Matrici e Operazioni Vettoriali

Informatica (ICA) AA 2020 / 2021

Giacomo Boracchi

20 Ottobre 2020

giacomo.boracchi@polimi.it

Esercizio di Stretching Mentale

Scrivere un programma che determina se una parola è palindoma

Matrici: Array Bidimensionali

Le Matrici

Le matrici sono array 2-D

Hanno quindi due indici:

- L'indice di riga (il primo)
- L'indice di colonna (il secondo)

Esempio:

```
A = 16     2     3    13
     5    11    10     8
     9     7     6    12
```

Le Matrici

Le matrici sono array 2-D

Hanno quindi due indici:

- L'indice di riga (il primo)
- L'indice di colonna (il secondo)

Esempio:

```
A = 16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1
```

Elemento alla riga 2 colonna 3

```
>> A(2, 3)

ans = 10
```

Creazione di Matrici

Le matrici vengono definite **affiancando vettori di dimensioni compatibili**

- Usiamo sempre gli operatori **,** (spazio) e **;** (vai a capo)
- L'operazione di **trasposizione** inverte le righe e le colonne della matrice

Es:

```
>> a = [1 , 2 ; 3 ,4 ]
a =
     1     2
     3     4
```

```
a' =
     1     3
     2     4
```

Le dimensioni di una matrice

Per scorrere una matrice è spesso necessario conoscere le sue dimensioni

Il comando size(A, dim) restituisce il numero di elementi di A lungo la dimensione dim

```
A = 16     2     3    13
     5    11    10     8
     9     7     6    12
righe = size(A, 1);
colonne = size(A, 2);
```

Le Matrici: operatore CAT

La concatenazione dei vettori avviene mediante operatore **CAT** che richiede **dimensioni consistenti** dei vettori

```
>> a = [1 : 3]
a =
     1     2     3

>>  b = [4; 5; 6]
b =
     4
     5
     6
```

```
>> A = [a; b]
Error using vertcat
CAT arguments dimensions are
not consistent.

>> A =[a, b]
Error using horzcat
CAT arguments dimensions are
not consistent.

>> A =[a; b']
A =
     1     2     3
     4     5     6
```

Accedere agli Elementi di una Matrice

Per accedere agli elementi di una matrice occorre specificare un valore per ogni indice

`nomeMatrice(indice1, indice2)`

Seleziona il valore alla riga **`indice1`** colonna **`indice2`** nella variabile **`nomeMatrice`**

Es

```
>> A = [1 : 3; 4 : 6; 7: 9 ]
A =
     1     2     3
     4     5     6
     7     8     9
```

```
>> A(2, 3)
ans =
     6
>> A(3,5)
Index exceeds
matrix dimensions.
```

Esempio

Visualizzare a schermo le prime 10 tabelline (utilizzando solo operazioni matriciali)

Stampare solo la parte triangolare bassa delle tabelline

Stampare solo la parte triangolare alta delle tabelline

Memorizzazione degli Array

Gli array vengono salvati linearmente in memoria.

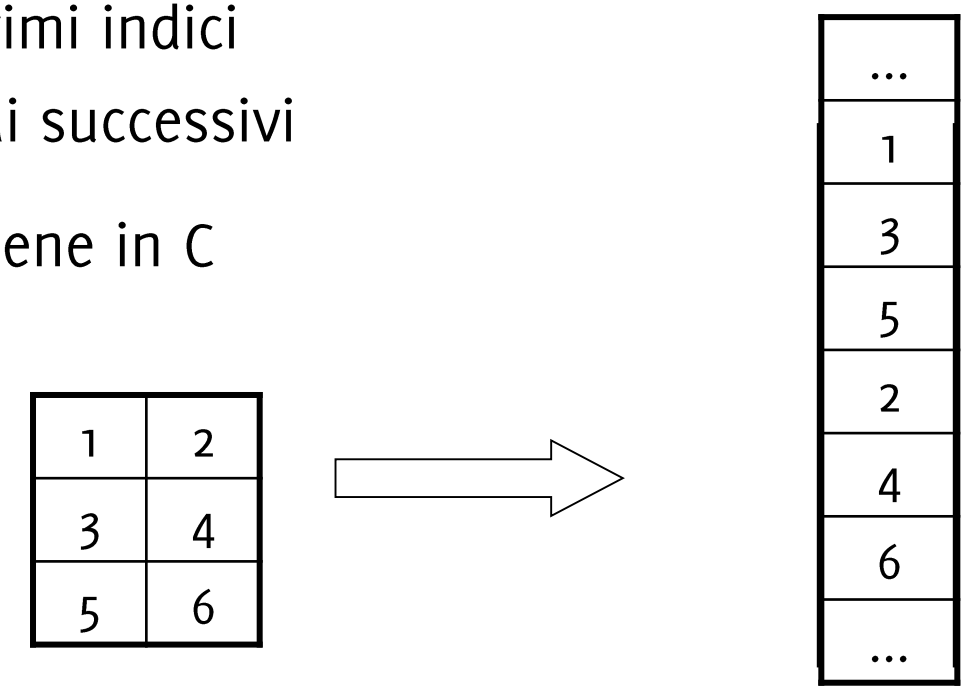
In particolare le matrici sono memorizzate

