a cura di Francesco Romani

Riduzione di rumore con metodi spettrali

Viene presentata un'implementazione didattica di un riduttore di rumore per segnali audio, basato sulla elaborazione nel dominio della frequenza. Come al solito la trattazione serve solo ad introdurre un esempio di applicazione Mathematica e non ha pretese di completezza.

Introduzione

Da quando la potenza dei personal computer ha raggiunto quella dei *mainframe* di una volta, molte elaborazioni che allora erano appannaggio dei laboratori della NASA adesso sono alla portata di tutte le tasche. Un esempio significativo sono i riduttori di rumore che permettono di "ripulire" vecchie registrazioni su nastro o su vinile prima del loro trasferimento su CD.

I problema di quanto questi trattamenti siano poi gradevoli al palato (pardon all'udito) è stato oggetto di vari articoli sulle riviste specializzate e non viene affrontato qui. Il nostro scopo è invece quello di implementare in *Mathematica* un riduttore di rumore che, se da un lato a causa della sua inefficienzanon si presta a scopi pratici, dall'altro è adattissimo a fare da banco di lavoro per misure ed esperimenti. Come algoritmo di riferimento abbiamo preso quello descritto da Fabrizio Montanucci in un numero di Audio Review (marzo 1998) dedicato al trattamento digitale dei segnali audio.

L'algoritmo

L'idea che sta alla base del riduttore di rumore è la seguente. Si dispone di un campione di rumore e se ne ricava lo spettro di frequenza. Si prende il segnale (ovvero un lungo vettore di valori numerici) lo si divide in fette (vettore di campioni digitali di lunghezza limitata). Ciascuna fetta viene trasformata nel dominio della frequenza (con l'ausilio della *Fast Fourier Transform*) e, in qualche modo ad essa viene sottratto il rumore, si ritorna poi nel dominio del tempo e si ricompone il segnale riattaccando le fette. Purtroppo un tale procedimento è più facile a dirsi che a farsi. Sorgono infatti alcuni problemi di natura matematica.

- Il segnale da ripulire è un segnale musicale il cui contributo spettrale varia rapidamente nel tempo e la scelta delle dimensioni delle fette è critica. Se le fette sono piccole (meno di 4000 campioni) non vi è abbastanza risoluzione (specialmente alle basse frequenze). Se le fette sono troppo grandi il costo della FFT aumenta e si tende a mediare tra contributi spettrali in istanti differenti.
- Per ottenere una approssimazione ragionevole dello spettro del segnale occorre usare una **finestra di pesatura** che mette a zero i primi e gli ultimi campioni di ogni fetta. Questo fa si che solo la parte centrale del segnale antitrasformato abbia significato.
- Per ovviare a questo secondo problema è necessario sovrapporre le fette tenendo per buona solo la parte centrale.

Con questi accorgimenti, anche prescindendo dal metodo vero e proprio di ripulitura, abbiamo una famiglia di algoritmi che dipendono dai seguenti parametri:

- la lunghezza della fetta su cui si lavora;
- il fattore di sovrapposizione tra le fette;
- il tipo di finestra utilizzata.

L'esperimento che vogliamo implementare consiste nei seguenti passi.

- Lettura di un file audio mono della durata di pochi secondi.
- Aggiunta al segnale "buono" di un rumore predeterminato (nel nostro caso un tono continuo di 1000Hz a -20dB).
- Scelta dei parametri e delle finestra.
- Analisi spettrale di un campione di rumore con i parametri scelti.
- Suddivisione del segnale audio in fette.
- Applicazione della finestra,
- Calcolo della FFT.
- Confronto tra lo spettro del segnale e quello del rumore ed eliminazione delle bande in cui il rumore è dominante.
- Calcolo della FFT inversa
- Applicazione della finestra inversa alla parte centrale della fetta.
- Ricomposizione del segnale per giustapposizione

Contro ogni aspettativa tutto questo ben di Dio si può implementare in Mathematica con poche istruzioni (anche se poi le richieste di tempo e di memoria rendono l'algoritmo non utilizzabile in pratica).

