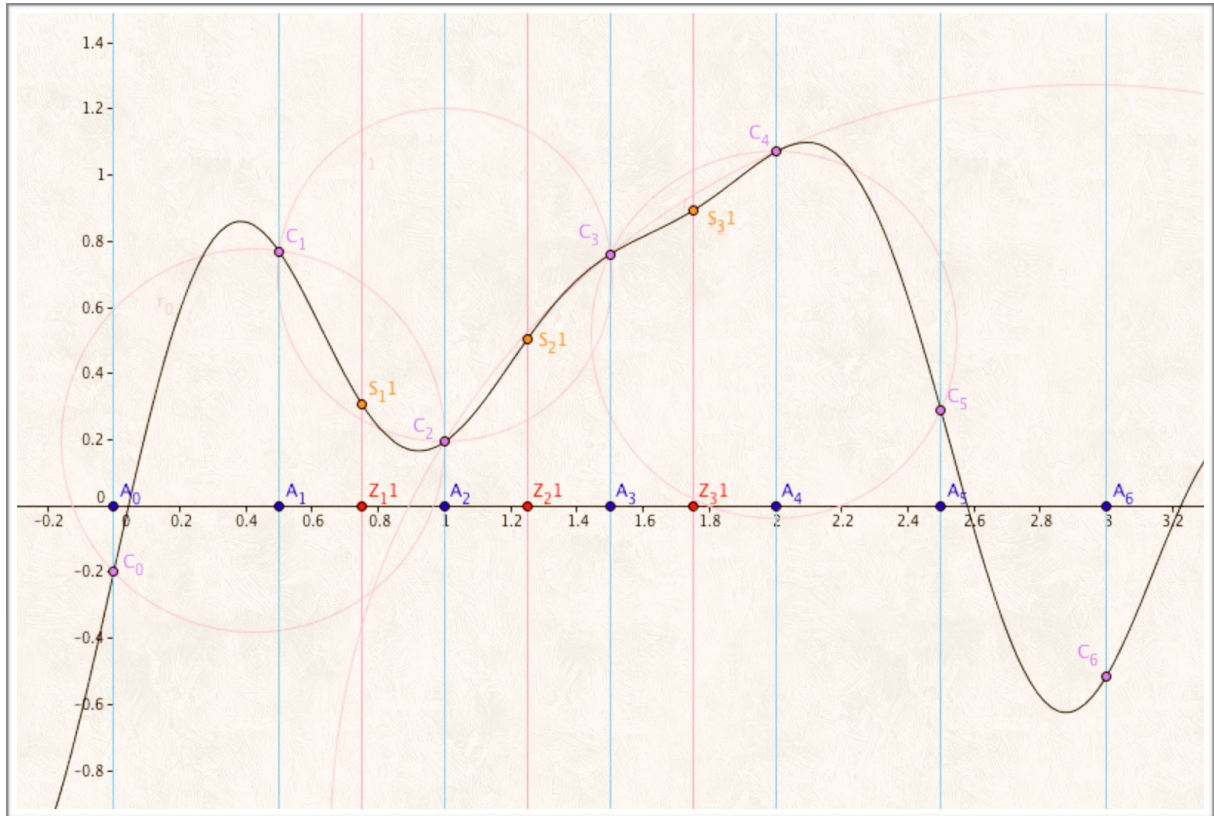


SinWhisky e FLT

Quando una previsione rifinita è (quasi) meglio di una previsione corretta



Riccardo Cecchini

dev@lizard.ink

Autunno 2014

La partenza

Quel monitor del computer di Torino, che rischiara la mezzanotte, tra più catene non interrotte di controviali, tutto a incroci e semafori, a seconda della partenza o ritorno di frenetici cittadini, vien, quasi ad un tratto, a restringersi la pazienza e a prender corso Regina Margherita e fuggire dal fiume di vetture...

Durante l'autunno del 2013 si leggeva sui giornali la notizia che riferiva di un virus capace di comunicare con altri computer sfruttando gli ultrasuoni; fui convinto che fosse una cosa assolutamente interessante, che dovessi realizzarla pure io.

