Argomenti Avanzati di Programmazione Matlab

Informatica ICA (LC)

Giacomo Boracchi

giacomo.boracchi@polimi.it

26 Ottobre 2017

Altre Operazioni sugli Array

Subarray

Cancellazione elementi

Sottoarray (un vettore come indice di un vettore)

Si denota un sottoinsieme di un array usando vettori per valori degli indici

```
nomeVettore(vettoreIndici)
```

restituisce un vettore che comprende gli elementi di **`nomeVettore`** che compaiono nelle posizioni **`vettoreIndici`**

Estende il modo l'accesso ad un singolo elemento

```
nomeVettore(indice)
```

Esempi

```
> v=[6 8 4 2 4 5 1 3]
v = 6   8   4   2   4   5   1   3

>> v([1 4 7])
ans =     6     2     1

>> v(2:2:6)
ans =     8     2     5
```

primo, quarto settimo elemento

2:2:6 è il vettore [2, 4, 6]: indica secondo, quarto, sesto elemento

Sottovettori definiti da vettori di indici

Quindi, la notazione

```
a(v)
```

corrisponde a

```
[a(v(1)), a(v(2)),…, a(v(end))]
```

Attenzione che i valori di `v` devono essere interi positivi e minori delle dimensioni di `a`, devono essere indici validi.

La Keyword `end`

All'interno di **`vettoreIndici`** si può usare la keyword **`end`** che assume il valore dell'ultimo indice disponibile su una specifica dimensione di **`nomeVettore`**.

In questo modo non occorre conoscere le dimensioni del vettore

Esempi

```
» a = [1 : 10]
a =
     1     2     3     4     5     6     7     8     9    10
```

Toglie l'ultimo elemento

```
» b = a(1 : end - 1)
b =
     1     2     3     4     5     6     7     8     9
```

Legge il vettore dall'ultimo elemento al primo

```
» b = a(end : -1 : 1)
b =
    10     9     8     7     6     5     4     3     2     1
```

Esempi

```
> v=[6 8 4 2 4 5 1 3]
v =     6    8    4    2    4    5    1    3
>> v([1 4 7])


>> v(2:2:6)


>> v(3:end-2)


>> v(v)


>> v([1, 1, 1, 2, end])
```

Esempi

```
> v=[6 8 4 2 4 5 1 3]
v =     6    8    4    2    4    5    1    3
>> v([1 4 7])
ans =      6      2      1

>> v(2:2:6)


>> v(3:end-2)


>> v(v)


>> v([1, 1, 1, 2, end])
```

primo, quarto settimo elemento

Esempi

```
> v=[6 8 4 2 4 5 1 3]
v =     6    8    4    2    4    5    1    3
>> v([1 4 7])
ans =      6      2      1

>> v(2:2:6)
ans =      8      2      5

>> v(3:end-2)



>> v(v)



>> v([1, 1, 1, 2, end])
```

primo, quarto settimo elemento

2:2:6 è il vettore [2, 4, 6]: indica secondo, quarto, sesto elemento

Esempi

```
> v=[6 8 4 2 4 5 1 3]
v =     6    8    4    2    4    5    1    3
>> v([1 4 7])
ans =      6      2      1

>> v(2:2:6)
ans =      8      2      5

>> v(3:end-2)
ans =      4      2      4      5

>> v(v)



>> v([1, 1, 1, 2, end])
```

primo, quarto settimo elemento

2:2:6 è il vettore [2, 4, 6]: indica secondo, quarto, sesto elemento

dal terzo al terz'ultimo elemento

Esempi

```
> v=[6 8 4 2 4 5 1 3]
v =     6    8    4    2    4    5    1    3
>> v([1 4 7])
ans =      6      2      1

>> v(2:2:6)
ans =      8      2      5

>> v(3:end-2)
ans =      4      2      4      5

>> v(v)
ans =     5    3    2    8    2    4    6    4

>> v([1, 1, 1, 2, end])
```

primo, quarto settimo elemento

2:2:6 è il vettore [2, 4, 6]: indica secondo, quarto, sesto elemento

dal terzo al terz'ultimo elemento

i *valori* di v usati come indice

Esempi

```
> v=[6 8 4 2 4 5 1 3]
v =     6    8    4    2    4    5    1    3
>> v([1 4 7])
ans =      6      2      1

>> v(2:2:6)
ans =      8      2      5

>> v(3:end-2)
ans =      4      2      4      5

>> v(v)
ans =     5    3    2    8    2    4    6    4

>> v([1, 1, 1, 2, end])
ans = 6    6    6    8    3
```

primo, quarto settimo elemento

2:2:6 è il vettore [2, 4, 6]: indica secondo, quarto, sesto elemento

dal terzo al terz'ultimo elemento

i *valori* di v usati come indice

Indici ripetuti fanno replicare i valori nel sottovettore

Modificare un Sotto-Array

È possibile **effettuare l'assegnamento tra sottovettori** per modificare una parte del vettore

```
v1(vettoreIndici) = v2
```