Implementazione

Fissiamo anzitutto l'intervallo temporale tra i campioni (ovvero l'inverso della frequenza di campionamento), seguendo lo standard CD audio.

In[1]:=

```
δ = 1./44100.;
```

Gli altri parametri che useremo sono **Len** (la lunghezza della fetta), **overlap** il fattore di sovrapposizione e **Lblock = len/overlap** (la parte di fetta utile)

La funzione **sample** permette di campionare una funzione matematica ad intervalli di δ secondi e la funzione **zeri** genera un segnale nullo.

In[2]:=

```
Sample[f_, n_] := Table[N[f[i δ]], {i, n}]
zeri[l_] := Array[0&, l];
```

La finestra **win** è una funzione predefinita che vale 0 in 1 e -1 e 1 in 0; **win** viene valutata su un numero di punti pari alla lunghezza della fetta generando un array **windowL**. La finestra inversa viene valutata solo nella parte centrale per una lunghezza **Lblock** (evitando così la divisione per zero).

In[3]:=

```
InitWindow[win_] := (
  windowL=
   Table[N[win[2i/(Len-1)-1]],{i,0,Len-1}];
  windowS=
   1/Take[windowL,{(Len-Lblock)/2+1,
   (Len+Lblock)/2}];)
```

L'*array* **SND** contiene i data ancora da trattare, **BUFFER** la fetta in corso di elaborazione. La funzione **scorri** taglia da **SND** una fetta lunga Len avanzando ogni volta di **Lblock** posizioni, il taglio avviene come effetto di bordo, la funzione rende **True** fino a che c'è roba da tagliare. L'array **SND** andrà inizializzato aggiungendo un numero opportuno di zeri in cima e in fondo al segnale originale.

In[1]:=

```
scorri := If[Length[SND] ≥ Len,
    BUFFER = Take[SND, Len];
    SND = Drop[SND, Lblock];
    True,
  False, False]
```

La funzione **GOOD** rende la parte buona della fetta.

In[1]:=

```
GOOD := Take[BUFFER, {(Len - Lblock)/2 + 1, (Len
+ Lblock)/2}];
```

La funzione **filter** prende due *array* **buf** e **campione** e azzera in **buf** quelle componenti che in valore assoluto sono minori delle corrispondenti componenti di **campione**

In[1]:=

```
filter[buf_, campione_] := filter1 /@ Transpo-
se[{x, y}]
filter1[{x_, y_}] := If[Abs[x] < y, 0, x]
```

Il programma **elab** riceve in input i campioni, la lunghezza della fetta, il fattore di sovrapposizione e la finestra da utilizzare e mette tutto insieme.

In[1]:=

```
elab[snd_, len_, overlap_, win_] := (
    Len = len;
    Lblock = len/overlap;
    InitWindow[win];
      noiseREF = 2 Abs[Fourier[Sample[noise,
Len]*windowL]];
     SND = Join[zeri[(Len - Lblock)/2], snd, ze-
ri[(Len + Lblock)/2]];
    RES = {};
    While[scorri ,
      BUFFER = Fourier[BUFFER*windowL];
      BUFFER = filter[BUFFER, noiseREF];
      BUFFER = Chop[InverseFourier[BUFFER]];
      AppendTo[RES, GOOD*windowS]];
    Take[Flatten[RES], Length[snd]])
```

La trasformata diretta di Fourier (funzione **Fourier[]** *built-in* in *Mathematica*) viene applicata prima al campione di rumore e poi ad ogni fetta.

