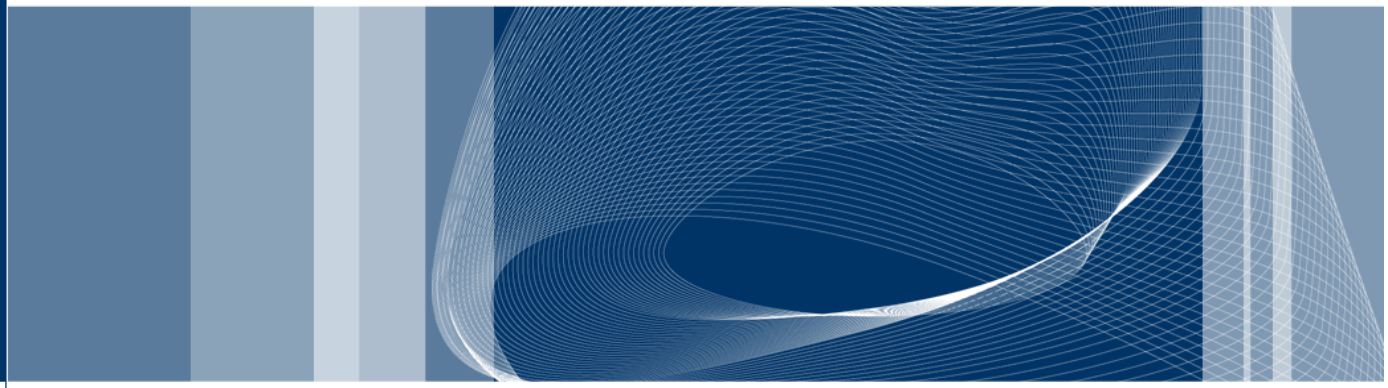
POLITECNICO DI MILANO



Strutture di controllo condizionali in Matlab

Informatica B

Daniele Loiacono

Le strutture condizionali

A cosa servono le strutture condizionali?

```
%Inserimento dei parametri della parabola: y=ax^2+bx+c
a = input('inserire a: ');
b = input('inserire b: ');
c = input('inserire c: ');
delta = b^2-4*a*c;
x(1) = (- b - sqrt(delta))/(2*a);
x(2) = (- b + sqrt(delta))/(2*a);
y = a*x.^2 + b*x + c;
```

Istruzioni in sequenza

- Cosa succede se il discriminante non è positivo ?

Costrutto if

- Consente di effettuare scelte e modificare il **flusso di esecuzione** del programma

```
if delta > 0
  x(1) = (- b - sqrt(delta))/(2*a);
  x(2) = (- b + sqrt(delta))/(2*a);
  y = a*x.^2 + b*x + c;
end
```

- **Semantica:** il corpo viene eseguito <u>solo</u> se la condizione è **vera**

```
if <condizione>
    <corpo>
end
```

Sintassi

Costrutto if-else

```
if delta > 0
    x(1) = (- b - sqrt(delta))/(2*a);
    x(2) = (- b + sqrt(delta))/(2*a);
    y = a*x.^2 + b*x + c;
  else
    disp('discriminante non positivo');
  end
```

- **Semantica:** il ramo-if viene eseguito solo se la condizione è **vera**, il ramo-else solo se la condizione è **falsa**

```
if <condizione>
    <ramo-if>
else
    <ramo-else>
end
```

Sintassi

Costrutto if-elseif-else

```
if delta > 0
  x(1) = (- b - sqrt(delta))/(2*a);
  x(2) = (- b + sqrt(delta))/(2*a);
  y = a*x.^2 + b*x + c;
elseif delta == 0
  x = -b/(2*a);
  y = a*x.^2 + b*x + c;
else
  disp('No intersezioni');
end
```

```
if <condizione-1>
   <ramo-1>
elseif <condizione-2>
   <ramo-2>
…
elseif <condizione-n>
   <ramo-n>
else
   <ramo-else>
end
```

Sintassi

- **Semantica:** viene eseguito il ramo i-esimo solo se la condizione i-esima è **vera**; se nessuna condizione è **vera**, viene eseguito il ramo-else.