- 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
```

TODO: Esercizio su Matrici

Si scriva un programma che prende in ingresso una matrice quadrata e controlla che sia simmetrica.

Una matrice è simmetrica se per ogni elemento vale la seguente proprietà: L'elemento alla riga i, colonna j coincide con l'elemento alla riga j colonna i

Es di matrice simmetrica

1	12	1
12	0	3
1	3	23

Array Multidimensionali

È possibile specificare una terza (quarta, quinta…) dimensione lungo la quale indicizzare un array.

Ad esempio le immagini a colori sono definite con tre piani colore (RGB), quindi

- un'immagine a colori 10 Mpixels, aspect ratio (3:4) richiede una matrice 3D di 2736 x 3648 x 3 elementi
- 10 sec di video full HD (1080 x 768) a 24fps richiede una matrice 4D di 1080 x 768 x 3 x (10 x 24) elementi

Strutture in Matlab

Struct vs Array

Gli **array** permettono di aggregare variabili **omogenee** in una sequenza

Le `struct` permettono di aggregare variabili **eterogenee** in una sola variabile

- Le `struct` è una sorta di "contenitore" per variabili disomogenee di tipi più semplici.
- Le variabili aggregate nella struct sono dette **campi** della struct

Esempio: variabile per contenere anagrafica di impiegati

- *nome, cognome, codice fiscale, indirizzo, numero di telefono, stipendio, data di assunzione etc.*
- *Non posso metterli in un array, sono variabili diverse, è molto sconveniente metterle in variabili separate, specialmente se ho diversi impiegati*

Creazione di una struct

Creazione di una struttura :

Utilizzando la funzione struct()

```
studente = struct('nome', 'Giovanni', 'eta', 24)
```

Assegnamento dei valori ai campi (e contestuale definizione dei campi)

```
studente.nome = 'Giovanni';
studente.eta = 24;
```

Accedere ai campi di una `struct`

Per accedere ai campi si usa l'operatore ***dot***.
Sintassi:

```
nomeStruct.nomeCampo;
```

Quindi, **`nomeStruct.nomeCampo`** diventa, a tutti gli effetti, **una «normale» variabile** del tipo di **`nomeCampo`**.

- **Ai campi** di una struttura applicabili tutte le **operazioni caratteristiche** del tipo di appartenenza
- In questo senso, il *dot* è l'omologo di **`(indice)`** per gli array

Creazione di una struttura campo per campo

Esempio: creo una struttura studente

```
studente.nome = 'Giovanni Rossi';
studente.indirizzo = 'Via Roma 23';
studente.citta = 'Cosenza';
studente.eta = 25;
```

Accesso ai campi come nel C con l'operatore .

```
nomeStruttura.nomeCampo
```

Es

```
disp([studente.nome, ' (' , studente.citta ,') ha ' ,
num2str(studente.eta) , ' anni'])
```

Creazione di una struttura campo per campo

Esempio: la struttura studente

```
studente.nome = 'Giovanni Rossi';
studente.indirizzo = 'Via Roma 23';
studente.citta = 'Cosenza';
studente.media = 25;
```

É possibile far diventare **studente** un array di strutture, accodando un altro elemento in **studente(2)**.

```
studente(2).nome = 'Giulia Gatti';
studente(2).media = 30;
```

Tutte le strutture dell'array devono avere gli stessi campi (**l'array deve essere omogeneo, la struttura non necessariamente**).

É possibile assegnare solo alcuni campi a **studente(2)**: i campi non assegnati rimangono vuoti.

Aggiunta di campi

Aggiunta di un campo

```
%facciamo riferiemento alla definizione di studente delle slide precedenti
studente(2).esami = [20 25 30];
```

Il campo esami viene aggiunto a tutte le strutture che fanno parte di studente

- Avrà un valore iniziale per studente(2). Sarà vuoto per tutti gli altri elementi dell'array

Creazione di una struttura mediante la funzione struct

Consente di preallocare una struttura o un array di strutture

```
S = struct('campo1',val1,'campo2',val2, …)
```

Es:
```
rilieviAltimetrici =
struct('latitudine',30,'longitudine',60, 'altitudine',
1920)
```

Creazione di una struttura mediante la funzione struct

Consente di preallocare una struttura o un array di strutture