La trasformata inversa di Fourier (funzione **InverseFourier[]**) viene applicata dopo il trattamento della fetta. Si noti che nel dominio del tempo abbiamo numeri reali, nel dominio della frequenza quantità complesse. Poiché il filtraggio non altera le proprietà di simmetria l'antitrasformata dovrebbe essere reale, in pratica **Chop** elimina le piccole componenti immaginarie dovute agli errori numerici. Attenzione, quando si lavora con linguaggi a basso livello (C, Pascal Java) tutte le elaborazioni sono su numeri reali (o coppie di numeri reali per rappresentare i complessi) e conviene sfruttare le simmetrie per usare con vettori più piccoli; con *Mathematica* invece è meglio usare **Fourier** nel campo complesso in quanto la ridondanza è meno costosa del codice necessario per sfruttare le simmetrie.

Le finestre

Come abbiamo già detto prima di calcolare la trasformata di Fourier è opportuno pesare il segnale con una finestra (una funzione non negativa con valore massimo 1 che vale zero al di fuori dell'intervallo in esame). La finestra banale è quella *Rettangolare* (che vale 1 all'interno di tutto l'intervallo) finestre più consigliabili sono quella *Triangolare* e soprattutto quella di *Hanning* che ha una espressione basta su funzioni trigonometriche.

In[1]:=

```
Rect[x_] := 1;
```

In[2]:=

```
Triangle[x_] := 1 - Abs[x];
```

In[3]:=

```
Hanning[x_] := N[1/2 + 1/2 Cos[ x]];
```

La **Figura 1** mostra le tre finestre nel dominio del tempo (rettangolare nera, triangolare rossa, Hanning blu).

Per apprezzare la influenza delle finestre sull'analisi di Fourier è bene studiarle nel dominio della frequenza. Infatti quando si approssima uno spettro continuo con una trasformata discreta si ottiene un insieme di valori distinti, ognuno dei quali somma il contributo di molte frequenze diverse. Si può dimostrare che ogni valore è un **cassetto** (in inglese ***bin***) in cui va a finire l'integrare del prodotto dello spettro vero per la trasformata discreta della finestra, centrata sulla banda in esame (discorso terribile che sostituisce informalmente 2 pagine di formule matematiche). La finestra ideale dovrebbe avere una trasformata rettangolare ma nessuna finestra di

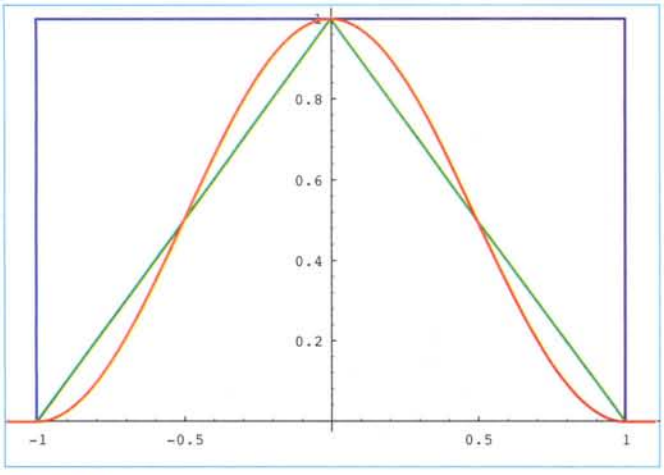


Figura 1

lunghezza finita può avere una trasformata di lunghezza finita. Nella **Figura 2** vediamo le trasformate delle tre finestre in esame confrontate con l'irraggiungibile ideale (in nero) Si nota come la finestra di *Hanning* (quella rossa) sia un accettabile compromesso, mentre la finestra triangolare e soprattutto quella rettangolare introducono contributi di frequenze lontane da quella in esame.

Esperimenti

Dapprima si legge il segnale audio (già portato in **AIFF**-mono con un programma di utilità). Il segnale viene normalizzato ovvero trasformato in un vettore di numeri reali di valore assoluto al più 1. (N.B. Import funziona anche su PC con un segnale **WAW**).

