**Laboratorio di Telecomunicazioni - a.a. 2010/2011**
**Lezione n. 5**

# Esempi di elaborazione dei segnali

L.Verdoliva

In questa quarta lezione ci occuperemo di rappresentare ed elaborare in Matlab segnali "reali" come le immagini e il segnale audio. Consideremo alcune delle operazioni LTI studiate nelle scorse lezioni e le applicheremo a questi segnali per comprenderne meglio i loro effetti.

## 1 Rappresentazione delle immagini

Un'immagine digitale (monocromatica) può essere rappresentata in Matlab con una matrice bidimensionale il cui generico campione $x(m, n)$ è chiamato *pixel* (picture element) e varia nel range $[0, K-1]$. Le immagini sono quindi rappresentate con $K$ livelli di grigio, e ogni pixel è codificato con $\log_2 K$ bit. Solitamente le immagini sono rappresentate con 256 livelli di grigio (8 bit sono allora necessari alla codifica di ogni campione).
Scriviamo i comandi Matlab per leggere da file un'immagine in formato JPEG, e che ci consentono di memorizzarne i valori in una matrice $X$:

```
>> X = imread('nome_file.jpg');  % leggiamo il file
>> [M,N] = size(X);              % memorizziamo le dimensioni in M e N
>> whos;                         % cosa abbiamo adesso in memoria?
```

Attraverso il comando `imread` è possibile leggere immagini con formato standard tipo jpeg, bitmap, tiff, gif, bmp. Fate attenzione al fatto che le immagini che volete leggere devono trovarsi nella directory corrente, in alternativa potete dare il corrispondente percorso. Una volta in memoria l'immagine può essere visualizzata nel modo seguente:

```
>> figure(1);          % apriamo una figura
>> image(X);           % visualizza l'argomento come immagine
```

Se $X(m, n) = k$, il comando `image` disegna nel punto $(m, n)$ un pixel col $k$-esimo colore disponibile. I colori sono definiti assegnando le corrispondenti quantità di rosso, verde e blu (RGB) nella palette (tavolozza); per definire una palette con $K$ toni di grigio si può usare il comando gray(K); ad esempio, attraverso il comando

```
>> colormap(gray(256));
```

si definisce una tavolozza di colori su livelli di grigio, tale da associare pixel scuri a valori piccoli (0, 1...) e pixel chiari a valori grandi (...254, 255). Notate come nonostante il segnale sia discreto, il campionamento effettuato sull'immagine analogica, è stato così fitto da dare la sensazione di continuità. Solo effettuando uno zoom (cliccate sul simbolo con la lente di ingrandimento) è possibile accorgersi della natura discreta del segnale.
La visualizzazione così realizzata non si adatta però alle dimensioni reali dell'immagine, allora per evitare distorsioni aggiungete `axis square` se l'immagine è quadrata, o più in generale `axis image` che adatta le dimensioni della figura a quelle dell'immagine.

## 1.1 Esempi ed esperimenti proposti

a) *Lettura di un'immagine*

Utilizzando i comandi visti nel paragrafo precedente, provate a scrivere una funzione *vedi* in grado di visualizzare le immagini su 256 livelli di grigio in formato JPEG. Il prototipo potrebbe essere il seguente:

```
function X = vedi(nome_immagine);
```

b) *Analisi di un'immagine*

Le caratteristiche di un'immagine possono essere valutate semplicemente con i seguenti comandi:

```
>> x = X(:);              % serve a 'vettorizzare' l'immagine
>> xmin = min(x)          % minimo
>> xmax = max(x)          % massimo
>> xmed = mean(x)         % media
>> xene = sum(x.^2)       % energia
```

Se cercate di eseguire l'ultimo comando si verifica errore, infatti Matlab realizza tutte le operazioni interne con precisione massima (64 bit), mentre i valori dell'immagine sono stati memorizzati come interi senza segno su 8 bit (uint8). E' conveniente quindi rappresentare i valori dell'immagine in formato double, per evitare che Matlab generi errore. Questo significa che, prima di realizzare qualsiasi operazione, bisogna realizzare una conversione di tipo:

```
>> X=double(X);
```

Se si vuole visualizzare invece l'andamento di una riga o di una colonna dell'immagine, bisogna eseguire i seguenti comandi:

```
>> x=X(50,:);             % estraiamo la riga n.50
>> plot(x);               % visualizziamo l'andamento
```