Così, con buone speranze e Visual Studio, incominciai a studiare come poter adempire al mio obbiettivo. Il progetto lo chiamai "TCP Pigeon" perché nel mio immaginario doveva diventare un protocollo di rete trasmesso per l'etere atmosferico. Col passare dei giorni le mie presenze in aula diminuirono e ripresi di mia spontanea volontà a studiare Fourier per poter scoprire le sue teorie e capire come trasformare il segnale audio in entrata in frequenze divise e distinguibili per "demodulare" i vari bit ricevuti. L'algoritmo usato infine, per la precisione, è ovviamente il Fast Fourier Transform, inventato per i calcolatori in principio da James Cooley e John Tukey.

Di risultati ne riuscii ad ottenere, ma non erano così soddisfacenti e precisi come avrei realmente desiderato. I problemi erano essenzialmente lato hardware: la scarsa qualità degli altoparlanti del mio portatile e la (relativamente) scarsa qualità del microfono integrato.

Il primo è un problema piuttosto semplice da risolvere: bisogna prima di tutto fare un'analisi per classificare quali delle alte frequenze vengono eseguite correttamente e quali invece restituiscono un segnale più sporco, per poi riprodurre i segnali più puliti ad una amplificazione accettabile. Oltre che ad altri vari accorgimenti, come per esempio calcolare la distanza e la quantità delle frequenze usate per trasmettere contemporaneamente in modo accettabile. Questi stessi accorgimenti possono essere usati in maniera leggermente differente per sfruttare al massimo la qualità di un altoparlante per la riproduzione musicale, ma ne parlerò più avanti.

Il secondo problema è decisamente più complesso, anche se sinceramente più divertente: se il segnale in entrata non è abbastanza accurato, per esempio soli 16 bit per sample aggiunti ad un sample rate basso, la FFT restituisce valori sporchi e poco precisi.

L'array di sample da elaborare nella FFT deve essere per dimensione una potenza di due (per questioni algoritmiche) e restituisce un'altra array dalla stessa grandezza, contenente le varie sinusoidi che la compongono (e gli eventuali sfasamenti). Per sapere a quale frequenza corrisponde un elemento in posizione *pos* nell'array restituita basta seguire la seguente formula:

$$freq = pos * (sampleRate / fuorierSamples)$$

Con un maggiore sample rate, quindi, in rapporto i dati restituiti con la stessa dimensione di array in input sono più puliti grazie al minore inquinamento dovuto a frequenze intermittenti, anche se la concentrazione è minore (ci sono più frequenze alte, irrilevanti in campo audio). Bisogna trovare una giusta via di mezzo in qualunque caso: con *fourierSamples* troppo basso i dati a disposizione non possono essere sufficienti per scandire correttamente l'onda e con *fourierSamples* troppo alto il risultato sarebbe impreciso dato il cambiamento continuo di quella che dovrebbe essere un'onda periodica, oltre ad uno sforzo decisamente maggiore per il processore.

Bisogna quindi ricordarsi che la trasformazione in serie di Fourier serve per scandire un'onda periodica (cioè la cui funzione non varia nel corso del tempo e che, appunto, tenderà a ripetersi) e determinare tutte le onde armoniche (cioè le semplici onde sinusoidale con una determinate ampiezze e sfasamenti) che la compongono.

Per esempio l'onda che si vede in copertina, apparentemente casuale, corrisponde alla seguente funzione:

Funzione

$$f(x) = -0.2 + \sin(x) + \frac{\sin(x \cdot 4)}{2} + \frac{\sin(x \cdot 6)}{4}$$

Invece l'onda, per esempio, di un brano musicale è un continuo variare di queste sinusoidi, risultando non periodica.

Quindi, come fare ad ottenere un'onda sufficientemente dettagliata per un'analisi corretta?

Meglio, con SinWaver

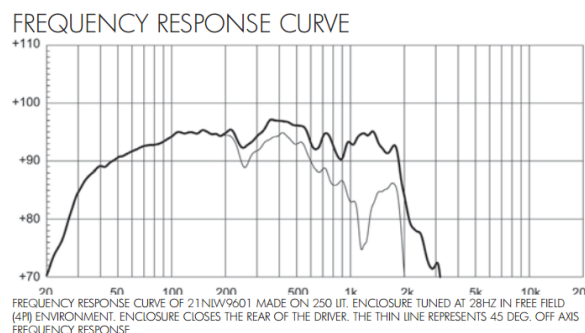
Passarono diversi mesi e ormai la mia anima si stava preparando per l'arrivo di quella che a molti si rivelò un'imbrogliata: l'estate 2014. Ci fu una prima illusoria settimana di caldo afoso, quel tipo di giornate che per qualche strano motivo mi provocano folgorazioni d'apparenti idee geniali, attribuibili presumibilmente alla scarsa affluenza di sangue al cervello.