Viene però richiesto che **`v2`** abbia **le stesse dimensioni** di **`v1(vettoreIndici)`**

```
>> a = [1 : 10]
a =
     1  2  3  4  5  6  7  8  9  10

>> a(1 : 3) = [0 0 0]
a =
     0  0  0  4  5  6  7  8  9  10
```

Modificare un Sotto-Array

È possibile **effettuare l'assegnamento tra sottovettori** per modificare una parte del vettore

`v1(vettoreIndici) = v2`

Viene però richiesto che **`v2`** abbia **le stesse dimensioni** di **`v1(vettoreIndici)`**

```
>> a =
      0  0  0  4  5  6  7  8  9  10
>> a(2 : 2 : end) = 2 * a(2 : 2 : end)
```

Modificare un Sotto-Array

È possibile **effettuare l'assegnamento tra sottovettori** per modificare una parte del vettore

```
v1(vettoreIndici) = v2
```

Viene però richiesto che **`v2`** abbia **le stesse dimensioni** di **`v1(vettoreIndici)`**

```
>> a =
     0  0  0  4  5  6  7  8  9 10
>> a(2 : 2 : end) = 2 * a(2 : 2 : end)
a =
     0  0  0  8  5 12  7 16  9 20
```

Modificare un Sotto-Array

È possibile **effettuare l'assegnamento tra sottovettori** per modificare una parte del vettore

`v1(vettoreIndici) = v2`

Viene però richiesto che **`v2`** abbia **le stesse dimensioni** di **`v1(vettoreIndici)`**

```
>> a =
      0  0  0  4  5  6  7  8  9 10
>> a(2 : 2 : end) = 2 * a(2 : 2 : end)
a =
      0  0  0  8  5 12  7 16  9 20
>> a(1 : 2 : end) = a(end : -2 : 1)
```

Modificare un Sotto-Array

È possibile **effettuare l'assegnamento tra sottovettori** per modificare una parte del vettore

```
v1(vettoreIndici) = v2
```

Viene però richiesto che **v2** abbia **le stesse dimensioni** di **v1(vettoreIndici)**

```
>> a =
     0  0  0  4  5  6  7  8  9 10
>> a(2 : 2 : end) = 2 * a(2 : 2 : end)
a =
     0  0  0  8  5 12  7 16  9 20
>> a(1 : 2 : end) = a(end : -2 : 1)
```

Modificare un Sotto-Array

È possibile **effettuare l'assegnamento tra sottovettori** per modificare una parte del vettore

```
v1(vettoreIndici) = v2
```

Viene però richiesto che **`v2`** abbia **le stesse dimensioni** di **`v1(vettoreIndici)`**

```
>> a =
      0  0  0  4  5  6  7  8  9 10
>> a(2 : 2 : end) = 2 * a(2 : 2 : end)
a =
      0  0  0  8  5 12  7 16  9 20
>> a(1 : 2 : end) = a(end : -2 : 1)
                    20 16 12  8  0
```

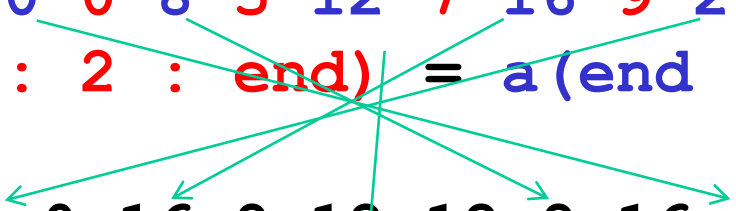
Modificare un Sotto-Array

È possibile **effettuare l'assegnamento tra sottovettori** per modificare una parte del vettore

```
v1(vettoreIndici) = v2
```

Viene però richiesto che **v2** abbia **le stesse dimensioni** di **v1(vettoreIndici)**

```
>> a =
     0  0  0  4  5  6  7  8  9 10
>> a(2 : 2 : end) = 2 * a(2 : 2 : end)
a =
     0  0  0  8  5 12  7 16  9 20
>> a(1 : 2 : end) = a(end : -2 : 1)
a =
    20  0 16  8 12 12  8 16  0 20
```



Sottoarray: Applicazione a Matrici

Si denota un sottoinsieme di un array usando vettori per valori degli indici

```
nomeMatrice(vettore1,vettore2)
```

restituisce una matrice che comprende gli elementi di **`nomeMatrice`** alle righe di indice in **`vettore1`** e alle colonne di indice in **`vettore2`**.

