Corso di Elaborazione dei Segnali e Immagini - A.A. 2022/2023

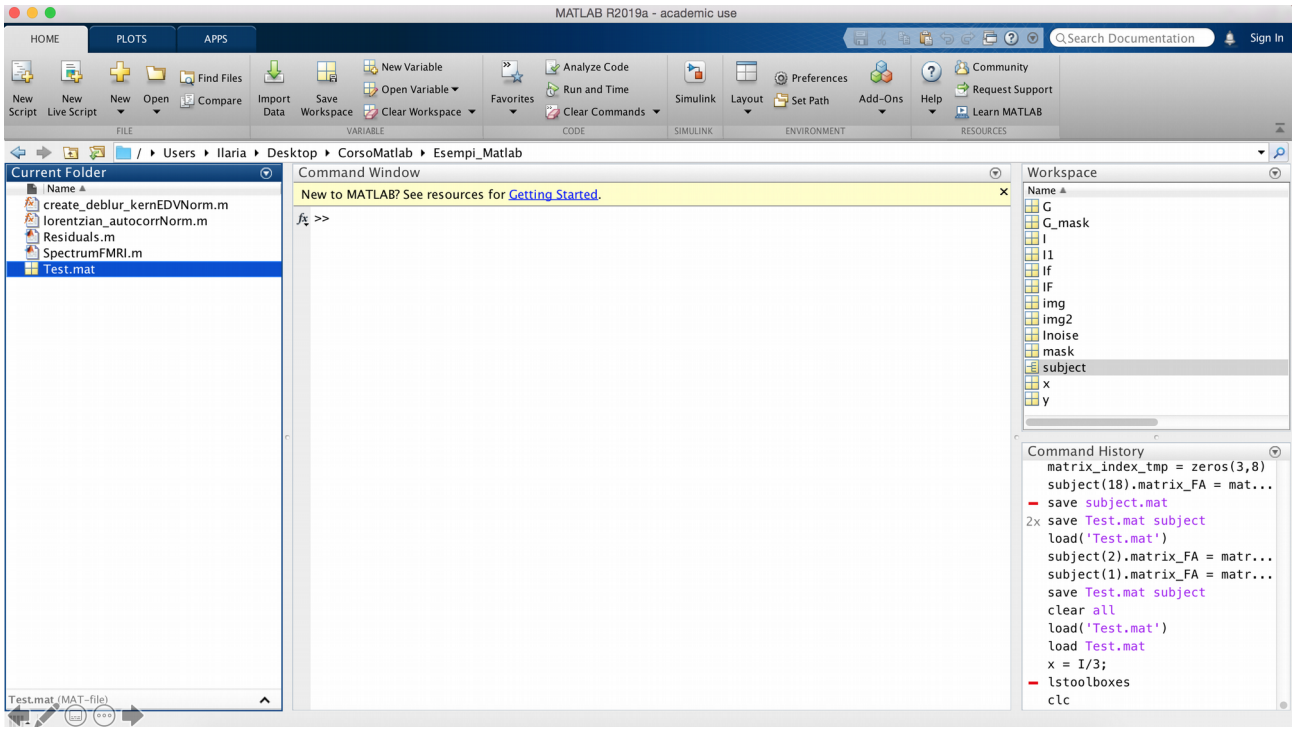
LAB 1 - MATLAB

Queste note presentano alcune caratteristiche di base del programma, e sono state ispirate dal tutorial di Alessandro Rivola (http://diem1.ing.unibo.it/mechmach/rivola/).
Un'altra ottima fonte per impratichirsi con MATLAB, oltre ai link segnalati nell'Introduzione al corso, è il canale ufficiale di training della Mathworks: https://uk.mathworks.com/help/pdf_doc/matlab/matlab/matlab/index.html
Aspetti avanzati di MATLAB (celle, analisi a basso livello dell'ambiente grafico, funzioni inline, programmazione parallela, porting) verranno esplorati gradualmente durante il corso, limitatamente al tempo a disposizione.

INTRODUZIONE A MATLAB - PARTE 1

1. INTRODUZIONE

Per lanciare la finestra di MATLAB occorre cliccare sulla relativa icona.



Il layout di MATLAB è costituito da quattro sezioni principali:

1) **Command Window:** finestra principale interattiva, contiene il comando prompt (»). È la finestra che mi permette di interagire con MATLAB stesso, ad esempio è utilizzata per inserire comandi e stampare a video i risultati;

2) **Command History:** visualizza una lista dei comandi precedentemente digitati. La cronologia rimane per più sessioni, i comandi possono essere caricati direttamente sulla Command Window ed eventualmente modificati, oppure double-clicked per rieseguirli;

3) **Workspace:** contiene la lista di tutte le variabili generate nella sessione corrente, insieme alle loro informazioni principali;

4) **Current Directory:** mostra tutti i files che sono contenuti nella directory di lavoro, anche suddivisi per tipologia.

