Laboratorio di Telecomunicazioni - a.a. 2010/2011
Lezione n. 2

Elaborazioni elementari dei segnali

L.Verdoliva

In questa seconda lezione realizziamo le operazioni fondamentali che si possono effettuare sui segnali a tempo discreto, sia quelle definite sull'asse dei tempi (traslazione, riflessione, decimazione e espansione) che quelle definite sulle ampiezze (cambiamento di scala, somma e prodotto).

1 Operazioni definite sulla variabile indipendente

Se i segnali sono rappresentati in Matlab da vettori riga, è possibile operare su di essi utilizzando le operazioni definite per i vettori riga. Dobbiamo però ricordare che ogni segnale è definito da *due* vettori, uno per i tempi e l'altro per le ampiezze. Questo costringe a qualche cautela. Supponiamo, ad esempio, di avere un segnale così definito:

```
>> nx = [-3:3]; x = [2,1,-1,0,1,4,3];
```

e di voler determinare il segnale traslato $y(n) = x(n-5)$. In questo caso l'elaborazione riguarda *solo* il vettore dei tempi, che va ritardato di 5 posti, effettuando la seguente operazione:

```
>> ny = nx + 5; y = x;
```

mentre con l'operazione:

```
>> ny = nx - 3; y = x;
```

si anticipa il segnale di 3 posti, ottenendo $y(n) = x(n+3)$. Rappresentate il segnale di partenza e quello ritardato (o anticipato) nella stessa finestra utilizzando il comando `subplot` nel modo seguente:

```
>> subplot(211); stem(nx,x);
>> subplot(212); stem(ny,y);
```

Costruite una funzione `trasla` che effettua la traslazione di un segnale a tempo discreto. Il prototipo di tale funzione potrebbe essere il seguente:

```
function [y,ny] = trasla(x,nx,n0)
```

1.1 Esempi ed esperimenti proposti

a) *Riflessione di un segnale.* Dato un segnale $x(n)$, è facile ottenere il segnale $y(n) = x(-n)$:

```
>> nx = [-5:3]; x = [0:8];
>> subplot(211); stem(nx,x); axis([-6 6 -1 9]);
>> ny = -nx; y = x;
>> subplot(212); stem(ny,y); axis([-6 6 -1 9]);
```

L'istruzione importante è `ny = -nx` che effettua il ribaltamento dell'asse temporale. Le altre istruzioni servono semplicemente per disegnare i due segnali nella stessa figura e allineare gli assi.

Osservate i valori delle due variabili `ny` e `y` e noterete che gli istanti temporali sono memorizzati in ordine decrescente, mentre il segnale è uguale a quello di partenza. Questo significa che se il segnale riflesso va ulteriormente elaborato è necessario procedere a ribaltare i valori dei due vettori usando il seguente comando:

```
>> ny = fliplr(ny); y = fliplr(y);
```

Sulla base di questo esempio, scrivete una funzione `rifletti` il cui prototipo è il seguente:

```
function [y,ny] = rifletti(x,nx)
```

b) *Decimazione/espansione.* Queste operazioni sono molto delicate da implementare, tuttavia è possibile usare la capacità di indicizzazione del Matlab e l'operazione di resto modulo M per produrre un codice semplice, veloce ed efficiente. Definiamo un segnale di prova per testare il codice e consideriamo:

```
>> x=[1:10]; nx=[-6:3];
>> stem(nx,x,'filled');
```

Ricordiamo che decimare un segnale significa realizzare una compressione, eliminando uno ogni M campioni: $y(n) = x(Mn)$. Cominciamo col modificare il vettore dei tempi per poter realizzare correttamente l'operazione (in fondo la decimazione è definita sulla variabile indipendente):

```
>> ny = nx/M;
```

per ottenere il nuovo vettore dei tempi. Poiché l'operazione di divisione genera un vettore dei tempi definito anche in valori non interi, essi vanno eliminati. Per fare ciò si può effettuare un controllo per scegliere solo i valori interi mediante l'operazione di divisione: se il resto della divisione per 1 è 0 il valore è intero, altrimenti no:

```
>> rem(ny,1)==0;
```