Sottoarray: Applicazione a Matrici

```
    m =  9     8     7
         6     5     4
         3     2     1
         0    11    12
         0     0     0

>> m([1 4], [2 3])
```

Sottoarray: Applicazione a Matrici

```
m =  9     8     7
     6     5     4
     3     2     1
     0    11    12
     0     0     0
```

```
>> m([1 4], [2 3])
ans =   8     7
       11    12
```

tutti gli elementi sulle
righe 1 e 4 e sulle colonne 2 e 3

Sottoarray: Applicazione a Matrici

```
    m =  9     8     7
         6     5     4
         3     2     1
         0    11    12
         0     0     0

>> m([1 4], [2 3])
ans =   8     7
        11    12

>> m(1:2:5, 1:end)
```

tutti gli elementi sulle
righe 1 e 4 e sulle colonne 2 e 3

Sottoarray: Applicazione a Matrici

```
m =  9     8     7
     6     5     4
     3     2     1
     0    11    12
     0     0     0
```

```
>> m([1 4], [2 3])
ans =  8     7
      11    12
```

tutti gli elementi sulle
righe 1 e 4 e sulle colonne 2 e 3

```
>> m(1:2:5, 1:end)
ans =  9     8     7
       3     2     1
       0     0     0
```

tutti gli elementi delle
righe 1, 3 e 5

Sottoarray: Applicazione a Matrici

```
    m =  9     8     7
         6     5     4
         3     2     1
         0    11    12
         0     0     0
>> m(1:2:5, :)
ans =    9     8     7
         3     2     1
         0     0     0
```

notazione ':' abbreviata per 1:end, cioè tutti i valori di quell'indice

Sottoarray: Applicazione a Matrici

```
    m =  9     8     7
         6     5     4
         3     2     1
         0    11    12
         0     0     0
>> m(1:2:5, :)
ans =    9     8     7
         3     2     1
         0     0     0

>> m(2:2:4, :) = [-1 -2 -3; -4 -5 -6]
m =      9     8     7
        -1    -2    -3
         3     2     1
        -4    -5    -6
         0     0     0
```

notazione ':' abbreviata per 1:end, cioè tutti i valori di quell'indice

uso della notazione dei sottoarray per individuare elementi oggetto di assegnamento

Assegnamenti con Scalari

È possibile associare a qualsiasi sotto array un valore scalare

`nomeVettore(vettoreIndici) = k`

Fa si che a tutti gli elementi di **`nomeVettore`** alle posizioni **`vettoreIndici`** venga assegnato il valore **`k`**

In questo modo è possibile inizializzare nuovi vettori.

```
» a = [1 : 10]
a =
    1    2    3    4    5    6    7    8    9   10

» a(1 : 3) = 0
a =
    0    0    0    4    5    6    7    8    9   10
```

Assegnamenti con Scalari

Esempio

m(1:4, 1:3) = 3 $\longrightarrow$ $\begin{bmatrix} 3 & 3 & 3 \\ 3 & 3 & 3 \\ 3 & 3 & 3 \\ 3 & 3 & 3 \end{bmatrix}$

Il modo con cui uno scalare viene assegnato a un array dipende dalla forma dell'array che viene specificata a sinistra dell'assegnamento

Esempio

m(1:2, 1:2) = 4 $\longrightarrow$ $\begin{bmatrix} 4 & 4 & 3 \\ 4 & 4 & 3 \\ 3 & 3 & 3 \\ 3 & 3 & 3 \end{bmatrix}$

