = sizeof(bool);
28    NewWatch2.TypeVar = WTC_BOOLEAN;
29    NewWatch2.IDVar = BYPASS;
30    sprintf_s(NewWatch2.VarName, MAXCARDEBUGPLUGIN, "Bypass");
31    CBFunction(this, NUTS_ADDRTWATCH, TRUE, &NewWatch2);
32  }
```

Listing 5.6: Codice per la realizzazione dell'interfaccia utente per la modifica dei parametri del polinomio.

Sono stati inseriti tre parametri modificabili all'interno dell'interfaccia utente:

- N_order: è un intero e serve a specificare l'ordine del polinomio che verrà applicato al segnale da elaborare. In questo caso è inizializzato a tre;
- Coefficienti del polinomio: variabile **char** e deve contenere il path del file che contiene i coefficienti moltiplicativi del polinomio. È bene che il numero di coefficienti contenuti nel file caricato sia N_order. In caso contrario comunque non sono generati errori, semplicemente i coefficienti oltre N_order non verranno considerati, o se minore saranno settati a 0;
- Bypass: è presente un flagh che permette di bypassare o meno l'elaborazione presente nella "process".

Rispetto ai primi due, il bypass può essere attivato e disattivato in qualsiasi momento anche in fase di *Run*, mentre N_order e Coefficienti del polinomio, hanno validità solo se cambiati in condizione di *Stop*. Il `Codice 5.6` indica la dichiarazione e la visualizzazione delle variabili dell'interfaccia utente, tuttavia, per far avvenire effettivamente qualcosa all'interno dell'elaborazione, vanno specificati i parametri di input ed output, realizzando le sezioni ***LEGetParameter*** e ***LESetParameter***. Queste sono mostrate nel `Codice 5.7`.

```
////////////////////////////LEGetParameter//////////////////////////

int  __stdcall PlugIn::LEGetParameter(int Index,void *Data)
{
  if (Index == ORDER)
  {
    *((double*)Data) = N;
  }
  if (Index == BYPASS)
  {
    *((bool*)Data) = K;
  }
  if (Index == COEFF) {
    strcpy((char*)Data, save_name_dir);
  }
  return 0;
}

////////////////////////////LESetParameter//////////////////////////

void __stdcall PlugIn::LESetParameter(int Index,void *Data,LPVOID
    bBroadCastInfo)
{
  if (Index == ORDER) {
    {
      N = *((double*)Data);
      CBFunction(this, NUTS_UPDATERTWATCH, ORDER, 0);
    }
  }
  if (Index == COEFF) {
    strcpy(save_name_dir, (char*)Data);
    CBFunction(this, NUTS_UPDATERTWATCH, COEFF, 0);
  }
  if (Index == BYPASS) {
    K = *((bool*)Data);
    CBFunction(this, NUTS_UPDATERTWATCH, BYPASS, 0);
  }
}
```

Listing 5.7: GetParameter e SetParameters per l'interfaccia utente del NUTS

Dopo aver realizzato il NUTS "polinomio" è stato eseguito il filtraggio in real-time per la pre-distorsione del diffusore. Con lo schema illustrato in Figura 5.1 è stato possibile confrontare le risposte del diffusore originale con quelle dopo la compensazione. I test sono stati effettuati con due condizioni di funzionamento: il primo con una sezione dell'array in funzione, ed il secondo con due sezioni in funzione, cosi da valutare le prestazioni dell'algoritmo in presenza di distorsioni più o meno significative. In Figura 5.2 è illustrata la risposta del diffusore nella condizione uno senza correzione. L'andamento in questo caso risulta poco distorto, comunque non a spettro piatto.

