Logica e relazioni

- Operatori relaizonali
- Vettori logici e indirizzamento
- Operatori logici
- Funzioni logiche (find, ecc…)

Operatori Relazionali in MATLAB

A operatore B

- A e B possono essere:
  - Variabili, costanti o espressioni da valutare
  - Scalari o Vettori (dimensioni compatibili!)
  - Numerici o stringhe
- Operatori: `>` `>=` `==` `<` `<=` `~=`
- Il risultato è vero (1) o falso (0) – anche vettore

- Altri esempi

espressione	risultato
5 < 7	1
[3 5 2] >= [1 0 12]	1 1 0
max(1:6) <= 7	1
[3 pi -12] > 1	1 1 0
'Tom' == 'Bob'	0 1 0
'Tom' == 'm'	0 0 1

I vettori e le stringhe devono avere le stesse dimensioni!

Questi sono valori logici!! v. Workspace

- Note:
  - I vettori logici possono ancora essere usati come numerici!
  - Possiamo usare il risultato per fare un calcolo: es.

    “Quante lettere “s” in una parola?”

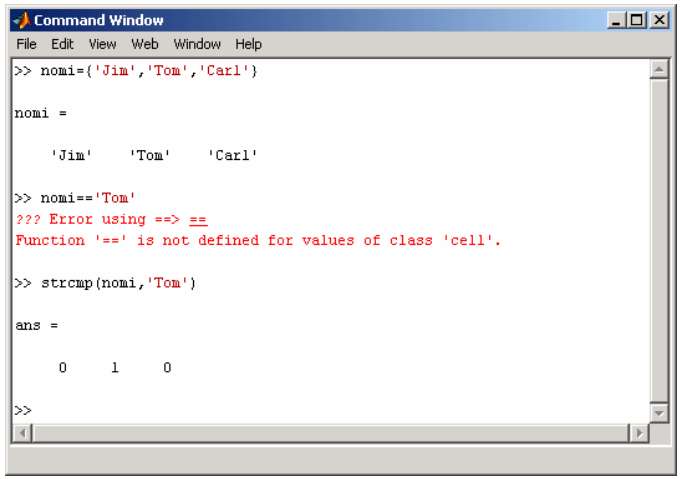
```
>> string = 'Mississippi'

string =

Mississippi

>> sum(string=='s')

ans =

     4
```

STRCMP e vettori di celle

- Per cercare una parola in un vettore di celle contenente stringhe non possiamo usare l'operatore == perché non è definito per le celle.
- Si può usare la funzione strcmp
- Per altre funzioni sulle stringhe digitare:

  help strfun

STRCMP: esempio



- Non confondiamo **= =** e **=**
- La precisione finita può "fregare" il **~ =**

  sind(0) = = 0 ⟶ 1

  sind(180) = = 0 ⟶ 0

Per i numeri piccoli usiamo eps!

abs(sind(180)) < = eps ⟶ 1

Selezionare elementi di vettori usando i vettori logici

Quando utilizziamo un vettore logico come indice per un array, vengono estratti gli elementi corrispondenti ai valori 1 del vettore logico.

Quindi se digitiamo A(j), dove j è un vettore logico della stessa dimensione di A, otteniamo i valori di A corrsipondenti agli indici degli 1 del vettore j.

N.B.Per creare un vettore logico NON basta creare un vettore di 0 e 1 (numeri), bisogna convertirlo con la funzione `logical`.

```
>> A = 1:5
A =
     1    2    3    4    5
>> i = [1,0,0,0,1];
i =
     1    0    0    0    1
>> j=logical(i)
j =
     1    0    0    0    1
>> A(j)
ans =
     1    5
>> A(i)
??? Index into matrix is negative or zero.
  See release notes on change to logical
  indices.
```


Gli operatori relazionali possono essere usati direttamente per selezionare gli elementi di un vettore.

Per esempio, se `x = [6,3,9]` e `y = [14,2,9]`, digitando

```
z = x(x<y)
```

Troviamo tutti gli elementi di x che sono minori del corrispondente elemento in y.
Il risultato sarà:

```
z = 6.
```


Operatori logici

A operatore B

- A e B possono essere:
  - Logici o Numerici
  - Variabili, costanti o espressioni da valutare
  - Scalari o Vettori (dimensioni compatibili!)
- A e B, se sono numerici, verranno interpretati come logici (binari):
  - Il numero 0 viene interpretato come falso
  - Tutto il resto viene interpretato come vero
- Il risultato è vero (1) o falso (0) – anche vettore

- Operatori base: and & or |
  xor not ~

A	B	A&B	A|B	xor(A,B)	~A
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

"tabella di verità"

Operatore "unario"

Esempi:

- “Hai tra 25 e 30 anni?”

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

- “siamo in inverno?”

  (mese==12 & giorno>=22) | (mese==1) |
  (mese==2) | (mese==3 & giorno<=21)

- Con i vettori:

  Score = [70, 55, 88, 98, 80, 73, 90]
  C = (Score > 70) & (Score < 81)
  → C = [0 0 0 0 1 1 0]

- Utile per contare quanti elementi soddisfano una condizione:

  B_grades = sum(Score<91 & Score>80)

- Esempio con testo:

  'Tom'= ='m' | 'Tom'= ='o' → 0 1 1

  name = input('enter name','s');
  name = = 'Tom' | name = = 'Bob'

- Lancio di un dado:

  roll = sum(ceil(6*rand(1,2)));
  roll = = 7 | roll = = 11

Precedenza degli operatori (da sinistra a destra)

1. Parentesi ( )
2. Trasposta (') and potenza (.^)
3. Negativo (-) e negazione logica (~)
4. moltiplicazione (.*) e divisione (./),
5. Addizione (+) sottrazione (-)
6. Operatore due punti (:)
7. Operatori relazionali (<, <=, >, >=, = =, ~=)
8. AND logico (&)
9. OR logico (|)

Gli operatori aritmetici +, -, *, /, e \ hanno la precedenza rispetto agli operatori relazionali. Quindi il comando

```
z = 5 > 2 + 7
```

è equivalente a

```
z = 5 >(2+7)
```

e dà come risultato `z = 0` (logical).

