



 POLITECNICO DI MILANO

Dipartimento di
Elettronica e Informazione

Sesta esercitazione

Riccardo Cattaneo

Alessandro A. Nacci



POLITECNICO
DI MILANO

13/11/2013



Quinta esercitazione: agenda

- (20') Blocco 1: MATLAB base
- (40') Blocco 2: MATLAB base++
- (60') Blocco 3: MATLAB base²

Blocco 1 / 1 (5')

- 1) Scrivere un programma che inizializza un vettore monodimensionale di interi
- 2) Stampare il terzo elemento dell'array a
- 3) Copiare il vettore in un altro vettore della stessa dimensione. salvare la dimensione dell'array b in una variabile
- 4) Aggiungere un elemento all'array copia in coda a tutti gli elementi
- 5) Copiare l'array a in un array a_trasposto. Anche b
- 6) Concatenare ed impilare i due vettori rispettivamente
- 7) Dichiarare un vettore che contiene tutti i numeri da 1 a 100
- 8) Dichiarare un vettore che contiene tutti i numeri pari da 1 a 100
- 9) Dichiarare un vettore che contiene tutti i numeri dispari da 1 a 100

Blocco 1 / 1 (5')

```
% Scrivere un programma che inizializza un vettore monodimensionale di interi
a = [22 33 44 55 66 77 88 99]
% Stampare il terzo elemento dell'array a
a(3)

%copiare il vettore in un altro vettore della stessa dimensione.
b= a

%salvare la dimensione dell'array b in una variabile
dim = size(b) % dim = [1 8]

                % in alternative è possibile usare la funzione
                % 'length' che restituisce la dimensione
                % più grande di una matrice
% Aggiungere un elemento all'array copia in coda a tutti gli elementi
b(dim(2) + 1) = 10
% Copiare l'array a in un array a_trasposto. Anche b
a_trasposto = a'
b_trasposto = b'
% Concatenare ed impilare i due vettori rispettivamente
c = [a,b]
c_trasposto = [a_trasposto;b_trasposto]
% Dichiarare un vettore che contiene tutti i numeri da 1 a 100
cento = 1:100

% Dichiarare un vettore che contiene tutti i numeri pari da 1 a 100
pari = 2:2:100
% Dichiarare un vettore che contiene tutti i numeri dispari da 1 a 100
dispari = 1:2:100
```

Blocco 1 / 2 (10')

- 1) Dichiarare ed inizializzare una matrice 4x3 di interi
- 2) Sommare uno a tutti gli elementi della matrice
- 3) Fare la matrice trasposta
- 4) Aggiungere una riga a vostro piacere alla matrice
- 5) Creare una matrice 4x4 che ha tutti gli elementi a 0
% tranne quelli della diagonale che devono essere
% quelli di a
- 6) Estrarre la diagonale della matrice a e salvarla
% in un vettore riga
- 7) Trovare il massimo tra i numeri inseriti nella matrice
% nota: max, applicato ad una matrice, restituisce
% la riga che contiene il valore massimo
- 8) Trovare il massimo tra i numeri della diagonale
- 9) Trovare il minimo della matrice
- 10) Se il massimo della matrice è contenuto sulla diagonale
% portare la diagonale della matrice ad avere
% tutti i valori pari al minimo

Blocco 1 / 2 (10')

```
% Dichiarare ed inizializzare una matrice 4x3 di interi
a = [10 20 30 40; 50 60 70 80; 90 100 110 120]
% Sommare uno a tutti gli elementi della matrice
uno = ones(3,4)
b = a + uno
% Fare la matrice traposto
a_T = a'

% Aggiungere una riga alla matrice
a = [a; 3 5 6 7]

% Creare una matrice 4x4 che ha tutti gli elementi a 0
% tranne quelli della diagonale che devono essere
% quelli di a
I = eye(4)
a_2 = a .* I
% Estrarre la diagonale della matrice a e salvarla
% in un vettore riga
diagonale = diag(a)'
```

Blocco 1 / 2 (10')

```
% Trovare il massimo tra i numeri inseriti nella matrice % nota: max,  
applicato ad una matrice, restituisce  
% la riga che contiene il valore massimo massimo_matrice = max(max(a))  
  
% Trovare il massimo tra i numeri della diagonale  
massimo_diagonale = max(diagonale)  
% Trovare il minimo della matrice  
minimo_matrice = min(min(a))  
% Se il massimo della matrice è contenuto sulla diagonale  
% portare la diagonale della matrice ad avere  
% tutti i valori pari al minimo  
if massimo_matrice == massimo_diagonale  
    minimi_diag = I .* minimo_matrice  
    filtro = ones(4) - eye(4)  
    a = a .* filtro + minimi_diag  
end
```