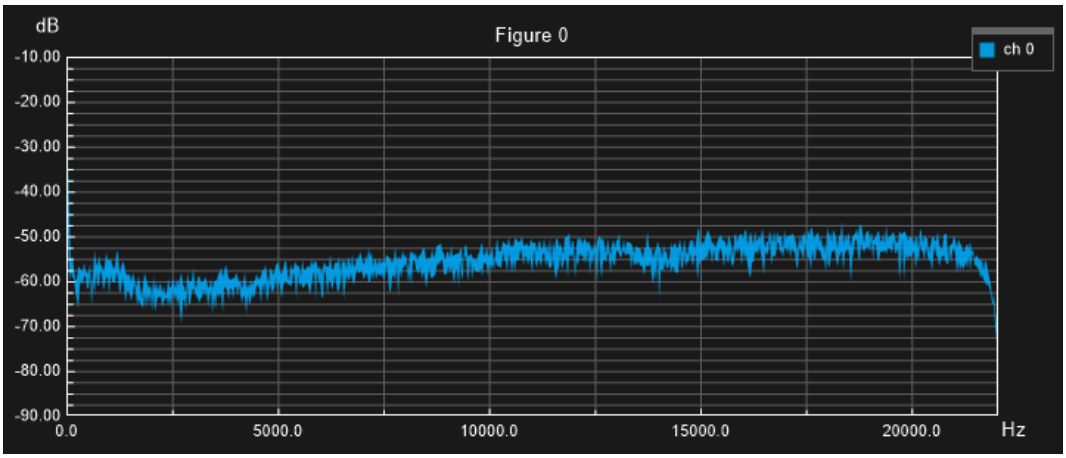


Figura 5.2: Risposta del diffusore senza elaborazione

In Figura 5.3 è illustrato l'andamento in frequenza del diffusore dopo la compensazione. Come si può notare la risposta risulta piatta nello spettro 20Hz-20kHz. Il troncamento a 20kHz è dovuto al filtraggio per la limitazione in banda dopo 20kHz, necessaria per eliminare l'anomalia nell'inversione causata dal microfono di acquisizione descritta precedentemente nel Capitolo 3.

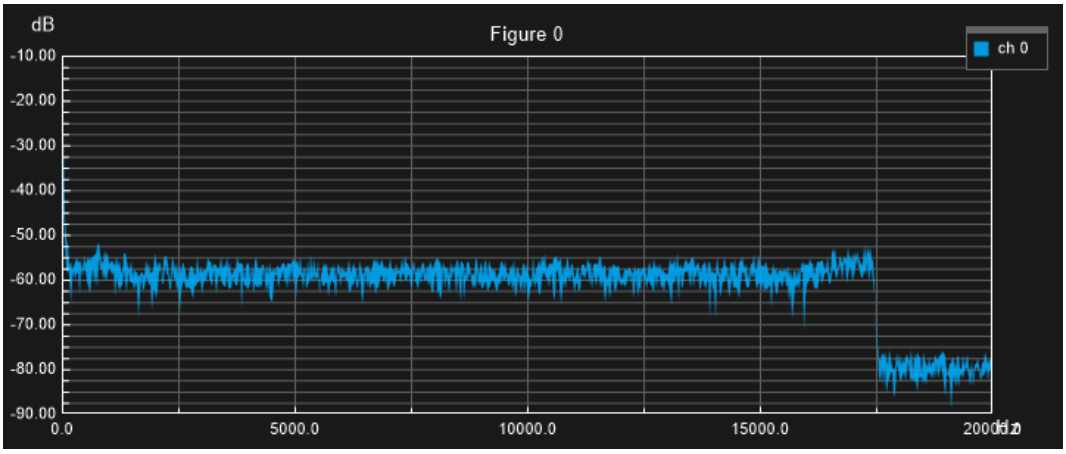


Figura 5.3: Risposta del diffusore con pre-distorsione per la compensazione

Il secondo test è stato effettuato con due sezioni in funzione, ed evidenzia la presenza di una distorsione maggiore dovuta presumibilmente ad un maggior contributo introdotto dalle vibrazioni delle pareti del diffusore causate dal funzionamento di più altoparlanti. In Figura 5.4, è riportata la risposta del diffusore prima della compensazione. Osservando l'andamento si può cogliere la presenza di una distorsione armonica più influente che nel caso precedente non è presente.