Il mio quesito di base era piuttosto semplice: si può fare un buon amplificatore combinando due amplificatori mediocri? Quindi se volessi costruirmi una cassa a tre vie di buona qualità senza eccedere nei prezzi potrei usare due casse (o più) per via sfruttando le frequenze che ogni tipologia di cono può suonare meglio senza distorsioni?

Dal mio personale punto di vista era possibile: ci sono un enorme quantità di coni in vendita di mediocre qualità nel range di frequenza per cui sono utilizzati, ma di ottima per particolari range e in determinati casi. Come stabilire quali fossero questi casi in particolare?

Incominciai a scrivere in C# un software che riuscisse ad analizzare un cono grazie ad un microfono e ne detraesse le vere frequenze da sfruttare. Presi in prestito frammenti di codice dal vecchio TCP Pigeon e inventai l'algoritmo che doveva determinare le qualità nascoste di ogni cono.

Ogni produttore di speaker che (più o meno) si rispetti mette a disposizione uno schema con lo spettro di frequenze riprodotte da ogni prodotto, quindi che senso avrebbe un programma che esegue nuovamente le stesse operazioni, magari con minore precisione a causa della scarsa qualità degli strumenti a disposizione.



Un esempio di altoparlante con grafico della curva di risposta alle frequenze

Certo, i dati riportati sono già molto utili per comprendere a grandi linee che speaker utilizzare per il progetto, ma mancano informazioni sul come devono interagire i due speaker per ottenere il suono corretto. Chiamai il software SinWaver e la sua funzione era quella di stressare a tal punto un cono da ottenere qualsiasi minima informazione a riguardo: non doveva limitarsi a

controllare frequenza per frequenza quale fosse la massima amplificazione restituita senza distorsioni e l'amplificazione reale restituita, bensì controllare come tutte le serie combinazioni di frequenze si comportavano, qual erano quelle che resistevano maggiormente ad amplificazioni maggiori e quali invece dovevano essere accompagnate dall'altro speaker o addirittura tagliate.

Infine, il secondo passo è quello di unire i risultati degli speaker da usare per la cassa ed ottenere i livelli di equalizzazione corretti speaker per speaker, addirittura specificare in quali casi una stessa frequenza dovesse essere suonata da un cono, in quali da un'altro e quali contemporaneamente.

Progetto complesso ma ambizioso: una volta parzialmente completato il secondo passo è risultato più che evidente che SinWaver, se avessi voluto distribuirlo anche a semplici audio-amatori, avesse bisogno di microfoni di alta qualità.

Ecco il mio nuovo quesito: posso prendere un'onda in entrata con un sample rate basso e aumentarle la precisione prevedendo i sample non specificati?

