for loop

A `for` loop repeats a group of commands a fixed, predetermined number of times. A `for` loop has the following structure:

```
for   variable=expression
      commands
end
```

The commands between the `for` and `end` statements are executed once for every column in the expression, beginning with the first column and stepping through to the last column. At each step, known as an iteration, the appropriate column of the expression is assigned to the variable. Thus, on step n, column n of the expression is assigned to the variable, which then can be operated on by one of the commands in the loop.

Regole per scrivere ed utilizzare un ciclo for:

- **Se *expression* è una matrice vuota: il ciclo non si esegue**
- **Se il risultato di *expression* è un vettore: si esegue il ciclo con il valore della variabile relativo a ciascun elemento del vettore**
- **Non si può modificare *variable* nel ciclo**
- **Alla fine del ciclo, *variable* ha l'ultimo valore assegnato**
- **Si può utilizzare l'operatore '*:*' per definire l'*expression*, con il seguente formato: *for index= initial:increment:limit***

esempi

```
% esempi di utuilizzo del comando "for - end“

% primo esempio
dati = 1:2:10;
for contatore = dati
    disp([‘cont =  ' num2str(contatore)]);
    Matrix(contatore) = contatore*2;
end
Matrix


%secondo esempio
for cont = 1:2:10
    disp([‘ cont = ' num2str(cont)]);
    M(cont) = cont*2;
end
M
```

N.B. : evitare di utilizzare cicli:

è quasi sempre possibile ed a volte è decisamente conveniente!!!

Esempio: (metodo meno conveniente)

```
% uso del ciclo for
clear
disp('calcolo del seno di t  ');
tmax = input('tempo massimo t: ');

tic;
for t= 1:tmax
    y(t)= sin(2*pi*t);
end
tempo = toc;

disp([ 'l''elaborazione e'' durata:  ', num2str(tempo) ' sec.'])
```

Esempio: (metodo alternativo)

```
% uso del  ciclo for
% con inizializzazione del vettore

clear
disp('calcolo del seno di t con inizializzazione di y ');
tmax = input('tempo massimo t: ');

tic;
t= 1:tmax;      %inizializzazione del vettore che verrà utilizzato nel ciclo
y = zeros(1,tmax);
for t=1:tmax
   y = sin(2*t*pi);
end
tempo = toc;

disp([ 'l''elaborazione e'' durata:  ', num2str(tempo) ' sec.'])
```

Esempio: (metodo migliore)

```
% senza utilizzo del ciclo for

clear
disp('calcolo del seno di t senza il "for" ');
tmax = input('tempo massimo t: ');

tic;
t= 1:tmax;
y = sin(2*t*pi);
tempo = toc;

disp([ 'l''elaborazione e'' durata:  ', num2str(tempo) ' sec.'])
```

while loop

A `while` loop repeats a group of commands as long as a specified condition is true. A `while` loop has the following structure:

```
while   expression
        commands
end
```

If **all** elements in expression are true, the commands between the `while` and `end` statements are executed. The expression is reevaluated and if all elements are still true, the commands are executed again. If any element in the expression is false, control skips to the statement following the `end`

- **E' necessario modificare nel ciclo la variabile inclusa in expression**
- **Se il valore di expression non cambia (sempre true, o non-zero) il ciclo diventa un ciclo infinito (infinite loop)**

esempio

```
% calcolo del valore di eps: il più piccolo numero che aggiunto a 1
% dà un risultato >1 con la precisione finita


cont = 0;      % contatore di iterazioni
EPS = 1;

while (1+EPS) > 1
    EPS = EPS/2;
    cont = cont + 1;
end

cont = cont - 1
EPS = 2*EPS
```

while loop e break

- **Il ciclo while permette l'esecuzione di una serie di comandi un numero indefinito di volte (finchè expression è true o non-zero).**
- **Il comando break permette di uscire dal ciclo while, anche se la condizione di expression è true (o non-zero)**

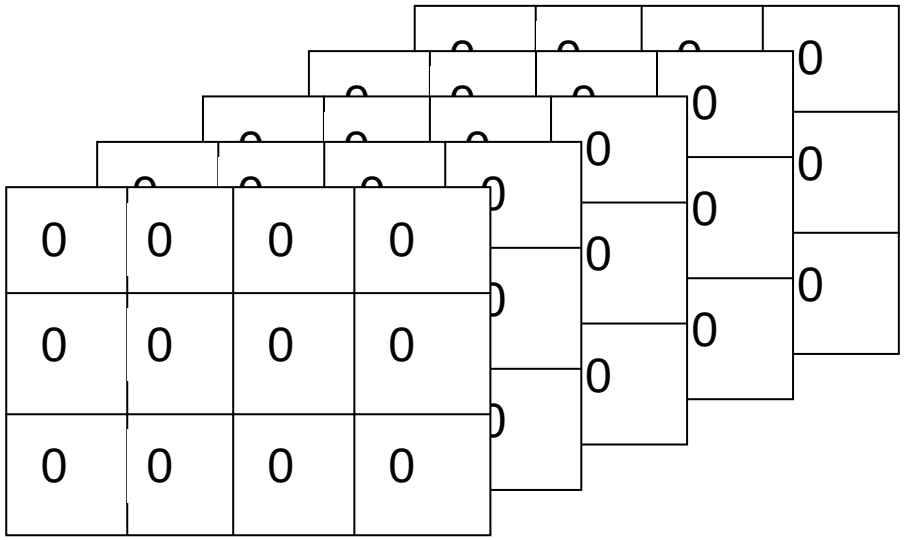
esempio

```
% esempio di utilizzo del comando "break"
% per uscire da un ciclo while
% prima del verificarsi della condizione

count = 0;
a = input('Valore di a: ');

while count<100
    a = a-2;
    if a < 0
        break
    end
    count = count + 1;
end

disp (['a = ' num2str(a) '     count = ' num2str(count)]);
```