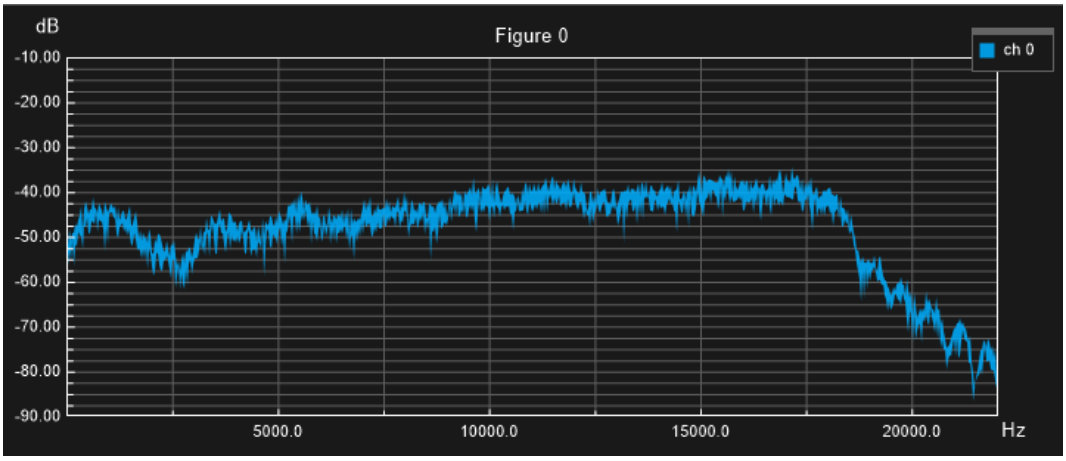


Figura 5.4: Risposta del diffusore senza elaborazione con due sezioni in funzione

Invece in Figura 5.5 è illustrato l'andamento della risposta del diffusore dopo la compensazione con il modello di Wiener.

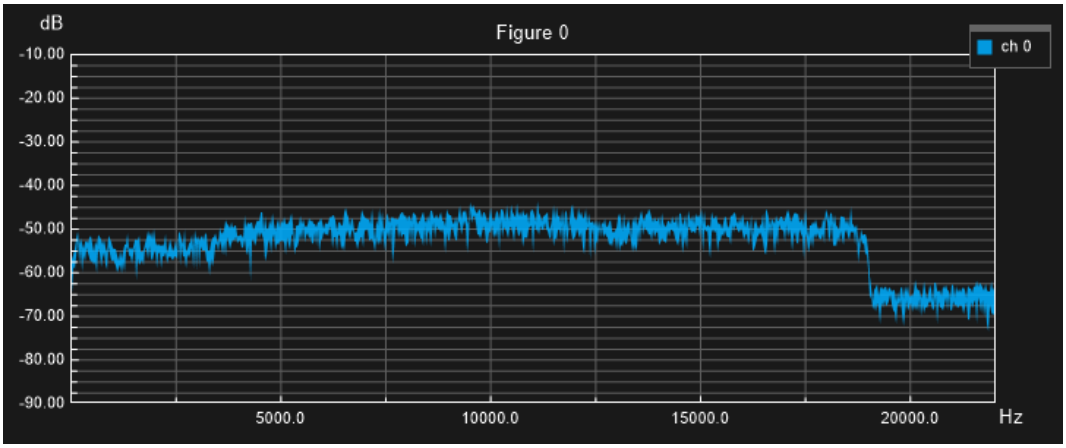


Figura 5.5: Risposta del diffusore con elaborazione con due sezioni in funzione

Come si nota anche in questo caso la correzione genera uno spettro più piatto rispetto a quello originale. Tutti i test sono stati effettuati con rumore bianco in ingresso. Dopo aver verificato il funzionamento della correzione sono stati effettuati test di ascolto con tracce musicali. In tutti i casi la compensazione ha portato miglioramenti nell'ascolto. Ovviamente questo aspetto è del tutto soggettivo, infatti spesso una risposta con spettro piatto non significa necessariamente che sia preferibile per l'ascolto, tuttavia i risultati raggiunti nei test sopra descritti sono la prova del funzionamento dell'algoritmo. Successivamente per rendere gradevole l'ascolto è possibile effettuare un'equalizzazione che modifichi la risposta a discrezione dell'ascoltatore.

5.2 Filtraggio in real-time su DSP

Nella sezione precedente è stata descritta la procedura di filtraggio in real-time con il software NU-Tech. Per poter applicare l'elaborazione tuttavia, è necessario utilizzare una scheda audio per la gestione di input ed output e per l'elaborazione dei frame in NU-Tech. In genere per un utilizzo utente non si dispone di tali risorse. Per tale motivo è necessario elaborare una soluzione che permetta di integrare la correzione senza dover utilizzare dispositivi esterni al diffusore. Disponendo di un componente per il digital signal processing a bordo degli amplificatori è stato deciso di implementare il filtraggio di compensazione direttamente sul diffusore. Il DSP montato è l'*ADAU1701* prodotto da *analog device*, uno dei più comuni per

l'utilizzo nel processing audio professionale. Per la programmazione dei DSP di questa categoria Analog device mette a disposizione il software *SigmaStudio*. Quest'ultimo permette di implementare algoritmi DSP con una semplice programmazione ad oggetti tramite plugin che consentono di eseguire numerose funzioni base di DSP. Il dispositivo ADAU1701 è uno dei più basilari della famiglia ADAU, quindi per realizzare la struttura di pre-compensazione desiderata occorre prestare attenzione per non superare le capacità computazionali del dispositivo. In Figura 5.6 è illustrato lo schema che implementa la struttura di Wiener in SigmaStudio.

