Funzioni e Operazioni Matriciali

Informatica, AA 2021/2022

Francesco Trovò

5 Novembre 2021

https://trovo.faculty.polimi.it/

francesco1.trovo@polimi.it

Array Vuoto

Un array vuoto si definisce così:

```
nomeVettore = []
```

Può essere una forma di dichiarazione di una variabile

```
>> a = []
a =
    []

>> whos a
  Name      Size            Bytes  Class     Attributes
  a         0x0                 0  double
```

Cancellare Parti di un Vettore

Quando si assegna il valore [] ad un elemento di un vettore, il corrispondente elemento viene rimosso e il vettore ridimensionato: non si crea un 'buco'

```
>> a = 1:5
a =
     1     2     3     4     5
>> whos a
Name  Size Bytes  Class    Attributes
  a    1x5 40     double
>> a(3) = []
a =
     1     2     4     5
>> whos a
Name  Size Bytes  Class    Attributes
  a    1x4  32    double
```

Cancellare Parti di una Matrice

L'array vuoto [] non è assegnabile a singoli elementi di matrici (non si possono "creare buchi")

```
>> m(1:3, 1:3) = 1
m =
     1     1     1
     1     1     1
     1     1     1
>> m(2, : ) = 5
m =
     1     1     1
     5     5     5
     1     1     1
>> m(2, 3)=[]
??? Subscripted assignment dimension
mismatch.
```

Cancellare Parti di una Matrice

È però assegnabile a intere righe o colonne di matrici, che vengono cancellate (ricompattando la matrice)

```
>> m(:, 2) = []
m =

     1     1
     5     5
     1     1
>> whos m
  Name        Size    Bytes Class    Attributes
  m           3x2        48 double
```

Esercizio: Cancellare Parti di una Matrice

- Creare la tabella pitagorica (fino alla tabellina del 10)
- Rimuovere la riga e la colonna della tabellina del 5
- Dopo averle rimosse rimettere nella stessa posizione una riga ed una colonna di zeri

Soluzione

```
F1 = [1 : 10];
F1 = [F1; F1; F1; F1; F1];
F1 = [F1; F1];
F2 = F1';
T = F1 .* F2;

% tolgo tabellina del 5 su righe e colonne
T(:, 5) = [];
T(5, :) = [];

% "collage", rimetto gli zeri
colZeri = zeros(9, 1);
Z = [T(:, 1 : 4), colZeri, T(:, 5 : 9)];

rigaZeri = zeros(1, 10);
Z = [Z(1 : 4, :); rigaZeri; Z(5 : 9, :)];
```

Memorizzazione degli Array

Gli array vengono salvati linearmente in memoria.

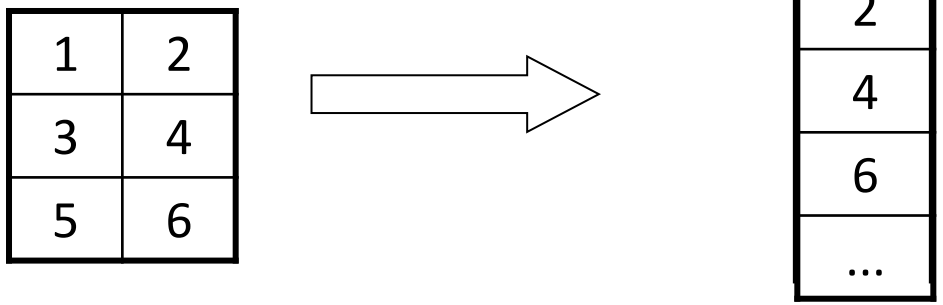
In particolare le matrici sono memorizzate

- per colonna: colonna 1, poi colonna 2, 3, etc.
- ogni colonna memorizzata per indici di riga crescenti