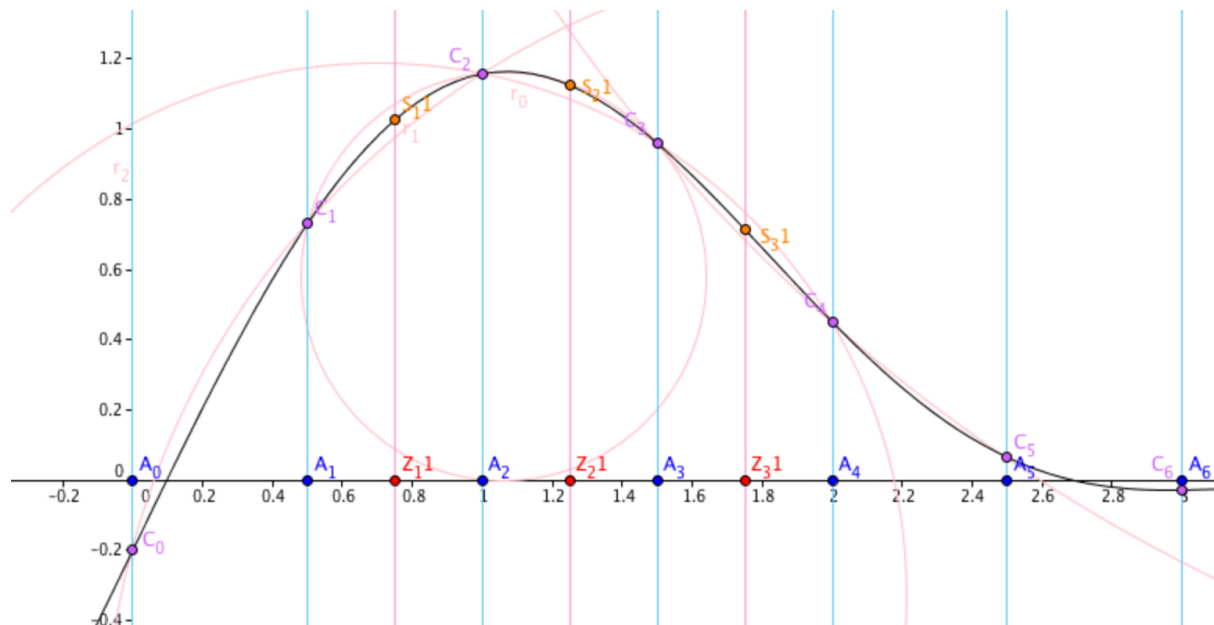
Sì e no, fu la mia risposta d'impulso: la questione non è nuova nel mondo dell'elettronica: algoritmi e circuiti di smoothing vengono usati ogni giorno in ogni genere di sistema per rendere ascoltabili segnali audio altrimenti eccessivamente in bassa qualità, dove naturalmente la traccia non viene migliorata ma fan solo sì che il suono sia udibile a livello umano riducendo gli sbalzi improvvisi di voltaggio. La prima idea che mi venne in mente fu quella di affrontare il dilemma a livello geometrico: se tracciassimo delle circonferenze passanti per ogni sample si potrebbe ottenere in maniera un po' più precisa l'andamento delle curve, anche se era ovvio che con sample rate troppo bassi sarebbe stato praticamente impossibile ottenere previsioni accettabili. Incominciai a scrivere il nuovo algoritmo da aggiungere a SinWaver e lo nominai SinWhisky, perché pretendere di poter intuire con precisione l'andamento non specificato di un'onda non periodica per me era come pretendere di intuire l'andamento della camminata di un ubriacone.

Ma, si sa, tentar non nuoce.

Come funziona SinWhisky

Ce l'avevo fatta. Il risultato è stato piuttosto soddisfacente: passando sotto SinWhisky una canzone da soli 8.000 sample al secondo potevo riottenere lo stesso file a 48.000 s/s con una qualità acustica decisamente migliore. L'unico difetto che perdurava era relativo all'equalizzazione: le frequenze più alte vengono facilmente tagliate, quindi bisognava studiare un metodo per recuperarle, metodo su cui avevo intenzione di riflettere in spiaggia quegli unici tre giorni al mare che le condizioni del tempo mi permettevano di fare. Credo che avesse più o meno avuto la stessa idea il "buon uomo" che, il primo giorno di vacanza, mi ha scassinato la macchina e mi ha rubato la borsa con computer, backup e soprattutto tutti i miei ultimi lavori. SinWaver si è salvato grazie al salvataggio su Dropbox insieme a delle porzioni di codice non ancora del tutto funzionanti di SinWhisky, nonostante tutto non mi persi d'animo. Compresi che per poter riprendere a lavorare seriamente al progetto non potevo più lavorare da solo: raccolsi un paio di amici nonché programmatori e creammo un piccolo team che chiamammo Lizard.

Nuovo computer e ricominciamo subito a riscrivere l'algoritmo di SinWhisky, questa volta in C per poterci assicurare un porting lineare su altre piattaforme.