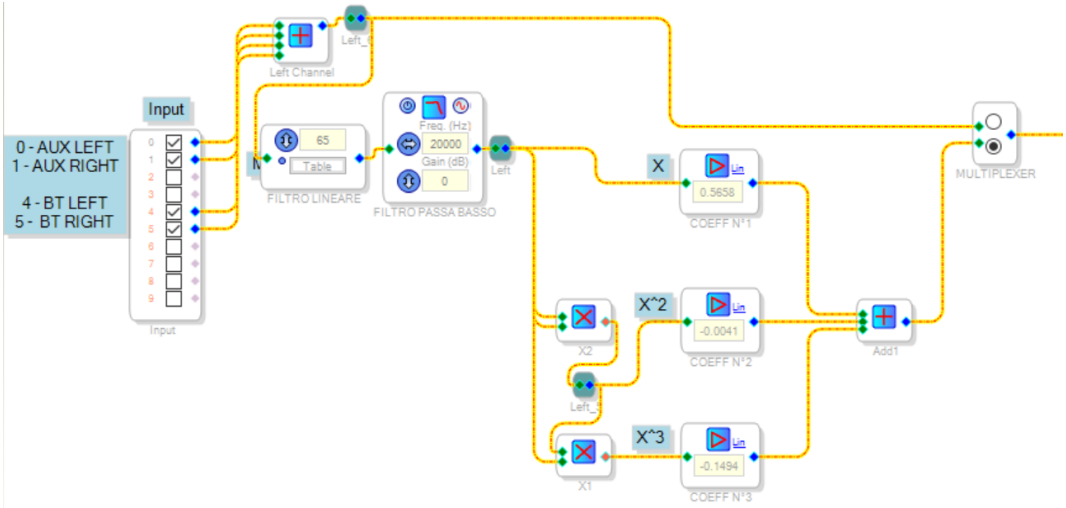


Figura 5.6: Topologia per filtraggio in SigmaStudio

Seguendo il flusso da sinistra verso destra si può notare in primo luogo il blocco che contiene gli input del dispositivo. In questo caso, i primi due, rispettivamente **input 0** e **input 1**, sono ingressi analogici forniti mediante ADC, mentre **input 4** e **input 5** sono ingressi digitali collegati ad un modulo bluetooth integrato. Poiché il diffusore non funziona in versione stereofonica tutti gli ingressi sono sommati in un mixer. L'uscita del mixer è splittata in due per consentire, con l'utilizzo di un multiplexer, di selezionare l'ingresso elaborato o quello originale. Per quanto riguarda il ramo di elaborazione si procede in primo luogo effettuando il filtraggio della parte lineare. A tale scopo è stato utilizzato il plugin *Fir* presente nella sottosezione *basic dsp*. *Fir* permette di effettuare un filtraggio di tipo FIR importando un numero limitato di coefficienti. Nella guida utente il massimo numero di tappi supportato è 800. Per integrare il filtraggio in SigmaStudio è stato inizialmente accorciato il filtro passa

banda, passando da uno a 1024 campioni, utilizzato per le simulazioni in matlab e NU-Tech, ad uno di 735, così da risultare, dopo la convoluzione con il FIR lungo 65 calcolato con l'algoritmo, lungo 800 tappi. Tuttavia in fase di caricamento si è verificata una eccezione della capacità di calcolo del DSP. A causa di ciò è stato deciso di eseguire il filtraggio di limitazione in banda fuori dal plugin *fir*. Questo è stato possibile con l'utilizzo del blocco LowPassFilter presente nella libresia che effettua un filtraggio di tipo IIR. Successivamente seguendo il flusso è implementato il polinomio. A tale scopo sono stati utilizzati i blocchi di moltiplicazione di segnali e guadagno costante presenti nella sezione aritmetics function. Dopo aver calcolato i coefficienti del polinomio è stata eseguita la somma dei tre segnali calcolati con il plugin *Add*. Quest'ultimo è collegato al multiplexer la cui uscita è data in ingresso alla porzione che effettua l'equalizzazione utilizzata per rendere l'ascolto più piacevole. Quest'ultima non sarà illustrata poiché non è inerente allo studio effettuato per questo lavoro di tesi. Grazie all'integrazione dell'elaborazione sul DSP è possibile riprodurre tracce musicali o qualsivoglia segnale audio mediante qualsiasi dispositivo munito di bluetooth o di uscita analogica come cellulari, PC, microfoni dinamici e altri dispositivi di questo genere.