MATLAB può essere inizialmente utilizzato per eseguire calcoli di base, come quelli che possono essere eseguiti in una calcolatrice (scientifica). Calcoli semplici con operazioni aritmetiche, esponenziali e numeri complessi possono essere inseriti direttamente nella Command Window.
L'aspetto principale del programma è la semplicità concettuale con cui vengono rappresentati i dati, che vengono introdotti nel programma in maniera molto semplice, mediante assegnamento. Le istruzioni sono frequentemente nella forma: *variabile = espressione* (l'ordine è essenziale!).
Una *variabile* è un nome simbolico associato ad un valore, sono molto importanti in quanto permettono di riferirsi facilmente a dei dati che possono essere complessi e possono cambiare. Tutte le variabili sono matrici (importante!).
Un'*espressione* può comprendere variabili, operatori, funzioni e caratteri speciali, e una volta valutata produce un certo risultato.
Ad esempio, nella Command Window, con l'istruzione:

```
>> a = 4
```

definiamo la variabile `a` assegnandole il valore `4`. Occorre notare che il programma ribadisce il risultato della istruzione precedente, visualizzandolo sullo schermo.
Il programma non richiede definizioni particolari di tipo durante l'inizializzazione di variabili, ma il tipo viene assegnato automaticamente in funzione del dato inserito.
Ad esempio, l'istruzione:

```
>> b =4+5i
```

definisce ed inizializza la variabile `b` al valore complesso `4+5i`.
MATLAB è case sensitive, quindi le variabili `name`, `Name` e `NAME` sono diverse.
Se il nome della variabile e il segno = sono omessi, MATLAB crea automaticamente la variabile `ans` ("risposta", ovvero la variabile che contiene il risultato dell'elaborazione richiesta), ad esempio l'istruzione:

```
>> 1900/81
```

produce come risultato:

```
ans =
    23.4568
```

Volendo conservare tale risultato si può scrivere:

```
>> res = ans
```

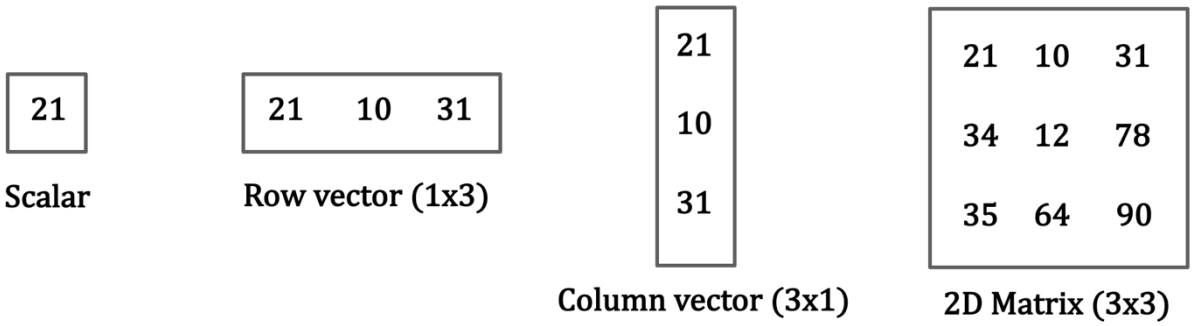
Di default il risultato è stampato nella Command Window, ma l'output può essere soppresso terminando l'espressione con il punto e virgola (;).

L'elenco delle variabili in memoria durante la sessione di lavoro di MATLAB si ritrova nel Workspace, con i relativi valori e tipi. In alternativa, in ogni momento si

può visualizzare quali sono le variabili disponibili usando i comandi `who` oppure `whos` (informazioni più dettagliate).
Altri comandi utili sono: `clear all` (per cancellare tutte le variabili) oppure `clear variable`, ad esempio `clear a` (per cancellare selettivamente).

2) DEFINIZIONE DI MATRICI E VETTORI

MATLAB lavora sempre con matrici. Gli *scalari* sono trattati come matrici 1x1.
I vettori sono particolari matrici con una sola riga e n colonne (*vettore riga*), oppure n righe e una sola colonna (*vettore colonna*). MATLAB alloca automaticamente le matrici in memoria pertanto non è necessario dichiararne le dimensioni.