Esempio

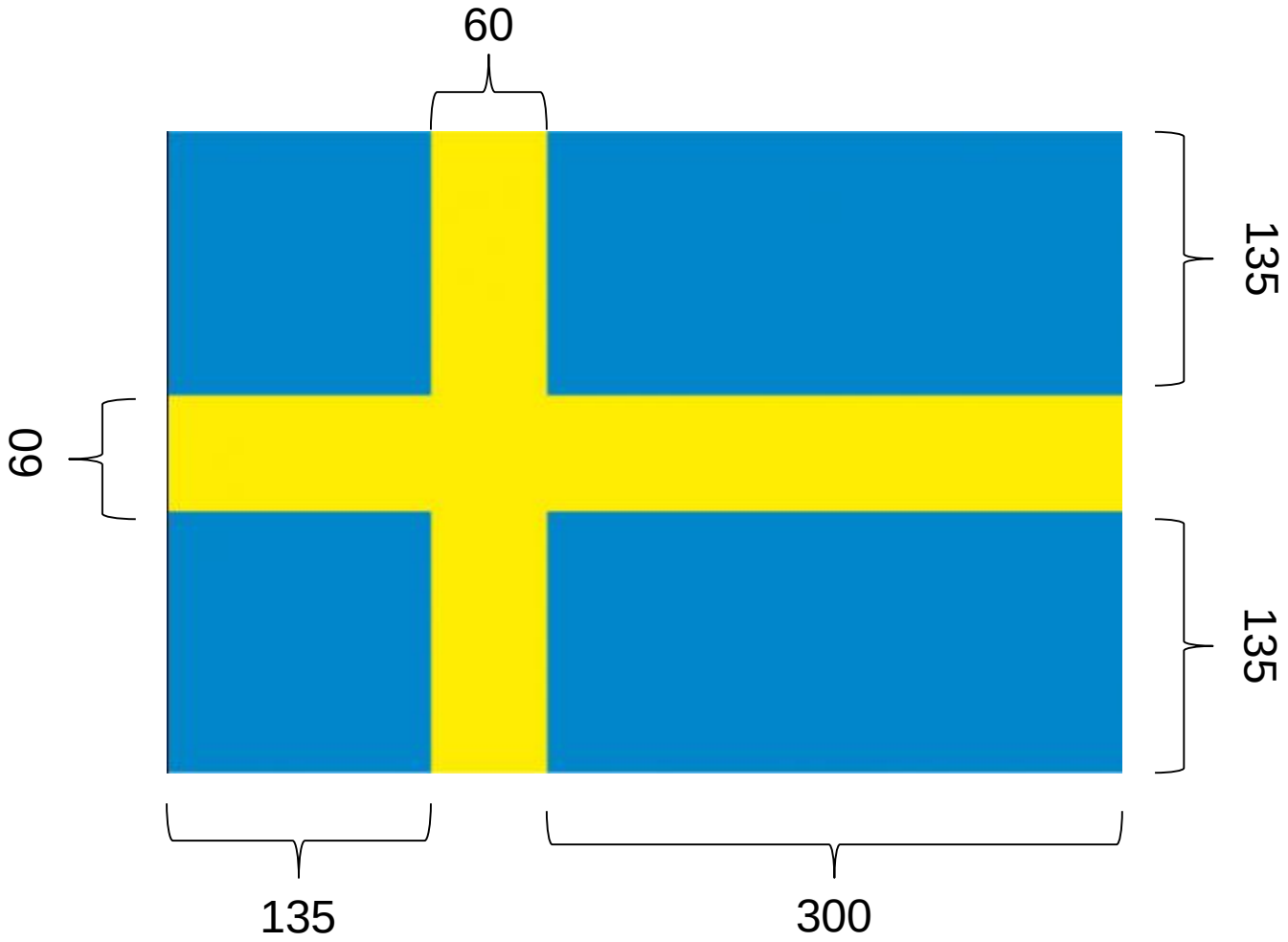
```
% inizializzare una matrice 5x5 con tutti valori a zero

% modificare la colonna centrale in 1

% modificare la riga centrale in 3

% sommare 2 ai valori della colonna centrale

% porre a 2 gli elementi nel primo quadrante

% copiare nell'ultima riga la prima riga letta al
contrario
```

Esempio

```
% inizializzare una matrice 5x5 con tutti valori a zero
A(5,5) = 0;
% modificare la colonna centrale in 1
A(: , 3) = 1;
% modificare la riga centrale in 3
A(3 , : ) = 3;
% sommare 2 ai valori della colonna centrale
A(: , 3) = A(: , 3) + 2; % NB termini a dx e sx
dell'uguale hanno la stessa dimensione
% porre a 2 gli elementi nel primo quadrante
A(1 : 2 , 1 : 2) = 2;
% copiare nell'ultima riga la prima riga letta al
contrario
A(end , :) = A(1 , end : -1 : 1)
```

Esercizio

Disegnare la bandiera svedese



GRAN CONTEST DELLE BANDIERINE

Regole:

- La bandiera deve essere realizzata interamente con uno script Matlab, senza interazione con l'utente
- E' necessario usare operazioni vettoriali, si possono usare cicli. Non assegnamenti pixel per pixel su tutta l'immagine.
- Potete presentare una sola bandiera a persona. Inviate una mail a me che include sia
  - Un'immagine della bandiera realizzata da voi in Matlab (salvare l'immagine come png)
  - Inviare lo script (opportunamente indentato e commentato) che genera l'immagine
  - Nel soggetto "Bandierine ICA LC"
- Deadline: Venerdì 10 Novembre

GRAN CONTEST DELLE BANDIERINE

Come creare le immagini a colori:

- E' possibile riprodurre i colori in due modi:
  - creando una matrice 3D B che ha N_righe x N_colonne x 3 dove il terzo piano indica il colore (B(:,:,1) è il rosso, B(:,:2) il verde, B(:,:,3) il blu) e visualizzando con imshow
  - Utilizzando la colormap e imagesc

Premi:

- Un punto alla bandierina più bella se lo studente è in grado di descrivere come ha raggiunto il risultato.

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           oxo                   o  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(3,4)=[]
??? 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
```

Memorizzazione degli Array

Gli array vengono salvati linearmente in memoria.

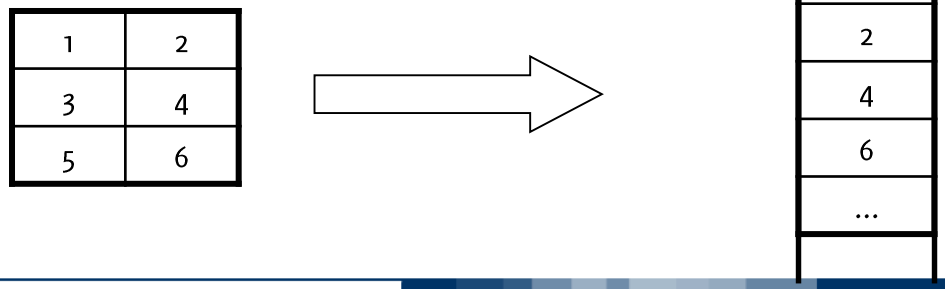
In particolare le matrici sono memorizzate

- 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*

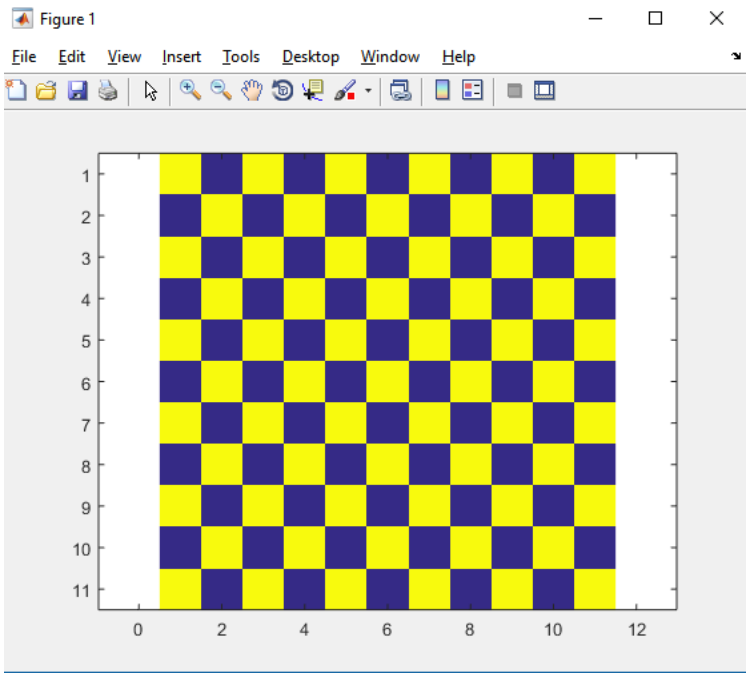
Si può accedere a un array a più dimensioni come se ne avesse una sola

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

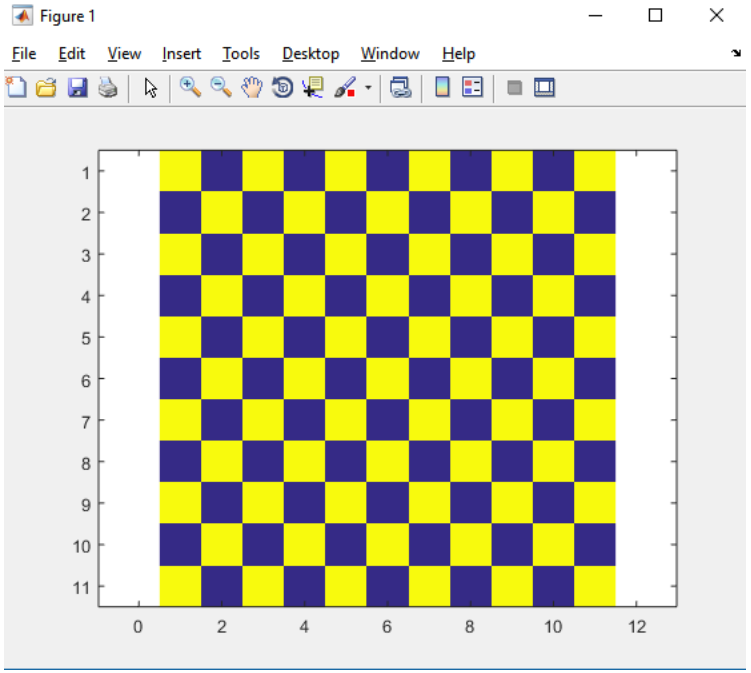
Scrivere una funzione che genera una matrice quadrata $n \times n$ di 0 ed 1 disposti «a scacchiera». La funzione controlla anche che n sia dispari



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)

if mod(n,2) == 0
    disp('n deve essere dispari');
else
     A = zeros(n);
     A(1 : 2 : end) = 1;
end
```