Il comando `plot` unisce con dei segmenti i punti corrispondenti ai valori discreti del segnale (interpolazione lineare).

c) *Immagine a colori*

Scriviamo i comandi per visualizzare un'immagine a colori ('fragole.jpg' e 'fiori.jpg') e le corrispondenti componenti di Rosso, Verde e Blu:

```
>> X=imread('nome_file.jpg')
>> R=X(:,:,1); G=X(:,:,2); B=X(:,:,3);
>> figure(1); colormap(gray(256)); image(R);
>> figure(2); colormap(gray(256)); image(G);
>> figure(3); colormap(gray(256)); image(B);
```

Leggendo l'immagine contenuta nel file 'fiori.jpg' si può notare che la componente di Blu è più luminosa in quanto è il colore predominante dell'immagine. Se si vuole memorizzare su di un file una delle componenti dell'immagine, basta digitare il seguente comando:

```
>> imwrite(B,'Blu.jpg');
```

## 2 Elaborazione delle immagini

Data un'immagine discreta, vogliamo valutare l'effetto di un'elaborazione LTI. Cominciamo col considerare una riga dell'immagine 'lena.jpg' e proviamo a filtrarla con un filtro FIR con $M = 7$ coefficienti di valore $1/M$ (i campioni in uscita non sono altro che la media aritmetica su $M$ valori in ingresso), digitiamo la sequenza di comandi:

```
>> x=X(200,:);          % preleviamo la riga n.200 dell'immagine
>> h=ones(1,7)/7;
>> y=conv(x,h);
>> figure(1);
>> subplot(2,1,1); plot(1:length(x),x);
>> subplot(2,1,2); plot(1:length(y),y);
>> axis([1 length(x) 0 255]);
```

Notate come il filtro tenda a smussare le discontinuità. Per applicare il filtraggio all'intera immagine bisogna filtrare sia lungo le righe che lungo le colonne, a tal fine si può usare il comando `conv2` nel modo seguente:

```
>> Y1=conv2(X,h);       % filtro l'immagine lungo le righe
>> h=h';                % vettore colonna con i coefficienti del filtro
>> Y=conv2(Y1,h);       % filtro l'immagine lungo le colonne
>> figure(2); image(X);  colormap(gray(256)); axis image;
>> figure(3); image(Y1); colormap(gray(256)); axis image;
>> figure(4); image(Y);  colormap(gray(256)); axis image;
```

Osservate con attenzione l'immagine prima di essere elaborata, $X$, l'immagine filtrata solo lungo le righe, $Y_1$, e l'immagine filtrata sia lungo le righe che lungo le colonne, $Y$. Cercate di individuare

le differenze tra queste tre immagini e spiegare perché è presente una specie di cornice nera nelle immagini filtrate.

Riscrivete il codice utilizzando la funzione `filter2`, facendo attenzione al fatto che la sintassi di questo comando richiede di dare in ingresso prima i coefficienti del filtro e poi il segnale da elaborare. Confrontando i risultati si può notare che l'effetto ai bordi è diminuito, perchè?

### 2.1 Esempi ed esperimenti proposti

a) *Esempio di applicazione.* Provate a filtrare l'immagine 'fax.jpg' con un filtro che realizza la media aritmetica su 2, 3 e 4 campioni e commentate il risultato.

b) *Filtri che enfatizzano le discontinuità.* Ripetiamo l'esperimento del filtraggio FIR sull'immagine 'lena.jpg' e sull'immagine 'impronta.jpg' usando i filtri con i seguenti coefficienti:

$h_1 = [1, -1], h_2 = [-1, 0, 1], h_3 = [-1, -2, 0, 1, 2]$.

Fate attenzione adesso alla visualizzazione dell'immagine filtrata che può assumere valori molto piccoli anche negativi, quindi è opportuno usare il comando `imagesc` che effettua un automatico riscalamento dei dati, in modo da usare al meglio la colormap (definita nel nostro caso su 256 livelli di grigio) e avere una migliore visualizzazione:

```
>> imagesc(X);
```

Come cambia il comportamento rispetto ai filtri visti prima?

c) *Ridimensionamento.* Realizziamo adesso l'operazione di decimazione di un'immagine e valutiamo il suo effetto al variare del fattore di decimazione che indicheremo con $a$:

```
>> [M,N]=size(X);
>> Y=X(1:a:M,1:a:N);
```

Visualizzando le immagini decimate si può notare che l'immagine viene rimpicciolita all'aumentare di $a$ e diminuisce quindi la sua risoluzione.