Capitolo 6

Conclusioni

In questo lavoro di tesi è stato proposto un metodo di compensazione per sistemi non lineari. In particolare, è stato utilizzato un diffusore composto da un array di altoparlanti circolare. In primo luogo, per poter quantificare il comportamento lineare e non lineare, sono state analizzate diverse tecniche di caratterizzazione presenti in letteratura. Si è posta l'attenzione sullo sviluppo in serie di Volterra, sul modello di Hammerstein e sul modello di Wiener. Per rendere l'algoritmo meno oneroso dal punto di vista computazionale è stato adottato il modello di Hammerstein full-band proposto in [33]. Dopo aver costruito il modello del diffusore, è stato elaborato il modello del pre-distorsore per eseguire la compensazione lineare e non lineare. Il sistema non lineare è stato caratterizzato con un modello di tipo Hammerstein e per la compensazione è stata applicata una struttura di Wiener. Nello specifico, il modello inizialmente considerato è composto da un polinomio di ordine pari a tre ed un filtro lineare di tipo IIR. In seguito, è stato scelto di trasformare la parte lineare come filtro FIR per aggiungere il vantaggio della stabilità e della risposta in fase lineare a discapito di un numero maggiore di coefficienti. Una volta calcolato il pre-distorsore è stato possibile eseguire alcuni test per verificare l'elaborazione, confrontando l'uscita del sistema con e senza compensazione. Per fare ciò e stato necessario realizzare un blocchetto di filtraggio custom nel software NU-Tech per l'applicazione del polinomio. Infine, per rendere il sistema più adatto ad un utilizzo utente, è stato integrato il filtraggio per il pre-distorsore all'interno del DSP adau1701 montato sulle board che gestiscono gli altoparlanti dell'array circolare. Oltre le misure, che hanno evidenziato un netto miglioramento della risposta, anche l'ascolto risulta più gradevole in presenza della correzione. Per il futuro, gli sviluppi da fare riguardano l'implementazione

real-time dell'adattamento dell'algoritmo, che in questo lavoro di tesi è stato fatto in matlab lasciando solo il filtraggio all'integrazione su DSP. Poter eseguire una procedura di adattamento real-time ad anello chiuso è necessario un dispositivo per il digital signal processing più prestazionale in grado di eseguire operazioni complesse con poca latenza. Con lo scopo di proporre un metodo di compensazione, il codice sviluppato lavora campione per campione e richiede capacità computazionali incompatibili con l'utilizzo in real-time, perciò è necessario ottimizzare l'algoritmo in modo da lavorare con frame di segnale. Un altro sviluppo riguarda il diffusore stesso e consiste nel migliorare la stabilità del box che contiene gli altoparlanti, così da attenuare la distorsione introdotta dalla vibrazione delle pareti del diffusore. Infine, potrebbe essere utile, per migliorare le prestazioni dell'algoritmo, utilizzare un modello più complesso in grado di identificare in modo migliore le caratteristiche degli altoparlanti, con l'obiettivo però di mantenere il sistema più efficiente possibile. Nello specifico si parla di strutture del tipo Hammerstein generalizzato. La serie di Volterra rimane ancora troppo complessa per l'applicazione in real-time. Per concludere, si può affermare che il lavoro di tesi ha portato ad esito positivo poiché l'algoritmo proposto riesce ad eseguire la compensazione del diffusore composto da un array di altoparlanti circolare con buone prestazioni.

Bibliografia

[1] Martin Schetzen. The volterra and wiener theories of nonlinear systems. 2006.