Il concetto finale di SinWhisky è piuttosto semplice: per due punti possono passare infinite circonferenze, per tre solo una. Per ogni punto, a parte il primo e l'ultimo, si crea una circonferenza passante anche per i due punti più vicini. Infine si riscrivono i samples seguendo gli andamenti medi di queste

circonferenze. In realtà è impossibile mostrare con precisione il funzionamento dell'algoritmo con una semplice rappresentazione grafica: ci sono diversi punti che possono essere modificati e sfruttati per rendere il tutto più funzionale.

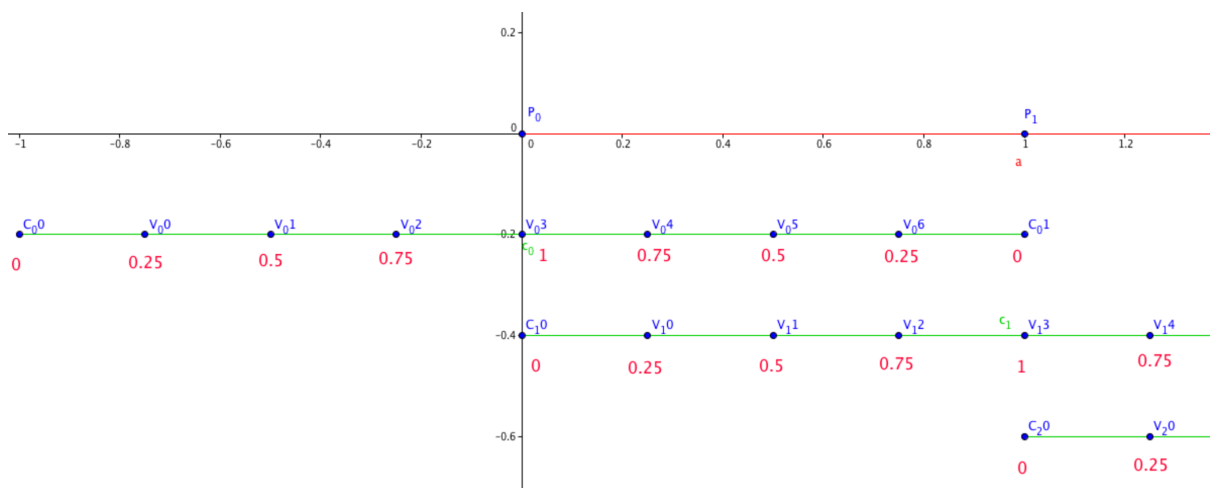
Per esempio dati tre punti consecutivi p_1 , p_2 e p_3 i tre avranno lo stesso incremento sull'asse delle ascisse, nell'algoritmo uguale a 1. Ma talvolta le ordinate hanno variazioni eccessive, tanto da restituire circonferenze da un raggio enorme. Il problema viene risolto cambiando temporaneamente l'incremento dell'ascisse prima del calcolo della circonferenza:

```
//Calc aperture
float maxapr = 0;
maxapr = max(fabs(circle->ys[0] - circle->ys[1]), maxapr);
maxapr = max(fabs(circle->ys[1] - circle->ys[2]), maxapr);
maxapr = max(fabs(circle->ys[0] - circle->ys[2]), maxapr);

circle->apr = maxapr;
for(int i=0; i<3; i++) circle->xs[i] *= circle->apr; //Set the new aperture
```

Come si deduce dal codice attualmente ci si limita a modificare “l’apertura” della circonferenza a seconda dell’alterazione delle ordinate con una formula piuttosto semplice e perfezionabile.

Un altro esempio può essere il come le varie circonferenze vengono unite per restituire una linea continua di previsione. Attualmente l'algoritmo può essere migliorato sotto diversi punti di vista, per esempio si limita ad unire le tre circonferenze che, con certezza, passano per il punto da prevedere, mentre nella previsione grafica si può notare che le circonferenze possono andare oltre ai tre punti circoscritti, dando potenzialmente maggiore precisione al disegno dell'onda. Anche il modo con cui le ordinate delle circonferenze vengono unite possono avere una grossa miglioria: l'algoritmo ‘rubato’ si limitava a fare una semplice media statistica in rapporto alla distanza dei punti a cui appartenevano le circonferenze.



