# Matlab. Istruzioni condizionali, cicli for e cicli while.

Alvise Sommariva

Università degli Studi di Padova
Dipartimento di Matematica

17 marzo 2016

# Introduzione

Il proposito di questa seconda lezione è mostrare le istruzioni condizionali, i cicli for e i cicli while in Matlab.

L'importanza di queste istruzioni è sottolineata dal teorema di Böhm-Jacopini (1966), che afferma che qualunque algoritmo può essere implementato in fase di programmazione utilizzando tre sole strutture di controllo

- sequenza;
- selezione (`if ... then ... else`);
- ciclo o iterazione (`while ...do`);

da utilizzare ricorsivamente alla composizione di operazioni elementari.

# Istruzione condizionale semplice

La struttura condizionale semplice è del tipo

```
if  <espressione logica e' verificata>
      <esegui processo>
end
```

Vediamone un esempio, digitando nel workspace

```
>> a=5;
>> stringa='segno da determinare';
>> if a > 0
stringa='numero positivo';
end
>> stringa

stringa =
numero positivo
>>
```

# Istruzione condizionale semplice

Nel caso la variabile "a" sia negativa o nulla, non viene eseguito alcun processo. Infatti

```
>> a=-5;
>> stringa='segno da determinare';
>> if a > 0
stringa='numero positivo';
end
>> stringa

stringa =
segno da determinare
>>
```

# Istruzione condizionale alternativa

La struttura condizionale alternativa è del tipo

```
if  <espressione logica e' verificata>
      <esegui processo 1>
      else
      <esegui processo 2>
end
```

Vediamone un esempio, digitando nel workspace una modifica del processo precedente, che stabilisca il segno nel caso $a \leq 0$.

```
>> a=-5;
>> if a > 0
stringa='numero positivo';
else
stringa='numero non positivo';
end
>> stringa
stringa =
numero non positivo
>>
```

# Istruzione condizionale multipla

La struttura condizionale multipla, sfrutta il fatto che nella struttura condizionale alternativa, si possano utilizzare nuovamente istruzioni condizionali (semplici o multiple), come ad esempio

```
if  <espressione logica 1 e' verificata>
      <esegui processo 1>
      else
          if  <espressione logica 2 e' verificata>
              <esegui processo 2>
             else
              <esegui processo 3>
          end
end
```

# Istruzione condizionale multipla

Vediamone un esempio, digitando nel workspace una modifica del processo precedente, che stabilisca il segno nel caso $a < 0$, $a > 0$ e $a = 0$.

```
>> a=0;
>> if a > 0
    stringa='numero positivo';
else
    if a < 0
        stringa='numero negativo';
    else
        stringa='numero nullo';
    end
end
>> stringa
stringa =
numero nullo
>>
```

# Espressioni logiche

Nel descrivere gli asserti, servono le cosiddette espressioni logiche.

Negli esempi precedenti, le espressioni logiche erano "a > 0" e "a < 0".

Di seguito listiamo simboli che permettono di eseguire espressioni logiche (operatori di relazione).

| | |
|---|---|
| == | uguale |
| ~= | non uguale |
| < | minore |
| > | maggiore |
| <= | minore uguale |
| >= | maggiore uguale |

# Espressioni logiche

Più espressioni logiche *semplici* possono essere utilizzate per comporre espressioni logiche più *complesse*.

In questo ambito necessitano i seguenti simboli (operatori logici):

| | |
|---|---|
| && | and |
| \|\| | or |
| ∽ | not |
| & | and (componente per componente) |
| \| | or (componente per componente) |

# Espressioni logiche

In Matlab, se un'espressione logica è

- verificata allora assume valore 1;
- non è verificata allora assume valore 0.