Blocco 1 / 3 (5')

```
% chiedere all'utente di inserire un vettore e un numero
%
% calcolare:
% # il numero di elementi del vettore uguali al numero insierito
% # il numero di elementi del vettore maggiori del numero insierito
% # il numero di elementi del vettore minori del numero insierito
%
% indicare poi il valore di tali elementi,
% la loro posizione del vettore
% il vettore binario per ogni operazione richiesta
```


Blocco 1 / 3 (5')

```
%x = input('inserire un numero')
%vettore = input('inserisci un vettore')
x = 10
vettore = [10 20 30 50 60]
%vettore binario
Buguali= vettore==x
Bmaggiori= vettore>x
Bminori= vettore<x
%valore degli elementi
Vuguali= vettore(vettore==x)
Vmaggiori= vettore(vettore>x)
Vminori= vettore(vettore<x)
%numero di elementi
Nuguali=size( vettore (vettore==x),2)
Nmaggiori=size( vettore (vettore>x),2)
Nminori=size( vettore (vettore<x),2)
%alternativa:
% Nmaggiori = sum(vettore > x)
%posizione degli elementi
Puguali= find(vettore==x)
Pmaggiori= find(vettore>x)
Pminori= find(vettore<x)
```

Quinta esercitazione: agenda

- (20') Blocco 1: MATLAB base
- (40') Blocco 2: MATLAB base++
- (60') Blocco 3: MATLAB base²

Blocco 2 / 1 (10')

- % Gioco dei dadi: due giocatori tirano 50 volte due dadi.
- % Ad ogni manche vince il giocatore con il numero piu alto
- % Alla fine vince chi ha vinto più manche
- % in questa configurazione la rand restituisce una colonna

Blocco 2 / 1 (10')

```
% Gioco dei dadi: due giocatori tirano 50 volte due dadi.  
% Ad ogni manche vince il giocatore con il numero piu alto  
% Alla fine vince chi ha vinto più manche  
% in questa configurazione la rand restituisce una colonna  
giocatore_1 = ceil(rand(50,1) * 6)'  
giocatore_2 = ceil(rand(50,1) * 6)'  
vincite_arr_1 = giocatore_1 > giocatore_2  
vincite_arr_2 = giocatore_2 > giocatore_1  
vincite_1 = sum(vincite_arr_1)  
vincite_2 = sum(vincite_arr_2)  
if vincite_1 > vincite_2  
    vincitore = 1  
else  
  
end  
  
%rand(50,1) => colonna 50 elementi e | 0 <= e <= 1
```

Blocco 2 / 2 (10')

```
% Trovare graficamente i vari punti di intersezione tra  
%  $y = f(x) = -5x + 10$   
%  $y = f(x) = 3x^2 + 2x + 3$   
%  $y = f(x) = \log(x/2)$ 
```

Blocco 2 / 2 (10')

```
% Trovare graficamente i vari punti di intersezione tra
%  $y = f(x) = -5x + 10$ 
%  $y = f(x) = 3x^2 + 2x + 3$ 
%  $y = f(x) = \log(x/2)$ 
x = 1:0.1:100

%prima equazione
eq1_1 = x .* -5
eq1 = eq1_1 + 10
%seconda equazione
eq2_1 = x .^ 2
eq2_2 = eq2_1 .* 3
eq2_3 = x .* 2
eq2 = eq2_2 + eq2_3 + 3
%terza equazione
eq3_1 = x ./ 2
eq3 = log(eq3_1)
plot(x, eq1)
hold on
plot(x, eq2)
plot(x, eq3)
hold off
```

Blocco 2 / 3 (20')

```
% ordine un array di n elementi  
% facendo uso delle istruzioni messe a disposizione  
% da matlab
```

Algoritmo di ordinamento: dato un array "disordinato" ed un array "ordinato", trova il valore minimo nell'array "disordinato" e mettilo nella prima posizione libera dell'array "ordinato", quindi rimuovilo dall'array "disordinato". Itera fino a che "disordinato" é lungo 0.