Possiamo usare le parentesi per cambiare l'ordine di precedenza; per esmpio, `z = (5 > 2) + 7` restituisce `z = 8` (numero)

Altri operatori logici:

- Estendere **|** ed **&** da op.binari a vettoriali :

  any(X) all(X)

- Per verificare dimensioni, valori e tipi dei vettori:

  isempty(A)

  isinf(A) isnan(A)

  ischar(A) isnumeric(A)

- Per trovare i valori per cui è vera una condizione: find( )

```
>> A = [ 5 6 7 2 10 ];
>> find(A>5)

ans =

     2     3     5
```


La funzione `find`

`find(A)`

Restituisce una matrice contenenti gli indici degli elementi non nulli del vettore A.

`[u,v,w] = find(A)`

Restituisce i vettori u e w contenteni l'indice di riga e di colonna degli elementi non nulli della matrice A e la matrice w (opzionale) contenente i valori degli elementi non nulli

La funzione `find` e gli operatori logici

Vediamo la sessione:

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

Notate che `find` ritorna gli *indici* e non i *valori* degli array!!!

(continua …)

La funzione `find` e gli operatori logici

Nella seguente sessione notate la differenza tra il risultato di `y(x&y)` e quello di `find(x&y)` nel lucido precedente.

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


- Altri operatori logici
  - Estensione di **|** e & ai vettori:
    any(X) all(X)
  - Per verificare la dimensione:
    isempty(A)
  - Per verificare il valore:
    isinf(A) isnan(A) finite(A)
  - Per verificare il tipo di dati:
    ischar(A) isnumeric(A) isreal(A)

Funzione logica	Definizione
`all(x)`	Ritorna uno scalare: 1 se tutti gli elementi del vettore `x` sono non nulli, 0 altrimenti.
`all(A)`	Ritorna un vettore riga, con lo stesso numero di colonne della matrice `A`, che contiene 1 o 0, in funzione del fatto che la corrispondente colonna di `A` contenga tutti elementi non nulli oppure no.
`any(x)`	Ritorna uno scalare: 1 se almeno un elemento del vettore `x` è non nullo, 0 altrimenti.
`any(A)`	Ritorna un vettore riga, con lo stesso numero di colonne della matrice `A`, che contiene 1 o 0, in funzione del fatto che la corrispondente colonna di `A` contenga almeno un elemento non nullo oppure no.
`isempty(A)`	Ritorna 1 se `A` è vuoto, 0 altirmenti.

Funzione logica	Definizione
`isinf(A)`	Ritorna un vettore (o matrice) delle stesse dimensioni di `A` con1 dove gli elementi di `A` sono `'inf'`, 0 altrove.
`isnan(A)`	Ritorna un vettore (o matrice) delle stesse dimensioni di `A` con1 dove gli elementi di `A` sono `'NaN'`, 0 altrove. (`'NaN'` sta per "not a number," cioè risultato indefinito)
`finite(A)`	Ritorna un vettore (o matrice) delle stesse dimensioni di `A`, con 1 dove gli elementi di `A` sono finiti, 0 altrove.

Funzione logica	Definizione
`ischar(A)`	Ritorna 1 se A è un array di caratteri, 0 altrimenti.
`isnumeric(A)`	Ritorna 1 se A è un vettore numerico, 0 altrimenti.
`isreal(A)`	Ritorna 1 se A ha solo elementi con parte immaginaria nulla, 0 altrimenti.

Appendice: Vettori di celle (precisazioni)

- Vengono usati di solito per contenere stringhe di diversa lunghezza.
- Si usano le parentesi graffe per crearli

```
>> nomi={'tim','tom','carl'}
nomi =
    'tim'    'tom'    'carl'
```


Possono anche essere matrici di celle

```
>> nomi={'tim','smith'
'tom','jones'
'carl','lee'}

nomi =

    'tim'     'smith'
    'tom'     'jones'
    'carl'    'lee'
```

L'indicizzazione è come per le matrici:

```
>> nomi(1,2)

ans =

    'smith'
```


N.B. Quello che otteniamo dall'indicizzazione è ancora una cella, non una stringa!

```
>> x=nomi(1,2)

x =

    'smith'

>> x=='smith'
??? Error using ==> ==
Function '==' is not defined for values of class 'cell'.
```

Si converte in stringa con la funzione char!!!

```
>> char(x)=='smith'

ans =

     1     1     1     1     1
```

L'abbiamo usato anche nella funzione fprintf!

Per concatenare si usano le parentesi quadre, come per i vettori!

```
>> nomi={'tim','tom','carl'}

nomi =

    'tim'    'tom'    'carl'

>> [nomi,'john']
??? Error using ==> horzcat
Conversion to cell from char is not possible.

>> [nomi,{'john'}]

ans =

    'tim'    'tom'    'carl'    'john'
```

Dobiamo mettere le graffe attorno al nuovo nome, per convertirlo in cella!

Per trovare un nome in un elenco è molto comoda la strcmp!

```
>> nomi={'tim','tom','carl'}

nomi =

    'tim'    'tom'    'carl'

>> strcmp(nomi,'tom')

ans =

     0     1     0

>> nomi(ans)

ans =

    'tom'
```

Possiamo usare questo Vettore logico per Indirizzare….