Condizioni logiche in Matlab

Tipo di dato logico

- È un tipo di dato che può avere solo due valori
  - true (vero) → rappresentato come 1
  - false (falso) → rappresentato come 0
- I valori di questo tipo possono essere generati
  - direttamente da due funzioni speciali (true e false)
  - dagli operatori relazionali
  - dagli operatori logici
- Esempio

```
a=true;
% a è un vettore 1x1 di tipo "logico"
```

Valori logici vs valori numerici

- I valori logici sono rappresentati come 0 e 1 ma **non sono** valori numerici
- Infatti, occupano meno memoria (possono valere solo 0 o 1) dei valori numerici
- Tuttavia… se usati in un espressione numerica o in una funzione come argomento numerico, i valori logici vengono convertiti automaticamente in valori numerici
  - Esempio

    `a = true + 3 % a`←`4`
- È possibile convertire un valore numerico in valore logico attraverso la funzione `logical`

```
b =logical(a)
```

  - ogni elemento di `a` diverso da zero viene convertito in `true`, gli elementi di a pari a 0 vengono convertiti in `false`

Operatori relazionali

- Gli operatori relazionali operano su valori numerici e carattere
- **Sintassi**: *a* OP *b*
  - *a* e *b* possono essere espressioni aritmetiche, variabili, o costanti
  - OP può essere ==, ~=, >, >=, <, <=
- Esempi:

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

Note

- Non bisogna confondere == e =
  - == è un operatore di confronto
  - = è un operatore di assegnamento
- Attenzione a possibili errori numerici di approssimazione:
  - `sin(0) == 0` %restituisce true
  - `sin(pi) == 0` %restituisce false
  - Invece dovrebbero essere vere entrambe!!
- Per i numeri piccoli conviene usare una soglia

```
abs( sin(pi) ) <= eps
```

Operatori logici

- Gli operatori logici consentono di costruire delle condizioni complesse a partire da condizioni più semplici
- I quattro operatori logici principali sono: AND, OR, NOT e XOR
- AND, OR e XOR sono operatori binari, NOT è un operatore unario
- Gli operatori vengono definiti attraverso una **tavola della verità**:

A	B	A AND B	A OR B	A XOR B
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

A	NOT A
0	1
1	0

Operatori logici in Matlab

- Operatori logici binari: a OP b
  - a,b possono essere variabili, costanti o espressioni di tipo logico
  - OP: && (AND), || (OR)
  - Operatore XOR disponibile come funzione: xor(a,b)
- Operatori logici unari: OP a
  - a,b possono essere variabili, costanti, espressioni da valutare, scalari o vettori (dimensioni compatibili)
  - OP1: NOT (~)
- Nel caso a e b siano espressioni/variabili di tipo numerico verranno valutate come segue:
  - il valore 0 viene valutato come `false`
  - tutti i valori diversi da 0 sono valutati come `true`

Operatori logici: esempio

```
anni = input('Quanti anni hai? ');
studente = input ('Sei studente (true o false)? ');
if xor( anni<12 || anni>65, studente)
    disp('Ingresso ridotto');
elseif ( anni<12 || anni>65 ) && studente
    disp('Ingresso gratuito');
else
    disp('Ingresso intero');
end
```

Ordine tra gli operatori

- Un'espressione viene valutata nel seguente ordine:
  - trasposizione ed elevamento a potenza
  - NOT
  - operatori aritmetici
  - operatori relazionali
  - AND
  - OR e XOR
- È possibile utilizzare le parentesi tonde per specificare la precedenza desiderata
  - Non si possono usare altri tipi di parentesi per questo scopo
  - È possibile inserire diversi livelli di parentesi (tonde) uno dentro l'altro

Strutture condizionali e array

Operatori logici/relazionali e array

- In Matlab è possibile applicare gli operatori logici e relazionali anche a variabili di tipo array.
- Quando si applica l'operatore unario NOT (~) ad un array A di dimension MxN, il risultato è un array di dimensioni MxN ottenuto applicando il NOT a ciascun elelemento di A
- Quando si applica un operatore binario, C = A OP B:
  - A, B e C sono entrambi array MxN e ciascun elemento di C è ottenuto applicando l'operatore OP ai corrispondenti elementi di A e B
  - A è un array MxN, B è uno scalare, C è un array MxN i cui elementi sono ottenuti applicando OP al corrispondente elemento di A e allo scalare 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

Operatori e array: esempio

- Scrivere una funzione per calcolare il numero di elementi in un vettore compresi fra due valori

```
function n = compresi(v,x,y)
   n = sum(v>=x & v<=y);
```