Arrays multidimensionali:

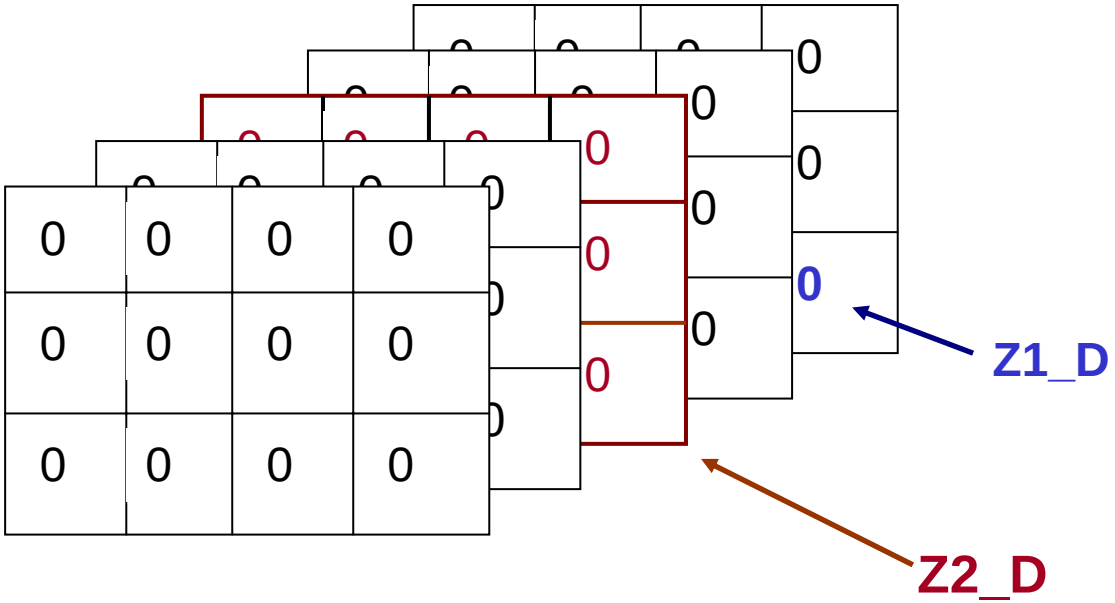
Sono arrays con un numero di indici >2:

```
Z3_D = zeros(3,4,5)     % Z3_D è una matrice tri-dimensionale, 3x4x5
                        % i cui elementi valgono 0
```



Array tri-tidimensionale

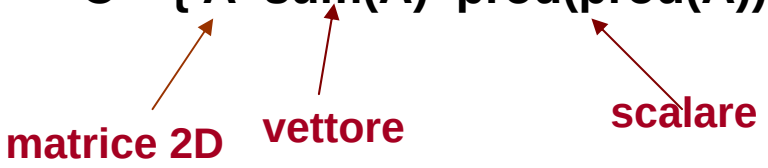
```
Z3_D = zeros(3,4,5)     % Z3_D è una matrice tri-dimensionale, 3x4x5
                        % i cui elementi valgono 0
Z2_D  = Z3_D(:,:,3)     %  Z2_D  è una matrice bi-dimensionale, 3x4,
                        %  ottenuta da Z3_D
Z1_D = Z3_D( end,end,end)  % Z1_D è uno scalare ottenuto da Z3_D
```



Cell array:

- **sono arrays i cui elementi sono copie di altri arrays**
- **un cell-array è creato mettendo insieme (tra parentesi graffe) più variabili:**

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

C = { A  sum(A)  prod(prod(A)) }
```



C è un cell-array di 1x3 elementi, i 3 elementi sono di dimensioni diverse:

I cell-arrays possono essere utilizzati per memorizzare sequenze di matrici di dimensioni diverse

- **si può creare un cell-array vuoto con la funzione *cell***

Cell array:

Come indirizzare elementi di un cell array:

```
A1 = C{1}     % ottengo una matrice 3x3, il primo elemento di C
P = C{3}      % ottengo uno scalare, il terzo elemento di C
```