Le matrici si inseriscono come segue:

- separando le colonne con virgole o spazi
- separando le righe con punto e virgola
- delimitando la matrice tra parentesi quadre

Ad esempio, l'istruzione:

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

produce il risultato:

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

I vettori sono introducibili in modo analogo a quanto fatto per le matrici. Ad esempio:

```
>> vettore_riga = [3 5 6]
```

```
>> vettore_colonna = [1;2;56]
```

Gli elementi di matrici e vettori possono essere espressioni di MATLAB, ad esempio l'istruzione:

```
>> x = [-1.3 (4*2)/3  (1+2+3)*4/5]
```

produce il risultato:

```
x =
    -1.3000    2.6667    4.8000
```

Una volta definite, si possono estrarre singoli valori o sotto-matrici. Ricordiamo che per effettuare gli accessi ad una matrice, prima devono essere indicate le righe, e poi le colonne, ad esempio in una matrice di dimensioni 3x3:

$$A = \begin{pmatrix} (1,1) & (1,2) & (1,3) \\ (2,1) & (2,2) & (2,3) \\ (3,1) & (3,2) & (3,3) \end{pmatrix}$$ **2D Matrix, 3x3 size**

Possiamo quindi definire per vettori/matrici:

`>> x(3)`	Estrae il valore corrispondente in posizione = 3
`>> A(2,3)`	Estrae il valore corrispondente alla riga = 2, colonna = 3
`>> A(:,1)`	Estrae tutti i valori nella colonna = 1
`>> A(2,:)`	Estrae tutti i valori nella riga = 2
`>> A(2:3,1:2)`	Estrae una sotto-matrice

I due punti indicano che l'intera riga o colonna vengono considerati, ad esempio nel caso limite `A(:,:)` coincide con la matrice `A` stessa.
Per accedere ad una componente di un vettore si può utilizzare un solo indice; tale modo di procedere può essere applicato anche alle matrici e risulta molto potente in quanto consente di eseguire operazioni anche complesse in una sola istruzione di codice. Ad esempio, digitando:

```
>> A(6)
    ans =
        8
```

Vale a dire che si è acceduti all'elemento di posizione (3,2). Nell'accesso ad un solo indice, MATLAB fornisce come risposta l'elemento che si ottiene contando a partire da quello di posto (1,1) scendendo per le diverse righe a partire dalla prima.
In pratica, in una matrice mxn per accedere all'elemento di posto `(i,j)`, basta usare l'indice `l = i+(j-1)*m`.

Le matrici possono essere costruite a partire da matrici di dimensioni minori. Ad esempio, l'istruzione:

```
>> x(5) = abs(x(1))
```

produce il risultato:

```
x =
  -1.3000    2.6667    4.8000       0    1.3000
```

Come si vede dall'esempio, la dimensione di `x` viene automaticamente incrementata in modo da inglobare il nuovo elemento e gli elementi indefiniti (in questo caso il quarto) viene posto uguale a zero. Per aggiungere elementi in matrice, posso ad esempio eseguire:

```
>> r = [10 11 12];
>> A = [A;r]
```

che produce come risultato:

```
A =
     1     2     3
     4     5     6
     7     8     9
    10    11    12
```

I valori contenuti in una matrice possono essere modificati, ad esempio:

```
>> A(1,4) = 100          Sostituisce un singolo valore
>> A(2,:) = 0            Sostituisce una intera riga
>> A(:,3) = [1 4 5 2]    Sostituisce una intera colonna
```

Per creare un vettore di punti equi-spaziati, anche di grandi dimensioni, posso usare l'istruzione `x = [inizio:passo:fine]`, ad esempio:

```
>> x = [1:4]      produce il risultato x = 1 2 3 4
>> x = [1:3:8]    produce il risultato x = 1 4 7
```

In generale:
`j:k` produce il risultato `j j+1 j+2 ... k`
`j:i:k` produce il risultato `j j+i j+2i ... k`

In alternativa, posso usare il comando `linspace`, per un vettore di numeri linearmente equispaziati, oppure `logspace` se voglio valori equispaziati logaritmicamente, usando la forma `nome_comando(inizio, fine, numero campioni)`:

```
>> z = linspace(2,14,6)
    z =
        2.0000    4.4000    6.8000    9.2000
        11.6000  14.0000

>> k = logspace(1,2,7)
    k =
        10.0000     14.6780     21.5443     31.6228
        46.4159     68.1292    100.0000
```

In MATLAB, sono implementate varie funzioni che permettono di definire in maniera semplice delle matrici particolari, tra queste ricordiamo:

- `zeros(m,n)` → permette di creare una matrice di zeri di ordine mxn:

```
>> Z = zeros(3,5)
  Z =
     0  0  0  0  0
     0  0  0  0  0
     0  0  0  0  0
```

Per creare una matrice di zeri quadrata di ordine n basta scrivere `Z = zeros(n);`

- `ones(m,n)` → permette di creare una matrice di uni di ordine mxn:

```
>> O = ones(2,4)
   O =
     1  1  1  1
     1  1  1  1
```