Nella nuova versione abbiamo preferito favorire i valori centrali di una circonferenza a discapito di quelli estremi, ponderando le circonferenze per il seno degli stessi valori. Purtroppo, dovendo lavorare ancora molto a riguardo, è ancora lunga la strada per la scelta delle metodologie più efficienti, ma una cosa è certa: più si approfondisce l'algoritmo e più risorse saranno necessarie per eseguirlo. E questo in molti casi potrebbe rendere il tutto estremamente pesante. Attualmente SinWhisky conta solo 300 linee di codice in C, un ottimo rapporto per il risultato ottenuto.

```

Quante serie di numeri vuoi inserire? 5
Di quanto vuoi zoommare i valori? 3
0: 3
1: 7
2: 9
3: 4
4: 6

Ok, elaboro.
0: 3.000000
1: 4.157316
2: 5.213207
3: 6.162361
0: 7.000000
1: 7.759608
2: 8.408535
3: 8.854186
0: 9.000000
1: 7.977124
2: 6.500000
3: 5.022877
0: 4.000000
1: 4.014061
2: 4.369707
3: 5.045354
0: 6.000000
Program ended with exit code: 0

```

Avendo creato l'interfaccia di SinWaver per Windows, nel riscrivere SinWhisky per le dimostrazioni semplici ci siamo limitati ad un temporaneo utilizzo della linea di comando. L'algoritmo richiede un valore "zoom", che indica quanti sample devono essere aggiunti in mezzo a quelli conosciuti. In questo caso, per esempio, ho impostato il valore a tre e, come output, ha restituito tutti i valori specificati più i tre per sample previsti.

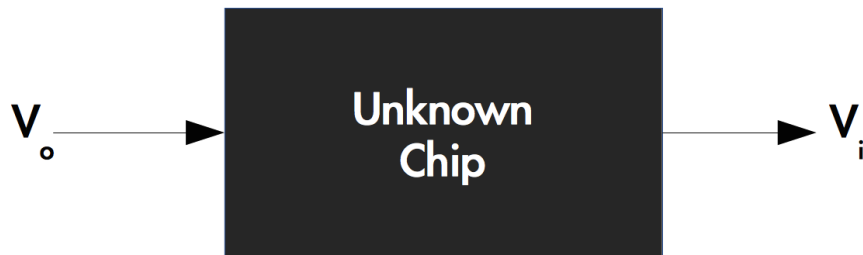
Okay, ma come detto in precedenza i sample in output non potranno mai essere davvero precisi e realistici, neanche se si apportassero cambiamenti chirurgici a SinWhisky. L'onda in uscita potrà avere un comportamento ancora più naturale, ma le frequenze più alte rimarranno inevitabilmente tagliate restituendo un suono poco vivo.

Quindi?

Fast Linear Transform

Durante una qualunque trasformazione, che sia dovuta a SinWhisky, o a una sua variante, oppure a qualsiasi altro algoritmo, ci sono delle alterazioni più o meno visibili nell'onda d'uscita rispetto a quella entrante. In qualunque caso, si può riportare l'onda ad uno stato di maggiore naturalezza?

Immaginiamo di avere un'onda complessa in ingresso ad un integrato casuale V_o e un'onda in uscita V_i , come si può riportare V_i allo stadio originale pur non conoscendo la funzione inversa dell'integrato?



L'unica possibilità a nostra disposizione è di agire dinamicamente sull'equalizzazione di V_i di volta in volta cercando di recuperare la tendenza originale dell'onda in ingresso. Questo utilizzando la Fast Fourier Transform e la Fast Linear Transform. D'ora in poi FFT e FLT.

Prima di tutto, dobbiamo scegliere il range di onde su cui siamo interessati ad operare: se è un'onda acustica è tipicamente impostabile da 0 a 11000 Hz, estremi non compresi. Secondo, la quantità di sample che vogliamo dare di volta in volta in elaborazione alla FFT: questo determina esattamente a quali frequenze otterremo i risultati, come detto in precedenza, determinato anche dal sample rate del segnale analizzato.

Per esempio, dato un segnale in ingresso con una definizione di 96000 sample al secondo (altissima qualità), elaborando in FFT 128 sample per volta avremo 14 frequenze che possono interessarci durante la FLT. In seguito mostro il risultato di un foglio di calcolo che restituisce in automatico le frequenze da prendere in considerazione e la loro quantità a seconda dei parametri dati.