  - Esempio di chiamata

```
compresi([18 27 24 23 29 30 20 26], 24, 27)
```

Operatori e array: esempio

- Scrivere una funzione per calcolare il numero di elementi in un vettore compresi fra due valori

```
function n = compresi(v,x,y)
   n = sum(v>=x & v<=y);
```

  - Esempio di chiamata

    `compresi([18 27 24 23 29 30 20 26], 24, 27)`

    `[18 27 24 23 29 30 20 26]>=24` → `[0 1 1 0 1 1 0 1]`

Operatori e array: esempio

- Scrivere una funzione per calcolare il numero di elementi in un vettore compresi fra due valori

```
function n = compresi(v,x,y)
   n = sum(v>=x & v<=y);
```

  - Esempio di chiamata

    `compresi([18 27 24 23 29 30 20 26], 24, 27)`

    `[18 27 24 23 29 30 20 26]>=24` → `[0 1 1 0 1 1 0 1]`

    `[18 27 24 23 29 30 20 26]<=27` → `[1 1 1 1 0 0 1 1]`

Operatori e array: esempio

- Scrivere una funzione per calcolare il numero di elementi in un vettore compresi fra due valori

```
function n = compresi(v,x,y)
   n = sum(v>=x & v<=y);
```

  - Esempio di chiamata

```
compresi([18 27 24 23 29 30 20 26], 24, 27)
```

```
[18 27 24 23 29 30 20 26]>=24 → [0 1 1 0 1 1 0 1]
                                          &
[18 27 24 23 29 30 20 26]<=27 → [1 1 1 1 0 0 1 1]
                                ----------------
                                [0 1 1 0 0 0 0 1]
```

Operatori e array: esempio

- Scrivere una funzione per calcolare il numero di elementi in un vettore compresi fra due valori

```
function n = compresi(v,x,y)
   n = sum(v>=x & v<=y);
```

  - Esempio di chiamata

    `compresi([18 27 24 23 29 30 20 26], 24, 27)`

```
[18 27 24 23 29 30 20 26]>=24 → [0 1 1 0 1 1 0 1]
                                          &
[18 27 24 23 29 30 20 26]<=27 → [1 1 1 1 0 0 1 1]
                                ----------------
                                [0 1 1 0 0 0 0 1]
```

→ `n = sum ([0 1 1 0 0 0 0 1])` → **`n = 3`**

Operatori logici/relazionali e stringhe

- Le stringhe sono vettori di caratteri, quindi possiamo sfruttare gli operatori logici e relazionali anche per lavorare con le stringhe
  - Ad esempio si possono facilmente confrontar due stringhe di lunghezza uguale:
    'pippo'=='pluto' [1 0 0 0 1]

Funzioni logiche

Nome della funzione	Elemento restituito
all(x)	un vettore riga, con lo stesso numero di colonne dell'array x, che contiene 1, se la corrispondente colonna di x contiene tutti elementi non nulli, o 0 altrimenti
any(x)	un vettore riga, con lo stesso numero di colonne dell'array x, che contiene 1, se la corrispondente colonna di x contiene almeno un elemento non nullo, o 0
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

Array logici e selezione

Indici e vettori logici

- In Matlab è possibile utilizzare un array logico come indice per selezionare gli elementi desiderati di un array:

```
sel = orig(idx)
```

  - `orig` è un array di dimensione MxN
  - `idx` è un array **di tipo logico** di dimensione MxN
  - `sel` è un vettore che contiene tutti gli elementi dell'array `orig`, che si trovano in corrispondenza degli elementi di `idx` pari a `1`

- Attenzione! Per creare un vettore logico NON basta creare un vettore di 0 e 1 (numeri), bisogna convertirlo con la funzione logical

```
i = [1 0 1; 0 1 0];
j = logical(i)
A = [1 2 3; 4 5 6];
A(j) → [1; 5; 3]
A(i) → errore
```

Indici e vettori logici: esempi

- Troviamo tutti gli elementi di x che sono minori del corrispondente elemento in y

```
x = [6,3,9]
y = [14,2,9]
z = x(x<y) %z = 6
```

Altre funzioni logiche

- i = 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
```

- Notate che find restituisce gli indici e non i valori degli array mentre usando i vettori logici come indici si ottengono i valori
- Esempio

```
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]
```