# Introduzione al Matlab

Soluzioni

## 1 Esempi di funzioni in matlab

a) Scrivete una funzione in cui, data una matrice di valori positivi, reali e distinti tra loro, e dimensioni $16 \times 16$, volete conservare solo i 4 valori più grandi e annullare tutti gli altri. Il prototipo della funzione deve essere il seguente: `function y=elabora(x)`, dove `x` è la matrice in ingresso e `y` quella in uscita.

Per scrivere una funzione che effettua questa operazione dobbiamo innanzitutto creare una matrice $16 \times 16$ di valori positivi reali distinti tra loro. Possiamo sicuramente fornire manualmente i singoli valori della matrice oppure crearla con la funzione `rand`:

```
>> x = rand(16);    % crea una matrice di valori casuali compresi tra 0 e 1
>> x = x*40;        % i valori sono numeri reali compresi tra 0 e 40
```

A questo punto abbiamo diversi modi per conservare i 4 valori più grandi e annullare tutti gli altri. Cominciamo con la soluzione più lenta per poi passare a quella più efficiente. Facciamo uso della funzione `sort` di Matlab che realizza l'ordinamento di un vettore e quindi ci permette di individuare la soglia al di sotto della quale i coefficienti devono essere azzerati: Una possibile soluzione che fa uso di due cicli for è la seguente:

```
function y=elabora(x)
% ELABORA conserva solo i 4 valori più grandi di una matrice
% che contiene valori positivi reali e distinti
% x: matrice di ingresso, y: matrice di uscita

z  = sort(x(:),'descend');     % ordina in senso decrescente
soglia = z(4);                 % calcolo della soglia

for i=1:16,
   for j=1:16,
      if(x(i,j) < soglia)
         x(i,j) = 0;
      end
   end
end
y = x;
```

Questo codice può essere scritto in maniera più efficiente, dato che i cicli for contengono un test logico, che può essere convertito in un'operazione matriciale. In particolare si possono adottare due soluzioni: una in cui si fa uso dell'operatore `find`, che determina la lista degli indici in un vettore per cui una determinata condizione è verificata, l'altra attraverso l'uso di una maschera.

Se digitiamo il comando `indici = find(x < soglia)` otterremo la lista degli indici della matrice relativi ai valori che sono più piccoli della soglia. In quelle posizioni bisogna annullare tutti i valori della matrice, quindi il codice si riscrive:

```
indici = find(x < soglia);
x(indici) = 0;
y = x;
```

In alternativa è possibile creare una matrice di test (maschera) i cui elementi valgono 1 quando gli elementi di `x` sono superiori alla soglia, 0 altrimenti, usando il seguente comando `mask=x>soglia`. Basta poi moltiplicare ogni elemento di `x` con la maschera `mask` per ottenere la matrice cercata:

```
mask = (x >= soglia)
y  = x.*mask;
```

b) Scrivete una funzione che effettua l'operazione di saturazione dei dati (amplificatore reale), vale a dire se i dati contenuti in un vettore superano un certo valore limite (in modulo) essi vengono resi uguali proprio a tale valore. Il prototipo della funzione deve essere il seguente: `function y=clip(x, Limit)`, dove `x` è il vettore in ingresso, `Limit` è il valore limite e `y` il vettore in uscita. Per verificare la correttezza della funzione, scrivete uno script dal nome exe.m in cui definite il vettore di ingresso: $[-10\ \ 3\ -6\ \ 0\ \ 1\ -2\ \ 3\ \ 4\ -15\ \ 3\ \ 21]$, e il valore limite (in modulo) pari a $8$, quindi chiamate le funzione `clip` e visualizzate il risultato.

Anche in questo caso ci sono diverse soluzioni più o meno efficienti, la prima è la seguente:

```
function y=clip(x, Limit)
% CLIP satura i valori di x(n) al valore limite
% x: vettore di ingresso, Limit: valore di saturazione, y: vettore di uscita
N=size(x);
for n=1:N,
   if(abs(x(n)) > Limit)
      x(n)=sign(x(n))*Limit;  % satura i valori conservando il segno
   end
end
y=x;
```

La versione che fa uso della funzione `find` è:

```
indici = find(abs(x)>Limit);
x(indici) = sign(x(indici))*Limit;
y = x;
```

Quella invece che utilizza la maschera è:

```
y=Limit*(x > Limit) - Limit*(x < -Limit) + x.*(abs(x) <= Limit);
```

In quest'ultimo caso l'espressione presente al secondo membro deve essere interpretata una sola volta a differenza del caso in cui è presente il ciclo for in cui le tre espressioni presenti devono essere interpretate ben $N$ volte. Per verificare l'efficienza della funzione potete valutare la differenza in tempo di esecuzione usando i comandi `tic` e `toc`, scritti all'inizio e alla fine del codice, rispettivamente.

Scriviamo infine lo script exe.m che richiama la funzione clip:

```
clear all; close all; clc;
x = [-10 3 -6 0 1 -2 3 4 -15 3 21];
Limit = 8;
tic
y = clip(x,Limit)
toc
```

A questo punto basta digitare `exe` dal prompt dei comandi per eseguire il programma.