Array memorizzati in forma lineare nella RAM variando

- più velocemente i primi indici
- più lentamente quelli successivi

NB: opposto a quanto avviene in C



Array: forma *linearizzata*

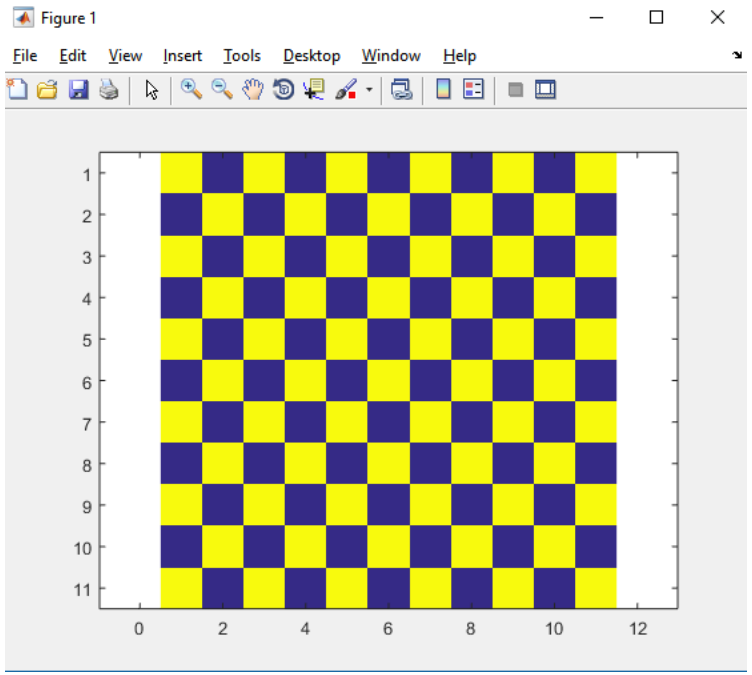
Si può accedere a un array a più dimensioni come se ne avesse una sola

Usando un unico indice si segue l'ordine della memorizzazione

```
>> a = [1 2 3; 4 5 6; 7 8 9; 10 11 12]
a =
     1     2     3
     4     5     6
     7     8     9
    10    11    12
>> a(3, 2)
ans =
     8
>> a(10)
ans =
     6
```

Esercizio

Scrivere una funzione che genera una matrice quadrata $n \times n$ di 0 ed 1 disposti «a scacchiera». La funzione controlla anche che n sia dispari




```
function A = scacchiera(n)
A = [];

if mod(n,2) == 1
    A = zeros(n);
    A(1 : 2 : end) = 1;
end
```

```
>> figure, imagesc(scacchiera(11)),
axis equal
```

Tipo di Dato Logico

Tipo di Dato Logico

È un tipo di dato che può avere solo due valori

- true (vero) 1
- false (falso) 0

I valori di questo tipo possono essere generati

- direttamente da due funzioni speciali (true e false)
- dagli operatori relazionali
- dagli operatori logici

I valori logici occupano un solo byte di memoria (i numeri ne occupano 8)

Esempi

```
>> a = true;
>> whos a
 Name   Size    Bytes  Class     Attributes
  a     1x1      1     logical
```

a è un vettore 1x1 che occupa 1 byte e appartiene alla classe “tipo logico”

```
>> a = 1>7
a =
     0
```

Operatori Relazionali e Logici

Operatori Relazionali

Operano su tipi numerici o stringhe

Possono essere usati per confrontare

- due scalari
- due vettori aventi la stessa dimensione

Forma generale: a OP b

- a,b possono essere espressioni aritmetiche, variabili, stringhe (della stessa dimensione)
- OP: `==, ~=, >, >=, <, <=`

Esempi:

- `3 < 4` true(1)
- `3 == 4` false(0)
- `'A' < 'B'` true(1)

Note

Come in C: non confondere == e =

- == è un operatore di confronto
- = è un operatore di assegnamento

La precisione finita può produrre errori con == e ~ =

- sin(0) == 0 → 1
- sin(pi) == 0 → 0
- eppure logicamente sono vere entrambe!!