[2] Tokunbo Ogunfunmi. Adaptive nonlinear system identification. The Volterra and Wiener model approaches. 01 2007.

[3] A.Y. Kibangou and G. Favier. Wiener-hammerstein systems modeling using diagonal volterra kernels coecients. IEEE Signal Processing Letters, 13(6):381–384, 2006.

[4] Esref Eskinat, Stanley H. Johnson, and William L. Luyben. Use of hammerstein models in identification of nonlinear systems. AIChE Journal, 37(2):255–268, 1991

[5] V .V olterra, “Theory Of Functionals and of Integral and Integro-Differential Equations”, Dover, New-York, 1958

[6] N.Wiener, “Non Linear Problems In Random Theory”, MIT Press, Cambridge, MA, 1958

[7] A.M.Kaizer, “Modeling of The Nonlinear Response of an Electrodynamic Loudspeaker by a Volterra Series Expansion”, Journal of the Audio Engineering Society, Vol 35, pp 421-433, 1987 June

[8] Martin J.Reed and Malcolm O.J. Hawksford, “Comparison of Audio System Nonlinear Performance in V olterra Space”, Audio Engineering Society 103rd Convention, September 1997, Preprint 4606

[9] A.Y. Kibangou and G. Favier. Wiener-hammerstein systems modeling using diagonal volterra kernels coecients. IEEE Signal Processing Letters, 13(6):381–384, 2006.

[10] B. Defraene, T. van Waterschoot, M. Diehl and M. Moonen, "Embedded-Optimization-Based Loudspeaker Precompensation Using a Hammerstein Loudspeaker Model," in IEEE/ACM Transactions on Audio, Speech, and Language Processing, vol. 22, no. 11, pp. 1648-1659, Nov. 2014, doi: 10.1109/TASLP.2014.2344862.

[11] Antonin Novak, Laurent Simon, Pierrick Lotton. Synchronized Swept-Sine: Theory, Applica- tion, and Implementation. Journal of the Audio Engineering Society, 2015, 63 (10), pp.786-798. 10.17743/jaes.2015.0071 . hal-02504321

[12] Biagini G. (2003). Identificazione di sistemi non-lineari del tipo di Hammerstein e di Wiener. Tesi di laurea presso l'Università degli Studi di Pisa.

[13] Boutayeb M.; Darouach M. (1995). Recursive identification method for MISO Wiener-Hammerstein model. IEEE Transactions on Automatic Control, 40, 287-291.

[14] Simone Orcioni, Alessandro Terenzi, Stefania Cecchi, Francesco Piazza, and Alberto Carini. Identification of volterra models of tube audio devices using multiple-variance method. Journal of the Audio Engineering Society, 66(10):823–838, 2018.

[15] Y. W. Lee and M. Schetzen. Measurement of the wiener kernels of a non-linear system by cross-correlation. International Journal of Control, 2(3):237–254, 1965.

[16] Orcioni, S., Pirani, M.& Turchetti, C. Advances in Lee–Schetzen Method for Volterra Filter Identification. Multidim Syst Sign Process 16, 265–284 (2005). https://doi.org/10.1007/s11045-004-1677-7

[17] Angelo Farina. Simultaneous measurement of impulse response and distortion with a swept-sine technique. 11 2000.

[18] Angelo Farina, Alberto Bellini, and Enrico Armelloni. Non-linear con- volution: A new approach for the auralization of distorting systems. 05 2001.

[19] J. O. Jungmann, R. Mazur, M. Kallinger, T. Mei, and A. Mertins, "Combined acoustic mimo channel crosstalk cancellation and room impulse response reshaping," Audio, Speech, and Language Processing, IEEE Transac- tions on, vol. 20, no. 6, pp. 1829–1842, 2012.

[20] D. G. C iric, M. Markovic, M. Mijic, and D. S˘umarac-Pavlovic, "On the effects of nonlinearities in room impulse response measurements with exponential sweeps," Applied Acoustics, vol. 74, no. 3, pp. 375–382, 2013.

[21] P. Majdak, P. Balazs, and B. Laback, "Multiple exponential sweep method for fast measurement of head-related transfer functions," Journal of the Audio Engineering Society, vol. 55, no. 7/8, pp. 623–637, 2007.