Vediamone degli esempi.

```
>> 1 == 1 % 1 e' uguale a 1?
ans =
     1
>> 0 == 1 % 0 e' uguale a 1?
ans =
     0
>> 1 ~= 0 % 1 non e' uguale a 0?
ans =
     1
>>
```

# Il commando switch

A volte risulta importante eseguire processi diversi al variare del valore di una variabile. Per rendere il codice più comprensibile, in molti linguaggi si introduce l'istruzione case of. In Matlab/Octave essa è rappresentata mediante il comando switch.

```
switch <espressione switch>
          case <valore 1>
            <processo 1>
          case <valore 2>
            <processo 2>
            . . . . . .
          otherwise
             <processo altrimenti>
end
```

# Il commando `switch`

Vediamone un esempio, ricordando che la funzione `sign(x)` vale

- 1 se $x > 0$,
- $-1$ se $x < 0$ ,
- 0 altrimenti (cioè se $x = 0$).

```
>> a=1; s=sign(a);
>> % s = 1 se a > 0, s = -1 se a < 0, s = 0 se a = 0.
>> switch s
case 1
   stringa='a > 0';
case -1
   stringa='a < 0';
otherwise
   stringa='a = 0';
end
>> stringa
stringa =
a > 0
>>
```

# Il ciclo `for`

Il *ciclo for* itera una porzione di codice, al variare di certi indici.

```
for variabile = vettore
  <processo dipendente dal valore di variabile>
end
```

Vediamone un esempio.

```
>> s=0;
for j=1:10
    s=s+j;
end
```

Passo passo, la variabile $j$ assume il valore 1 ed $s = s + j = 0 + 1 = 1$. Successivamente $j$ assume il valore 2 ed $s$ che precedentemente valeva 1, ora essendo $s = s + j = 1 + 2$ vale 3. Si itera il processo fino a che $j = 10$ (incluso) e alla fine $s = 55$. In effetti, la somma dei primi $n$ numeri interi positivi vale $n \cdot (n+1)/2$ che nel nostro caso è proprio 55.

# Il ciclo `while`

Il *ciclo while* itera un processo finchè una espressione logica è verificata.

```
while <espressione logica>
     <processo>
end
```

La differenza più palese tra "ciclo `for`" e "ciclo `while`" consiste nel fatto che

- il ciclo `for` è comunemente utilizzato quando è noto il numero di volte in cui compiere il ciclo, e a differenza del `while` non necessita di incremento dell'indice.
- il ciclo `while` può essere utilizzato anche quando non è noto il numero di volte in cui compiere il ciclo.

# Il ciclo `while`

Vediamo un esempio.

```
s=1;
while s > 0
    r=rand(1)
    s=rand(1)-0.5
end
```

In questo caso, il ciclo while si interrompe quando il numero random (casuale) "`rand`" è inferiore o uguale a 0.5, in quanto la variabile "`s`" diventa non positiva.
Una tipica esecuzione è

```
r =  0.56233
s =  0.062326 % PRIMA ITERAZIONE WHILE TERMINATA.
r =  0.89838
s =  0.39838  % SECONDA ITERAZIONE WHILE TERMINATA.
r =  0.44295
s = -0.057050 % TERMINAZIONE CICLO WHILE
```

# Il ciclo `while`

Vediamo un altro esempio.

```
>> s=0;
>> j=1;
>> while j <= 10
       s=s+j; j=j+1;
end
>> s
s =
    55
>>
```

In questo caso, si itera finchè $j$ è strettamente minore o uguale di 10, dovendo essere il test $j <= 10$ verificato.

Quindi l'ultimo $j$ sommato a $s$ è 10 ed è per questo che la somma vale 55, riproducendo i risultati visti in un precedente esempio via `ciclo for`.

# Il ciclo `while`

**Nota.**

*Attenzione ai cicli infiniti. Il ciclo si ripete quando la condizione assume valore "`true`". Se assume sempre valore "`true`" continua infinite volte!*

*Vediamo un esempio di ciclo infinito.*

```
>> s=0;
>> while s >= 0
       s=s+1;
end
```

# Esercizio

Si effettui un programma Matlab/Octave che risolva l'equazione di secondo grado

$$ax^2 + bx + c = 0,$$

utilizzando nel caso $a \neq 0$

- $x_{1,2} = \frac{-b \pm \sqrt{b^2-4ac}}{2a}$ (algoritmo instabile);
- $x_1 = \frac{-b - sign(b)\sqrt{b^2-4ac}}{2a}$ e $x_2 = \frac{c}{ax_1}$ (algoritmo stabile);

pur discutendo separatamente i casi in cui $a = 0$ e/o $b = 0$ e/o $c = 0$.