Blocco 2 / 3 (20')

```
% ordine un array di n elementi
% facendo uso delle istruzioni messe a disposizione
% da matlab
a = [4 6 9 2 5]

curr = min(a)
pos = find(a == curr) % così stampa anche la posizione
filtro_n = a==curr % maschera
filtro = ones(1,length(a)) - filtro_n % maschera "negata"
a1 = a(logical(filtro)) % tutti gli elementi meno il minimo
a_ord = curr

while length(a1) > 0
    curr = min(a1)
    pos = find(a1 == curr)
    filtro_n = a1==curr
    filtro = ones(1,length(a1)) - filtro_n % nega quello prima
    a1 = a1(logical(filtro))
    a_ord = [a_ord; curr]
end

a_ord
```


Quinta esercitazione: agenda

- (20') Blocco 1: MATLAB base
- (40') Blocco 2: MATLAB base++
- (60') Blocco 3: MATLAB base²

Blocco 3 / 1 (10')

```
% Dato un vettore in ingresso ed un numero a (entrambi float)
% controllare quante occorrenze del numero sono presenti
% nel vettore.
% In seguito elencare i numeri diversi da a presenti nel vettore
```

Blocco 3 / 1 (10')

```
vett = input('Inserisci un vettore ')\na = input('inserisci un numero con cui effettuare il confronto: ')\n%%\n%% Ricerca dei numeri uguali\n%% SONO NUMERI IN VIRGOLA MOBILE! Usare l'operatore '==' può\n%% portare ad errori dovuti alle approssimazioni!\nfprintf("\n\n\nIl numero di occorrenze del numero %f sono ", a)\ndisp(sum(abs(vett - a) <= eps))\nfprintf("I numeri diversi da %f sono:\n", a)\ndisp(vett(vett ~= a))\n%%\n%% Ricerca dei numeri maggiori\nfprintf("\nI numeri maggiori di %f sono ", a)\ndisp(sum(vett > a))\nfprintf("e in particolare sono:\n")\ndisp(vett(vett > a))
```

Blocco 3 / 2 (15')

```
% Se l'array inserito è numerico, effettuare i seguenti controlli
% Verificare se tutti i numeri sono positivi
% Verificare se esiste un numero negativo
% Applicare la radice quadrata a tutti i valori: ci sono dei
% valori complessi?
% Verificare se tutti i numeri sono pari
% Trovare gli indici dei numeri pari
% Verificare se esiste un numero dispari
    % Contare i numeri dispari se esistono
    % Dire in che posizioni sono
```

Blocco 3 / 2 (15')

```
vett = input('Inserisci un vettore ')\nif isnumeric(vett)\n    %numeri positivi\n    if all(vett > 0),\n        disp('tutti i numeri sono positivi')\n    else\n        disp('non tutti i numeri sono positivi')\n    end\n\n    %esistenza numero negativo\n    if any(vett < 0),\n        disp('esiste un numero negativo')\n    else\n        disp('non esiste alcun numero negativo')\n    end\n\n    %radice quadrata\n    sqrt_vett = sqrt(vett)\n    if isreal(sqrt_vett),\n        disp('non esistono valori complessi')\n        disp('esistono valori complessi')\n    end
```

Blocco 3 / 2 (15')

```
%numeri pari
carry_vett = mod(vett,2)
if(all(carry_vett==0))
    disp('tutti i numeri sono pari')
else
    disp('non tutti i numeri sono pari')
end
%indici dei numeri pari
pos_pari = find(1-carry_vett)
%numeri dispari
if any(carry_vett)
    disp('esiste almeno un numero dispari')
    pos_dispari = find(carry_vett),
    disp('i numeri dispari sono in posizione')

    disp(pos_dispari)
else
    disp('non esistono numeri dispari')
end
else % dell'"if isnumeric(...)"
    disp('Devi inserire un array numerico')
end
```

Blocco 3 / 3 (10')

```
% DISEGNARE LA TAVOLA PITAGORICA di dimensione nxn
```

```
%
```

```
% 1 2 3 4 5
```

```
% 2 4 6 8 10
```

```
% 3 6 9 12 15
```

```
% 4 8 12 16 20
```

```
% 5 10 15 20 25 %
```

```
%
```

```
% Per risolvere quest'esercizio, è sufficiente osservare la struttura della  
% tavola pitagorica: la prima riga è un vettore di numeri naturali, da 1 a  
% 10, così come la prima colonna; mentre l'elemento (i,j)-esimo è ottenuto  
% moltiplicando l'elemento j-esimo della prima riga e i-esimo della prima  
% colonna.
```

Blocco 3 / 3 (10')

```
% Ecco che è possibile vedere la colonna come la trasposta della prima riga,  
% e gli altri valori sono ottenibili effettuando una moltiplicazione tra  
% vettori  
n = input('Inserire la dimensione della tavola pitagorica')  
%Determino la riga della tavola pitagorica  
a=1:n  
%Calcolo (e visualizzazione) della tavola periodica  
a'*a
```


Blocco 3 / 4 (10')

```
% Progettare uno script matlab che chieda all'utente quale tra le  
% seguenti operazioni si vuole effettuare su un generico vettore  
%     MEDIA MASSIMO MINIMO
```