[22] P. Dietrich, B. Masiero, and M. Vorlander, "On the optimization of the multiple exponential sweep method," Journal of the Audio Engineering Society, vol. 61, no. 3, pp. 113–124, 2013.

[23] A. Novak, M. Bentahar, V. Tournat, R. El Guerjouma, and L. Simon, "Nonlinear acoustic characterization of micro-damaged materials through higher harmonic resonance analysis," NDT & E International, vol. 45, no. 1, pp. 1–8, 2012.

[24] J. Legland, V. Tournat, O. Dazel, A. Novak, and V. Gusev, “Linear and nonlinear biot waves in a noncohe- sive granular medium slab: Transfer function, self-action, second harmonic generation,” The Journal of the Acoustical Society of America, vol. 131, no. 6, pp. 4292–4303, 2012.

[25] A. Novak, L. Simon, P. Lotton, and F. Kadlec, “A new method for identification of nonlinear systems using miso model with swept-sine technique: Application to loudspeaker analysis,” in Audio Engineering Society Convention 124, Audio Engineering Society, May 2008.

[26] A. Novak, L. Simon, F. Kadlec, and P. Lotton, “Nonlinear system identification using exponential swept-sine signal,” Instrumentation and Measurement, IEEE Transactions on, vol. 59, no. 8, pp. 2220–2229, 2010.

[27] M. Rebillat, R. Hennequin, È. Corteel, and B. F. Katz, “Identification of cascade of hammerstein models for the description of nonlinearities in vibrating devices,” Journal of Sound and Vibration, vol. 330, no. 5, pp. 1018–1038, 2011.

[28] Wolfgang Klippel. The mirror filter-a new basis for reducing nonlinear distortion and equalizing response in woofer systems. J.9 Audio Eng. Soc, 40(9):675–691, 1992.

[29] L. Gan and E. Abd-Elrady, "Adaptive predistortion of IIR Hammerstein systems using the Nonlinear Filtered-x LMS algorithm," 2008 6th International Symposium on Communication Systems, Networks and Digital Signal Processing, Graz, Austria, 2008, pp. 702-705, doi: 10.1109/CSNDSP.2008.4610765.

[30] Y. H. Lim, Y. S. Cho, I. W. Cha, and D. H. Youn, “An adaptive nonlinear prefilter for compensation of distortion in nonlinear systems,” IEEE Tran. on Sig. Proc., vol. 46, no. 6, 1998.

[31] C. R. Johnson Jr., “Adaptive IIR filtering: Current results and open issues,” IEEE Trans. Inform. Theory, vol. IT-30, no. 2, pp. 237–250, 1984.

[32] D. Y. Zhou and V. E. DeBrunner, “Novel adaptive nonlinear predistorters based on the direct learning algorithm,” IEEE Trans. on Signal Processing, vol. 55, no. 1, 2007.

[33] A. Terenzi, V. Bruschi, S. Cornell, A. Castellani and S. Cecchi, "A Multiband Structure based on Hammerstein Model for Nonlinear Audio System Identification," 2019 11th International Symposium on Image and Signal Processing and Analysis (ISPA), Dubrovnik, Croatia, 2019, pp. 9-14, doi: 10.1109/ISPA.2019.8868713.

[34] A. Novak, L. Simon, F. Kadlec and P. Lotton, "Nonlinear System Identification Using Exponential Swept-Sine Signal," in IEEE Transactions on Instru-

mentation and Measurement, vol. 59, no. 8, pp. 2220-2229, Aug. 2010, doi: 10.1109/TIM.2009.2031836.

[35] Andrea Primavera, Michele Gasparini, Stefania Cecchi, Wataru Hariya, Shogo Murai, Koji Oishi, and Francesco Piazza. A novel measurement procedure for wiener/hammerstein classification of nonlinear audio systems. In Audio Engineering Society Convention 144, May 2018.

[36] Brunet, Pascal; Temme, Steve; 2006; A New Method for Measuring Distortion using a Multitone Stimulus and Non-Coherence [PDF]; Listen, Inc; Paper 6877; Available from: https://aes2.org/publications/elibrary-page/?id=13711

[37] T. Hatanaka, K. Uosaki and M. Koga, "Evolutionary computation approach to Wiener model identification," Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No.02TH8600), Honolulu, HI, USA, 2002, pp. 914-919 vol.1, doi: 10.1109/CEC.2002.1007047.