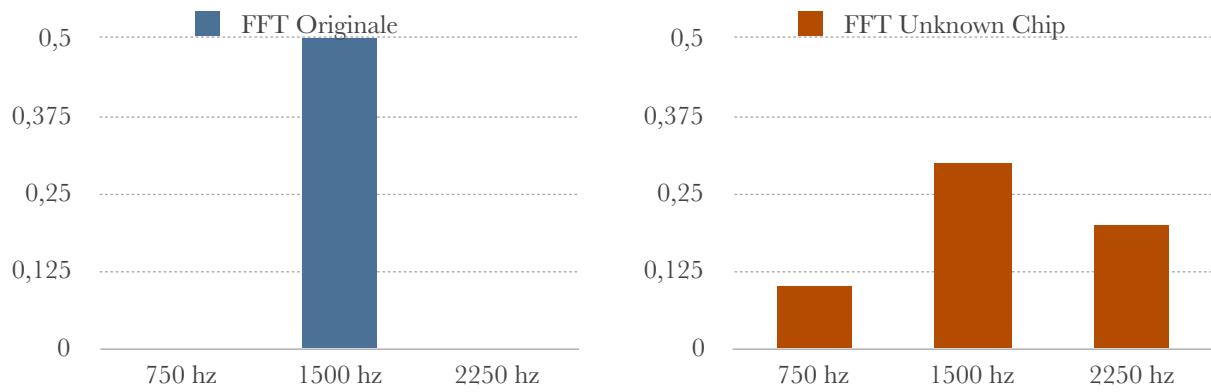
Standard SampleRate:	96000
Standard FFT Samples:	128
Interested freqs:	0 to 11000

Sample	Freq	R
0	0,0	
1	750,0	x
2	1500,0	x
3	2250,0	x
4	3000,0	x
5	3750,0	x
6	4500,0	x
7	5250,0	x
8	6000,0	x
9	6750,0	x
10	7500,0	x
11	8250,0	x
12	9000,0	x
13	9750,0	x
14	10500,0	x
15	11250,0	
16	12000,0	
17	12750,0	

Freqs to transform: 14

Quattordici frequenze è un numero accettabile nei casi più semplici, dato che riequalizzare più di 14 bande per volta potrebbe risultare complicato: è una quantità abbastanza bassa per essere presa in esame come esempio ma non per vere e proprie applicazioni, dato che solo 128 sample su un segnale di 96000 s/s corrispondono solo ad un lasso di tempo di un millesimo di secondo ($1.33 \cdot 10^{-3}$), non sufficienti per una ritrasformazione precisa su più fronti, oltre che ad una definizione delle bande troppo bassa. L'ideale, in questo caso, sarebbe di 256 o 512 sample.

Ora, cosa ce ne facciamo di queste 14 frequenze? Le passiamo una ad una per il chip. In questo caso iniziamo con la sinusoide di 1500 hertz esatti. Generato un insieme di 128 sample con un'onda di questa frequenza di ampiezza 0.5 su un valore massimo teorico uguale ad 1, passandola sotto la FFT restituirà il corretto valore alla posizione 2 con l'ampiezza originaria. Se invece eseguiamo la FFT sugli stessi sample ma elaborati prima dall'Unknown Chip restituirà dei valori sporchi e fuorvianti, mostrato nei grafici successivi.



Eseguendo questa operazione frequenza per frequenza si ottengono i valori di come ogni sinusoide semplice venga in realtà percepita in seguito al passaggio per l'Unknown Chip. Se sapessimo che per il chip possono passare solo una di queste onde in particolare per volta sarebbe piuttosto semplice comprendere quale sinusoide è stata appena trasformata e riequalizzarla (in questo caso basterebbe rigenerarla) al valore originale. Ma una traccia audio è composta dall'unione ad ampiezze diverse di tutte queste, quindi bisogna riottenere i valori reali conoscendo tutte le trasformazioni. Qui entra in gioco la Fast Linear Transform.