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

Tipo di Dato Logico

...e operazioni su vettori

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

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)) < = 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)

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

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:

- o come falso
- tutti i numeri diversi da o 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 o 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 o 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

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

Visualizzare solamente i numeri maggiori di 10

Portare a zero tutti gli elementi negativi

Sommare 10 ai numeri minori di 10

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

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.
```

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]
```

Esempio

Scrivere una funzione *cerca* che controlla se un elemento **x** appartiene ad un vettore **vett** e, in caso affermativo, ne restituisce la posizione

Esempi

```
function [pres, pos] = cerca(x, v)
    p=0; pos=[];
    for i=1:length(v)
        if v(i)==x
            p=p+1;
            pos(p)=i;
        end
    end
    pres=p>0;
```

```
>> A=[1, 2, 3, 4, 3, 4, 5, 4, 5, 6]
A = 1 2 3 4 3 4 5 4 5 6
>> [p, i]=cerca(4,A)
p = 1
i = 4 6 8
```

Esercizio: implementare usando find()

Esempi

```
function [pres, pos] = cerca2(x, v)
pres = 1;
pos = find(v == x);
if isempty(pos)
    pres = 0;
end
```

Esempi

```
function [pres, pos] = cerca2(x, v)
pres = 1;
pos = find(v == x);
if isempty(pos)
    pres = 0;
end
```

```
function [pres, pos] = cerca3(x, v)
pres = any(v == x);
pos = find(v == x);
```

Esercizio

Scrivere una funzione *closestVal* che prende in ingresso un vettore **vett** ed uno scalare **x** e restituisce il valore di **vett** più vicino ad **x**

Esercizio

Scrivere una funzione *closestVal* che prende in ingresso un vettore **vett** ed uno scalare **x** e restituisce il valore di **vett** più vicino ad **x**

```
function [closest, pos_closest] = closestVal(vett, val)

diff = vett - val;
abs_diff = abs(diff);
[~ , pos_closest] = min(abs_diff);
closest = vett(pos_closest);
Closest = unique(closest); % takes unique values
```

La funzione min ritorna due argomenti:
il valore minimo e le posizioni in cui questo compare.
A me serve solo il secondo argomento,
quindi scarto il primo inserendo la ~ al momento della chiamata

Esercizio

Scrivere un programma che determina se una parola è palindroma

Palindroma: soluzione

```
function res = palindroma(parola)

parolaAlContrario = parola(end : -1 : 1);

if parola == parolaAlContrario
    res = 1;
else
    res = 0;
end
```

Palindroma: soluzione

```
function res = palindroma(parola)

res = all(parola == parola(end : -1 : 1));
```

Esercizio

Scrivere un programma che richiede in ingresso due parole e determina se l'una è l'anagramma dell'altra

- Uno script si occupa dell'acquisizione delle parole.
- Implementare una funzione per creare l'istogramma delle parole.
- Eseguire nello script il confronto tra istogrammi e visualizzare a schermo il risultato.