Per i numeri piccoli conviene usare una soglia

- abs(sin(pi) - 0) <= eps

Vettori e stringhe

Gli operatori relazionali tra vettori vengono applicati in maniera **puntuale**

Il risultato di un confronto tra v1 e v2 è un vettore v3 di tipo boolean, aventi le stesse dimensioni di v1 (e v2)

$$\text{v3 = (v1 >= v2);} \quad v3(i) = \begin{cases} 1, & \text{se v1(i)} \geq \text{v2(i)} \\ 0, & \text{se v1(i)} < \text{v2(i)} \end{cases}$$

Esempi:

>> [1 0; -2 1] < 0 [false false; true false] ([0 0; 1 0])

>> [1 0; -2 1] >= [2 -1; 0 0] [false true; false true]

Si possono confrontare stringhe di lunghezza uguale

```
>> 'pippo'=='pluto'
 ans = [1 0 0 0 1]
```

Operatori Logici: Forma Generale

Operatori binari: **AND** (**&&**, oppure **&**), **OR** (**||**, oppure **|**), **XOR** (**xor**):

a OP1 b per la notazione simbolica
OP(a,b) per la notazione testuale

Operatori unari: NOT (~):

OP2 a

a,b possono essere variabili, costanti, espressioni da valutare, scalari o vettori (dimensioni compatibili)

Valori numerici di a, b vengono interpretati come logici:

- 0 come falso
- tutti i numeri diversi da 0 come vero

Richiamo, Tabelle di Verità

a	b	a && b	a || b	~a	`xor`(a, b)
0	0	0	0	1	0
0	1	0	1	1	1
1	0	0	1	0	1
1	1	1	1	0	0

Or esclusivo: vero quando è vera solo uno delle due espressioni coinvolte
a XOR b == a OR b AND (~(a AND b))

&& vs & e || vs |

`&&` (**`||`**) **funziona con gli scalari** e valuta prima l'operando più a sinistra. Se questo è sufficiente per decidere il valore di verità dell'espressione non va oltre

- **`a && b`**: se **`a`** è falso non valuta **`b`**
- **`a || b`**: se **`a`** è vero non valuta **`b`**

`&` (**`|`**) **funziona con scalari e vettori** e valuta **tutti** gli operandi prima di valutare l'espressione complessiva

Esempio: **`a / b > 10`**

- se **`b`** è 0 non voglio eseguire la divisione
- **`(b~=0)&(a/b>10)`** è la soluzione corretta: **`&&`** controlla prima **`b~=0`** e se questo è falso non valuta il secondo termine. Invece **`(b~=0)&(a/b>10)`** porterebbe ad una divisione per 0 quando **`b == 0`**

Esempi

"Hai tra 25 e 30 anni?"

```
(eta >= 25) & (eta <= 30)
```

Con i vettori:

```
Voto = [12, 15, 8, 29, 23, 24, 27]
 C = (Voto > 22) & (Voto < 25)
-> C = [0 0 0 0 1 1 0]
 D = (mod(Voto,2) == 0) | (Voto > 18)
-> D = [1 0 1 1 1 1 1]
 E = xor((mod(Voto,2)==0), (Voto>18))
-> E = [1 0 1 1 1 0 1]
```

Utile per contare quanti elementi soddisfano una condizione

```
nVoti= sum (Voto > 22 & Voto < 25)
```

Precedenze tra gli Operatori

Ogni espressione logica viene valutata rispettando il seguente ordine:

- operatori aritmetici
- operatori relazionali da sinistra verso destra
- NOT (**~**)
- AND (**&** e **&&**) da sinistra verso destra
- OR (**|** e **||**) e XOR da sinistra verso destra

Indicizzazione con i Logici

Vettori Logici per Selezionare Sottovettori

I vettori logici possono essere usati per selezionare gli elementi di un array al posto di un vettore di indici

`nomeVettore(vettoreLogico)`

- vengono estratti gli elementi di **`nomeVettore`** alle posizioni per cui **`vettoreLogico`** vale 1

Per esempio

```
>> x = [6,3,9]; y = [14,2,9];
>> b = x<=y ; % b = 1     0     1
>> z = x(b)
z =
     6     9
```

Esempi

Inizializzare a con i numeri da -10 a 20 con passo 3

```
>> a = [-10 : 3 : 20]
```

Visualizzare solamente i numeri maggiori di 10

```
>> a(a > 10);
```

Portare a zero tutti gli elementi negativi

```
>> a(a < 0) = 0;
```

Sommare 10 ai numeri minori di 10

```
>> a(a < 10) = a(a < 10) + 10;
```

Cambiare il segno a tutte le occorrenze di -7 o 17

```
>> a(a == -7 | a == 17) = -a(a == -7 | a == 17);
```