A questo punto è necessario selezionare gli indici corrispondenti mediante il comando `find` (questo comando è molto utile in Matlab, valuta se le operazioni logiche danno valore vero o falso e restituisce gli indici di quegli elementi che danno valore vero)

```
>> indici = find(rem(ny,1)==0)
```

e i valori relativi:

```
>> ny(indici)
```

che definiscono il nuovo vettore dei tempi. Equivalentemente si possono determinare i valori assunti dal segnale in corrispondenza a tali istanti temporali:

```
>> y = x(indici)
```

Adesso potete scrivere la funzione che realizza la decimazione con prototipo: `function [y,ny]=decima(x,nx,M)` e quella che realizza l'operazione di espansione: `function [y,ny] = espandi(x,nx,M)`. Provate le vostre funzioni per $M = 2, 3$ con segnale di ingresso $x(n) = \mathcal{B}_{12}(n + 6)$. Infine, applicate le due operazioni in cascata, e stabilite se il risultato è indipendente dall'ordine con cui effettuate le due operazioni.

2 Operazioni definite sulla variabile dipendente

Supponiamo di aver definito due segnali nel seguente modo

```
>> n1 = [-3:3]; x1 = [2,1,-1,0,1,4,3];
>> n2 = [-3:3]; x2 = [3,7,1,4,-1,10,2];
```

e di voler effettuare un cambiamento di scala sulle ampiezze, definendo i segnali $y_1(n) = 4\,x_1(n)$ e $y_2(n) = -3\,x_2(n)$, allora basta moltiplicare la costante per il vettore:

```
>> y1 = 4*x1; y2 = (-3)*x2;
```

Per realizzare la somma dei due segnali, poichè i valori dei tempi (`n1` ed `n2`) nei due casi sono gli stessi, basta porre:

```
>> n = n1;
>> y = x1 + x2;
```

Se invece i due segnali fossero definiti su insiemi temporali diversi, o avessero lunghezze differenti, la somma non si potrebbe fare così semplicemente, ma bisognerebbe in qualche modo "allineare" i segnali e renderli di uguale lunghezza prima di sommarli, altrimenti si rischia di eseguire l'operazione in modo non corretto. Quindi, se i due segnali non sono definiti sullo stesso insieme temporale, bisogna prima ridefinirli appropriatamente in modo che lo diventino. Nello specifico, va definito un nuovo asse temporale, `nz`, che comprenda le finestre temporali in cui sono definiti entrambi i segnali, e che quindi avrà come punto iniziale il minimo tra i due punti iniziali dei due vettori e quello finale pari al più grande punto finale dei due vettori:

```
>> nz = [min(min(nx),min(ny)): max(max(nx),max(ny))];
```

A questo punto i vettori da sommare vanno allineati, introducendo degli zeri fittizi. Ridefiniamo quindi due nuovi segnali `z1` e `z2`, definiti sull'intervallo esteso `nz`, che contengono i valori dei segnali da sommare:

```
>> z1 = zeros(1,length(nz));
>> z2 = zeros(1,length(nz))
>> indici1 = find((nz >= min(nx)) & (nz <= max(nx)));
>> indici2 = find((nz >= min(ny)) & (nz <= max(ny)));
>> z1(indici1) = x;
>> z2(indici2) = y;
```

L'operazione `find` ci consente di determinare gli indici dei due vettori in cui memorizzare i valori che poi andranno sommati:

```
>> z = z1 + z2;
```

Per comprendere bene queste operazioni visualizzate a video tutte le variabili create nel caso in cui vogliate sommare i due segnali $x_1(n) = \mathcal{R}_{10}(n)$ e $x_2(n) = n\mathcal{R}_6(n+6)$. Ovviamente questo accorgimento va preso tutte le volte in cui si vuole realizzare un'operazione di somma, differenza, prodotto o divisione tra segnali.

2.1 Esercizio

Un segnale tempo discreto è definito nel modo seguente:

$$x(n) = \begin{cases} 1 + \frac{n}{3} & -3 \leq n \leq -1 \\ 1 & 0 \leq n \leq 3 \\ 0 & \text{altrove} \end{cases}$$

dopo aver generato e rappresentato graficamente il segnale $x(n)$, generate e diagrammate i seguenti segnali:

1. $y(n) = x(4-n)$;
2. $y(n) = x(2n)x(-2-n)$;
3. $y(n) = x(n-1)\,\delta(n-3)$;
4. $y(n) = x(-n) + n\,x(n-2)$.