Per comprendere con semplicità il funzionamento della FLT supponiamo che la riequalizzazione riguardi un insieme di soli tre valori. Sapendo che ogni frequenza trasformata può avere un valore all'interno anche di altre frequenze si può dedurre che la risoluzione del problema si trovi in un semplice sistema di equazioni lineari di primo grado:

$$\begin{cases} V(0) = x_0 \cdot a_0(0) + x_1 \cdot a_1(0) + x_2 \cdot a_2(0) \\ V(1) = x_0 \cdot a_0(1) + x_1 \cdot a_1(1) + x_2 \cdot a_2(1) \\ V(2) = x_0 \cdot a_0(2) + x_1 \cdot a_1(2) + x_2 \cdot a_2(2) \end{cases}$$

La matrice $V(n)$ contiene i valori conosciuti ottenuti dalla FFT del segnale trasformato, mentre $a_f(y)$ contiene i valori conosciuti della frequenza f dispersa nelle restanti frequenze y . Infine le incognite x_n si riferiscono all'amplificazione di ogni onda pulita.

Per esempio, sappiamo che $V(0)$ è composta dalla somma di tutte le possibili frequenze disperse in seguito alla trasformazione con una data amplificazione.

Più complicato è stato tramutare in codice la FLT: avremmo potuto usare un math parser a cui far risolvere le varie equazioni, ma sarebbe stato eccessivamente lento e pesante. Così in una settimana abbiamo provato a creare diversi algoritmi che riuscissero a risolvere la questione meccanicamente;

Nei primi due algoritmi abbiamo provato ad affrontare la cosa a livello “umano”, cioè come uno studente alle superiori proverebbe a risolvere un sistema di equazioni. Il risultato era positivo, ma se mancavano diverse incognite, cambiando l'ordine di risoluzione delle equazioni i risultati, ovviamente, cambiavano. In ogni caso validi, perché con $x+y=4$ è sia vero che x sia uguale a 3 e y sia uguale ad 1 che viceversa. Come cercare il risultato statisticamente più probabile? Basta imporre all'incognita la somma tutte le equazioni in cui essa è contenuta.

Su un'equazione con due incognite un algoritmo impreciso semplificherebbe così:

$$\begin{cases} x_0 = \frac{-x_1 \cdot a_1(0) + V(0)}{a_0(0)} \\ x_1 = \frac{-x_0 \cdot a_0(1) + V(1)}{a_1(1)} \end{cases}$$

Invece se non si vuole escludere nessun valore a disposizione deve agire in questa maniera:

$$\begin{cases} x_0 = \left(\frac{-x_1 \cdot a_1(0) + V(0)}{a_0(0)} + \frac{-x_1 \cdot a_1(1) + V(1)}{a_0(1)} \right) \cdot 2^{-1} \\ x_1 = \left(\frac{-x_0 \cdot a_0(1) + V(1)}{a_1(1)} + \frac{-x_0 \cdot a_0(0) + V(0)}{a_1(0)} \right) \cdot 2^{-1} \end{cases}$$

Il tutto mantenendo delle buone performance per il processore. Questo algoritmo occupa soltanto 350 linee di codice restituendo i risultati corretti in doppia precisione floating. Esclude automaticamente le incognite trascurabili (dove le frequenze non hanno alcuna alterazione) per velocizzare la risoluzione dei dati.

Conclusioni

Ho preso in esame l'Unknown Chip per dimostrare come la FLT possa essere utilizzata in molteplici casi. Con SinWhisky bisogna generare l'onda nella qualità originale e ritrasformarla per ottenere i valori interessati alla FLT. I valori risultanti possono essere a loro volta utilizzati per definire un rapporto fra i valori della FFT del segnale trasformato con quelli ottenuti dalla FLT e trasformarli in un equalizzatore che corregga il tutto.

Queste tecnologie possono essere usate per migliorare nettamente la qualità audio di un suono in ingresso in maniera simulata o per gestire l'equalizzazione di un amplificatore.

Questa relazione è stata scritta per la Eighteen Sound al fine di presentare le potenzialità dei progetti in sviluppo dalla Lizard:

Siamo in cerca di un valido scambio di idee e di valori con altre case di sviluppo italiane per non soccombere nell'aridità dell'immaginario tecnologico italiano.

Aspettiamo considerazioni in merito, grazie.

Riccardo Cecchini

Lizard.ink

26/09/2014