```
S = struct('campo1',val1,'campo2',val2, …)
```

Es: `rilieviAltimetrici = struct('latitudine',30,'longitudine',60, 'altitudine', 1920)`

Esempio array di strutture:

```
s(5) = struct('x',10,'y',3);
```

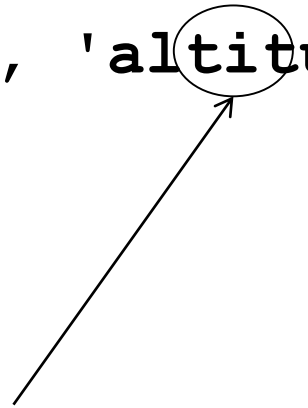
- s è un array 1x5 in cui ogni elemento ha attributi x e y
- solo il quinto elemento di s viene inizializzato con i valori x=10 e y=3
- gli altri elementi vengono inizializzato con il valore di default: [] (array vuoto)

Creazione di una struttura mediante la funzione struct

Consente di preallocare una struttura o un array di strutture

```
S = struct('campo1',val1,'campo2',val2, …)
```

Es: `rilieviAltimetrici(1000) = struct('latitudine',30,'longitudine',[], 'altitudine', 1920)`



Array vuoto. Attenzione: se si Inserisce un valore (es. 20), questo viene assunto dal campo longitudine dell'elemento 1000, ma non dallo stesso campo degli altri elementi dell'array

Array di strutture innestati

Un campo di una struttura può essere di qualsiasi tipo

E` quindi possibile avere un campo che è, di nuovo, una struttura o un array di strutture

Esempio

```
studente(1).corso(1).nome='InformaticaB';
studente(1).corso(1).docente='Von Neumann';
studente(1).corso(2).nome='Matematica';
studente(1).corso(2).docente='Eulero';
```

corso è un array di strutture

```
>> studente
studente =
    corso: [1x2 struct]
```

Esercizio

Si sviluppi uno script matlab che acquisisce da tastiera i dati relativi ad un numero arbitrario di rilievi altimetrici e che quindi stampa a video l'altitudine media di tutti i rilievi che si trovano nell'intervallo

- latitudine [30, 60]
- longitudine [10, 100]

Esercizio, la roulette

```
% Scrivere un programma per simulare il gioco della roulette
% la roulette possiede 38 numeri (da 1 a 36, lo zero e il doppiozero)
% 0 e 00 non sono ne pari ne dispari (vince il banco)
%
% il banco inizialmente possiede 5000 euro
% i giocatori possiedono inizialmente 5000 euro
%
% 1) assumere ad ogni giocata che il giocatore 1 punti 5 euro su pari
% o dispari con la stessa probabilità
% se vince, giocatore1, ottiene 2 volte la posta,
% se perde il banco incassa il valore giocato.
%
%   Mostrare la variazione dell'ammontare del banco e del
%   giocatore all'aumentare delle giocate fino a che o il
%   giocatore perde il banco viene sbancato
%
```

% hints
% - utilizzare la funzione rand() per generare numeri uniformemente distribuiti in [0,1]. Riscalarli quindi in [0 , 38] e approssimarli
% - utilizzare un array di strutture per contenere i giocatori (è possibile aggiungere ulteriori campi alle strutture)
% - utilizzare, dove possibile, funzioni da voi sviluppate

```
%
%2) aggiungere un secondo giocatore che punta sempre 1 euro
sul 15 (se esce 15 vince 36 volte la posta)
%
%3) aggiungere un terzo giocatore che usa la seguente strategia:
%   egli punta sempre sul pari e inizialmente punta un euro.
%   se vince ricomincia a puntare un euro sempre sul pari
%   se perde raddioppia la puntata sempre sul pari,
%   se non ha abbastanza soldi punta tutto quello che possiede
%
```

TODO: Esercizio su Matrici

Si scriva un programma che prende in ingresso una matrice quadrata e controlla che sia simmetrica.

Una matrice è simmetrica se per ogni elemento vale la seguente proprietà: L'elemento alla riga i, colonna j coincide con l'elemento alla riga j colonna i

Es di matrice simmetrica

1	12	1
12	0	3
1	3	23

Esercizio

Si sviluppi uno script matlab che acquisisce da tastiera i dati relativi ad un numero arbitrario di rilievi altimetrici e che quindi stampa a video l'altitudine media di tutti i rilievi che si trovano nell'intervallo

- latitudine [30, 60]
- longitudine [10, 100]

Soluzione