N.B. Ricordate di cancellare le variabili in memoria (`clear all`) e chiudere le figure (`close all`), prima di effettuare successive elaborazioni, per evitare di saturare la memoria a disposizione causando il blocco del programma.

## 3 Rappresentazione ed elaborazione di un segnale audio

Per i file audio a disposizione (con estensione .wav) potete invece usare la seguente sequenza di comandi per rappresentarne l'andamento nel dominio del tempo ed ascoltarli:

```
>> [x,fc,nbit] = wavread('nome_file');
>> n=[1:length(x)];
>> plot(n,x);
>> sound(x,fc);
```

I valori sono memorizzati nel vettore $x$ , la frequenza con cui l'audio è stato campionato è contenuta nella variabile $f_c$ e il numero di bit con cui è stato codificato si trova in $nbit$.

### 3.1 Esempi ed esperimenti proposti

a) *Filtraggio di un'interferenza sinusoidale.* Supponiamo di aggiungere al segnale vocale 'audio.wav' un'interferenza sinusoidale a frequenza $\nu_0 = 1/8$:

```
>> [i,fc]=wavread('Audio.wav');
>> i=i';
>> s=sin(pi*n/4);
>> sound(i,fc); pause;
>> sound(s);    pause;
>> x=i+s;
>> sound(x,fc); pause;
```

Utilizziamo adesso il seguente sistema MA pe rimuovere l'interferenza sinusoidale:

$$y(n) = x(n) - \sqrt{2}\, x(n-1) + x(n-2)$$

```
>> b=[1,-sqrt(2),1];
>> y=filter(b,1,x);
>> sound(y,fc);
```

Notate come nel segnale filtrato la fastidiosa interferenza sinusoidale sia stata quasi del tutto cancellata. Per comprenderne il motivo ponete in ingresso al filtro la sola interferenza sinusoidale vedrete che il segnale in uscita è nullo, infatti analiticamente risulta:

$$\begin{aligned} y(n) &= s(n) - \sqrt{2}\, s(n-1) + s(n-2) \\ &= \sin\left(\frac{\pi n}{4}\right) - \sqrt{2}\, \sin\left(\frac{\pi(n-1)}{4}\right) + \sin\left(\frac{\pi(n-2)}{4}\right) \\ &= \sin\left(\frac{\pi n}{4}\right) - \sqrt{2}\, \sin\left(\frac{\pi n}{4}\right) \frac{\sqrt{2}}{2} + \sqrt{2}\, \cos\left(\frac{\pi n}{4}\right) \frac{\sqrt{2}}{2} - \cos\left(\frac{\pi n}{4}\right) = 0 \end{aligned}$$

b) *Decimazione.* Provate ad ascoltare l'effetto della decimazione per 2 sul segnale 'Benigni_Troisi_progresso.wav'. Ciò che causa questa operazione somiglia molto ad ascoltare a velocità più veloce un segnale audio registrato su una cassetta.

c) *Generazione di echi.* Vediamo come realizzare in matlab un effetto audio digitale come l'*eco*. In fisica e in acustica l'eco è un fenomeno prodotto dalla riflessione di onde sonore contro un ostacolo che vengono a loro volta nuovamente percepite più o meno immutate (in termini di ampiezza del suono) e con un certo ritardo rispetto al suono diretto. Tale ritardo non deve essere inferiore ad 1/10 di secondo. Al di sotto di tale valore non si parla più di eco, ma di riverbero. In particolare si parla di eco quando si ha una singola riflessione dell'onda sonora percepita distintamente dall'ascoltatore. Da un punto di vista matematico:

$$y(n) = x(n) + g\, x(n-M)$$

dove $M$ è il ritardo e $g$ il guadagno (compreso tra 0 e 1) che permette di scalare l'ampiezza del segnale riflesso rispetto a quello diretto. Provate a generare un singolo eco dal file audio.wav, scegliete opportunamente il valore di $M$ e $g$ e ascoltate il risultato.