In[1]:=

```
snd = Import["sample.AIF", "AIFF"][[1, 1]];
snd = snd/Max[Abs[snd]];
NMAX = Length[snd]
time = NMAX/44100. Second
```

Out[1]:=

```
3.72812 Second
```

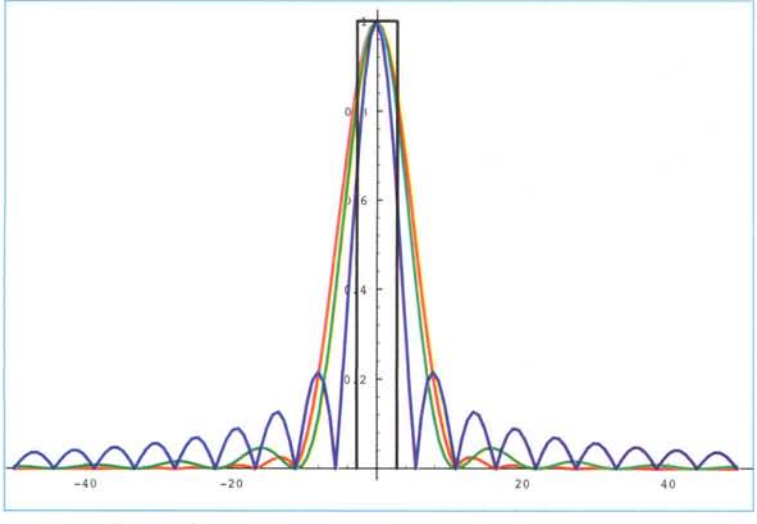


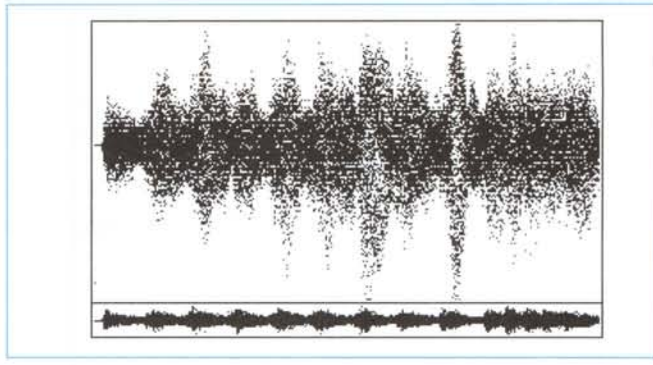
Figura 2

Il risultato può essere suonato con **ListPlay[]**

In[2]:=

```
ListPlay[snd, SampleRate -> 44100];
```

(Vedi Figura 3)



Si definisce la funzione rumore, la si campiona e la si aggiunge al segnale.

In[2]:=

```
noise[x_] := Cos[2 p 1000 x]
sndR = snd + Sample[noise, NMAX]/10;
```

Il risultato presenta un disturbo a 1000 Hz, perfettamente udibile.

In[3]:=

```
ListPlay[sndR, SampleRate -> 44100];
```

A questo punto si sceglie la finestra di *Hanning*, le dimensioni della fetta (16k), il fattore di overlap (4) e si può ripulire il segnale.

In[4]:=

```
RES = elab[sndR, 16*1024, 2, Hanning];// Timing
```

Out[4]:=

```
{32. Second, Null}
```

Il risultato è molto simile al segnale originale, udire per credere

In[5]:=

```
ListPlay[sndR, SampleRate -> 44100];
```

L'elaborazione di 3 secondi di musica ha richiesto circa 32 secondi di tempo e 30 Mb di memoria.

In[6]:=

```
MaxMemoryUsed[]/1024./1024. Mb
```

Out[6]:=

```
28.3592 Mb
```

È ovvia la valenza puramente didattica di questa implementazione.

È stata fatta una implementazione Java dello stesso algoritmo capace di leggere e scrivere *file* AIFF stereo e di elaborare un intero CD praticamente in tempo reale (su un Mac G3@400).

Tale programma è stato usato per provare l'influenza sul suono della scelta dei parametri.

MC

Bibliografia

Fabrizio Montanucci, The Audio Tool , Audio Review n. 179 Marzo 1998.

in edicola

WoW World of Web
è una pubblicazione
Pluricom
INFORMAZIONE e FORMAZIONE