Blocco 3 / 4 (10')

```
vettore = []
```

%Creazione menù: per prima cosa occorre che l'utente sia a conoscenza dell'esistenza di un menù e delle opzioni di scelta; per far ciò è sufficiente visualizzare delle stringe sul command window che indichino le opzioni disponibili

```
disp('selezionare un azione');  
disp('1: calcolo della media');  
disp('2: calcolo del massimo');  
disp('3: calcolo del minimo');  
disp('4: esci');
```

```
scelta=input('scelta: ');
```

%Le righe di codice precedenti hanno la SOLA funzione di visualizzare il menù. Per far sì che sia realmente un menu, occorre valutare con un test il contenuto della scelta effettuata (variabile scelta) A tal scopo, l'uso della struttura switch/case è consigliabile, in quando si hanno un numero di opzioni finite e note a priori

```
switch scelta  
    case 1  
        out=mean(vettore);  
    case 2  
        out=max(vettore);  
    case 3  
        out=min(vettore);  
    case 4  
end
```

%in questo caso, uscire corrisponde a non eseguire nessuna operazione

```
disp('esco')
```

Blocco 3 / 5 (15')

```
% Quanto guadagna ogni operaio?  
% Qual è il salario totale di tutti gli operai?  
% Quanti pezzi vengono prodotti?  
% Qual è il costo medio di un pezzo?  
% Quante ore occorrono in media per un pezzo?  
% Qual è l'operaio più efficiente?
```

Dati iniziali (operaio i-esimo, posizione i-esima)

```
pagaoraria=[5 5.5 6.5 5 6.25]  
oresett=[40 43 37 50 45]  
pezzi=[1000 1100 1000 1200 1100]
```

Blocco 3 / 5 (15')

```
% quanto guadagna ogni operaio?
% per rispondere a questa domanda, è sufficiente effettuare una
% moltiplicazione elemento per elemento di vettori paga oraria e oresett
paghe=pagaoraria.*oresett
% Qual'è il salario totale di tutti gli operai?
% Si chiede in pratica di sommare gli elementi della riga 'paghe', che per
% noi è un vettore. è quindi possibile usare direttamente la funzione
' sum',
% che effettua la somma di tutti gli elementi nel vettore in argomento
salariototale=sum(paghe)
% Quanti pezzi vengono prodotti?
% In analogia a quanto detto, si riutilizza la funzione 'sum'
pezzitotale=sum(pezzi)
```

Blocco 3 / 5 (15')

```
% Qual'è il costo medio di un prezzo?  
% Il costo medio si ricava considerando il numero di pezzi prodotti in  
totale  
% ed il costo totale di produzione (costo operai). Al pezzo, quindi si ha:  
% costo pezzo= salariototale/pezzitotale, valori già calcolati con le  
% domande precedenti  
costopezzo=salariototale/pezzitotale  
% Quante ore occorrono in media per pezzo?  
% Considerazioni analoghe al precedente, con la differenza che occorre  
% conoscere le ore totali, fin'ora non calcolate, valore ottenibile  
% facilmente con la funzione 'sum'  
oreperpezzo=sum(oresett)/pezzitotale
```

Blocco 3 / 5 (15')

```
% Qual'è l'operaio più efficiente?
% Per rispondere a questa domanda si crea un vettore efficienza, dove ogni
% elemento valuta l'efficienza del singolo operaio. L'efficienza è definita
% come il rapporto tra pezzi prodotti dall'operaio e dalle ore totali
% impiegate per produrre quei pezzi; occorre pertanto un'operazione di
% divisione elemento per elemento tra i vettori 'pezzi' e 'oresett'
efficienza=pezzi./oresett
% Ora che sono state calcolate tutte le efficienze dei singoli operai,
% occorre individuare l'operaio migliore, ovvero colui che ha efficienza più
% elevata. L'operazione si riduce a ricavare il valore massimo contenuto in
% un vettore (efficienza) e l'indice corrispondente, che individua
% l'operaio. A tal scopo si utilizza la funzione 'max'. help max per
% ulteriori informazioni
% L'operazione max funziona in questo modo:
% [valore_massimo, posizione_valore_massimo] = max(vettore_in_input)
[efficienzamax,chi]=max(efficienza)
```

Succo del discorso, fino ad ora :)

In MATLAB si ragiona con un paradigma diverso dal C:

- le funzioni operano in parallelo sui dati, contenuti in forma array multidimensionale nelle variabili (gli scalari essendo un caso - molto - particolare di array multidimensionale)
- le maschere tornano molto comode quando dovete filtrare in qualche modo i dati
- le operazioni matriciali sono esprimibili con poche istruzioni (al limite una sola istruzione)
- i for sono quasi sempre da evitare