NB qui non si può usare ||

Note

`nomeVettore` e **`vettoreLogico`** devono avere la **stessa dimensione**

Per **creare un vettore logico non basta** creare un **vettore di 0 e 1** (numeri), bisogna convertirlo con la funzione logical

```
>> ii = [1,0,0,0,1];
>> jj = (ii == 1); %oppure jj = logical(ii)
>> A = [1 2 3 4 5];
>> A(jj) → [1 5]
>> A(ii) → Subscript indices must either be real positive
     integers or logicals.
```

Una nota sul costrutto if

`espressione1` può coinvolgere vettori:

- in tal caso **`espressione1`** è vera solo se tutti gli elementi di **`espressione1`** sono non nulli

Esempio

```
v = input('inserire vettore: ');
if (v >= 0)
    disp([num2str(v),' tutti pos. o nulli']);
elseif(v<0)
    disp([num2str(v),' tutti negativi']);
else
    disp([num2str(v),' sia pos. che neg.']);
end
```

Funzioni Logiche

Nome	Elemento restituito
all(x)	un vettore riga, con lo stesso numero di colonne della matrice x, che contiene 1, se la corrispondente colonna di x contiene tutti elementi non nulli, o 0 altrimenti. Se x è un vettore restituisce 0 o 1 con lo stesso criterio.
any(x)	un vettore riga, con lo stesso numero di colonne della matarice x, che contiene 1, se la corrispondente colonna di x contiene almeno un elemento non nullo, o 0 altrimenti. Se x è un vettore restituisce 0 o 1 con lo stesso criterio.
isinf(x)	un array delle stesse dimensioni di x con 1 dove gli elementi di x sono 'inf', 0 altrove
isempty(x)	1 se x è vuoto, 0 altrimenti
isnan(x)	un array delle stesse dimensioni di x con 1 dove gli elementi di x sono 'NaN', 0 altrove
finite(x)	un array delle stesse dimensioni di x, con 1 dove gli elementi di x sono finiti, 0 altrove
ischar(x)	1 se x è di tipo char, 0 altrimenti
isnumeric(x)	1 se x è di tipo double, 0 altrimenti
isreal(x)	1 se x ha solo elementi con parte immaginaria nulla, 0 altrimenti

Altre Funzioni Logiche: find

`indx = find(x)` restituisce gli indici degli elementi non nulli dell'array **x**. x può essere un'espressione logica.

Esempio

```
a = [5 6 7 2 10]
find(a>5) -> ans = 2 3 5
```

Nota: **find restituisce** gli **indici** e **non estrae un sottovettore** (come invece posso fare utilizzando vettori di interi o vettori logici come indici di un vettore)

```
x = [5, -3, 0, 0, 8];
y = [2, 4, 0, 5, 7];
values = y(x&y) -> values = [2 4 7]
indexes = find(x&y) -> indexes = [1 2 5]
```

Esercizio

```
anni = [19 20 21 18 18 20]; voti = [18 21 18 27 22 24];

%% definizione del vettore logical

maggiori25 = voti > 25 * ones(1, 6) % confronto tra vettori
della stessa dimensione

maggiori25 = voti > 25; % possibile anche confronto con
scalari

%% funzione sum si può applicare al vettore logical

nMaggiori25 = sum(maggiori25)

nMaggiori25 = sum(voti > 25)

n18 = sum(voti == 18)
```

Esercizio

```
anni = [19 20 21 18 18 20]; voti = [18 21 18 27 22 24];

%% operatori logici su vettore logical

n20_25 = sum(voti > 20 & voti < 25) % vettore logico
risultato di operatore logico &

C = xor(voti > 20, voti < 25)
```

Esercizio

```
anni = [19 20 21 18 18 20]; voti = [18 21 18 27 22 24];

%% estrazione di un sottovettore mediante indicizzazione di vettore logical

voti(find(voti > 25)) % uso della funzione find per estrarre indici (non necessario)

voti(voti > 25) % sottovettore da vettore logical

voti18enni = voti(anni == 18) % è sempre possibile estrarre sottovettori logical a patto che il vettore di indici abbia le stesse dimensioni del vettore

%% combiniamo un po' le cose viste sopra..

% media degli studenti con meno di 20 anni

media = sum(voti(anni < 20)) / sum(anni < 20)
```