```
% s = struct('altezza',[],'latitudine',[], 'longitudine',[])
n = input(['quanti rilievi? ']);
 % acquisizione dei rilievi
for ii = 1 : n
    s(ii).altezza = input(['altezza rilievo nr ', num2str(ii), ' ']);
    s(ii).latitudine= input(['latitudine rilievo nr ', num2str(ii), ' ']);
    s(ii).longitudine= input(['longitudine rilievo nr ', num2str(ii), ' ']);
end
% creo dei vettori con i valori dei campi
LAT = [s.latitudine];
LON = [s.longitudine];
ALT = [s.altezza];
% operazioni logiche per definire il sottovettore da estrarre da altezza
latOK = (LAT > 30) & (LAT <60);
lonOK = (LON> 10) & (LON<100);
posOK =latOK & lonOK;

% estrazione sottovettore e calcolo media
mean(ALT(posOK));
```

Altre Operazioni sugli Array

Subarray

Cancellazione elementi

Esercizietto di riscaldamento

Dato un vettore **`v`** di interi ed un vettore **`indx`** di posizioni ammissibili per **`v`** copiare in un secondo vettore **`t`** tutti i valori di **`v`** che compaiono nelle posizioni **`indx`**.

Domanda: quali sono le posizioni ammissibili per **`v`**?

- Interi positivi (**>0**) che sono minori o uguali alla lunghezza di **`v`**

Esercizietto di riscaldamento

Dato un vettore **v** di interi ed un vettore **indx** di posizioni ammissibili per **v** copiare in un secondo vettore **t** tutti i valori di **v** che compaiono nelle posizioni **indx.**

```
t = [];
for ii = indx
    t = [t, v(ii)];
end
```

Nella pratica…

```
v = [10 : 2 : 16]
indx = [2, 3]


t = [];
for ii = indx
    t = [t, v(ii)];
end
disp(t);
```

oppure

```
t = [];
ii = 1;
while (ii <= length(indx))
    t(ii) = v(indx(ii));
    ii = ii + 1;
end
disp(t);
```

```
>> t =
    12    14
```

Oppure..

```
>> v(indx)

     12     14
```

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 = [10 : 10: 100]
a =
    10    20    30    40    50    60    70    80    90   100
```

Toglie l'ultimo elemento

```
» b = a(1 : end - 1)
b =
    10    20    30    40    50    60    70    80    90
```

Legge il vettore dall'ultimo elemento al primo

```
» b = a(end : -1 : 1)
b =
   100    90    80    70    60    50    40    30    20    10
```

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
```

primo, quarto settimo elemento

```
>> 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)
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
```

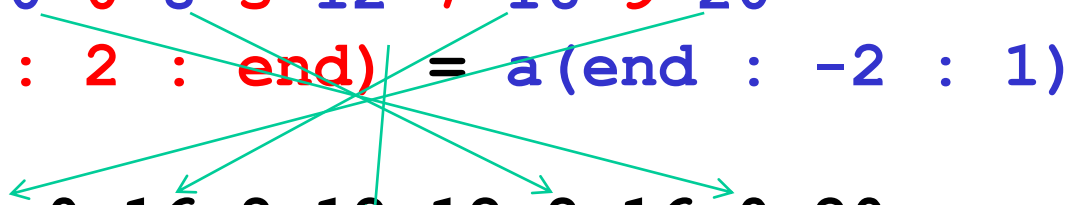
Modificare un Sotto-Array

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
```

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

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

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.

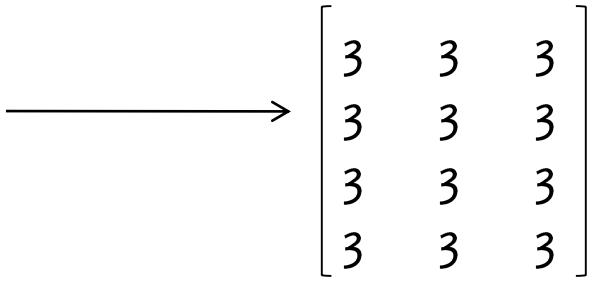
```
» a = [1 : 10]
a =
    1    2    3    4    5    6    7    8    9   10

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

Assegnamenti con Scalari

Esempio

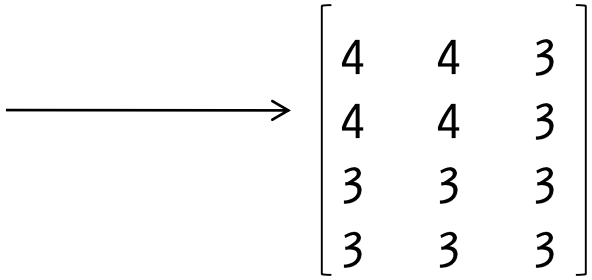
m(1:4, 1:3) = 3



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



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)
```

Assegnamenti con Scalari su matrici

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

Es.

```
>> clear m;

>> m(4, 3) = 3;

>> m(1:2, 1:2) = 4

ans =

     4     4     0

     4     4     0

     0     0     0

     0     0     3
```