Per creare una matrice quadrata si procede come per la precedente funzione. Per creare una matrice piena di un certo scalare, basta scrivere `alpha*ones(m,n)`, con `alpha = costante`

- `eye(n)` → permette di creare una matrice identità di ordine n:

```
>> I = eye(3)
   I =
     1  0  0
     0  1  0
     0  0  1
```

- `rand(m,n)` / `randn(m,n)` → permette di generare matrici o vettori i cui elementi sono distribuiti randomicamente secondo una distribuzione uniforme (`rand`) e gaussiana o normale (`randn`). Il loro utilizzo è del tutto simile a quello delle funzioni precedenti.

```
>> I = rand(3)
```

```
I =
    0.6557    0.9340    0.7431
    0.0357    0.6787    0.3922
    0.8491    0.7577    0.6555
```

Per verificare la dimensione di una matrice o il numero di elementi inclusi, ci sono diverse funzioni, tra cui:

- `size(matrice)` → restituisce le dimensioni della matrice
- `length(matrice)` → restituisce la lunghezza del vettore, nel caso di matrici corrisponde alla dimensione massima (ovvero MAX(SIZE(X)))
- `numel(matrice)` → restituisce il # totale di elementi nella matrice

3) OPERAZIONI TRA MATRICI

Le operazioni fondamentali sono definite tra matrici, che devono avere la stessa dimensione per poter essere eseguite. In caso contrario, MATLAB stampa un messaggio di errore qualora le dimensioni non siano corrette rispetto all'operazione da lanciare. Alcuni esempi di operazioni importanti sono:

```
>> C = A + B    Somma tra matrici
>> C = A - B    Differenza tra matrici
>> C = A / B    Divisione tra matrici
>> C = A * B    Moltiplicazione tra matrici
```

Per valutare una operazione *"termine per termine"*, come moltiplicare ogni termine del vettore `A` per ogni termine del vettore `B`, si usa l'operatore .* Ad esempio:

```
>> A = [1 3 2]
>> B = [-1 2 1]
>> C = A.*B
>> C =
    -1     6     2
```

Lo stesso si applica anche ad altre operazioni, come divisione o elevamento a potenza di una matrice (`C = A.^2`).
L'operazione di *trasposizione* (sia di vettori che di matrici) è l'apice ('). Ad esempio, dato un vettore `v = [1;2;56]`, l'istruzione:

```
>> vt = v'
```

produce il risultato:

```
>> vt =
        1  2  56
```

Se però il vettore (o la matrice) è complesso l'operatore "apice" esegue la trasposta coniugata. La semplice trasposizione si ottiene con l'operatore "punto apice" (`.'`):

```
>> vt = v.'
```

Altre operazioni matriciali importanti sono:

```
>> B = inv(A)      Inversa di una matrice (quadrata)
>> B = det(A)      Determinante di una matrice (quadrata)
>> B = diag(A)     Diagonale di una matrice
>> B = trace(A)    Somma gli elementi sulla diagonale (mat. quadrata)
>> B = eig(A)      Autovalori di una matrice
>> B = rank(A)     Rango di una matrice
>> B = sum(A)      Somma degli elementi colonna per colonna
>> B = prod(A)     Prodotto degli elementi colonna per colonna
>> B = min(A)      Minimo degli elementi colonna per colonna
>> B = max(A)      Massimo degli elementi colonna per colonna
>> B = mean(A)     Media degli elementi colonna per colonna
```

Ad esempio, data `A = [1 2; 3 4]` possiamo calcolare le operazioni qui sopra.

Nel caso di variabili complesse, in cui MATLAB usa indistintamente i oppure j, ci sono varie funzioni che possono essere utili, come ad esempio:

```
>> z1=2+3*j        Definisce un numero complesso
>> abs(z1)         Valuta il modulo
>> angle(z1)       Valuta la fase
>> real(z1)        Parte reale
>> imag(z1)        Parte immaginaria
```

4) COMANDI UTILI E ISTRUZIONI DOS-LIKE

In MATLAB, ci sono una serie di funzioni di base che possono essere d'aiuto nella programmazione. Per verificare il contenuto della directory di lavoro, basta digitare il comando DOS-like:

```
>> dir
```

e verrà mostrato il contenuto della directory di lavoro.
Per visualizzare la directory corrente di lavoro, si utilizza il comando:

```
>> pwd
```

Altre istruzioni DOS più utilizzate in ambiente MATLAB sono:

`type filename` visualizza il contenuto del file `filename`
`delete filename` elimina il file `filename` dalla directory corrente
`! command` invia l'istruzione `command` al sistema operativo
↑ (freccia in su della tastiera) richiama le istruzioni digitate in precedenza

Inoltre, esiste un help in linea molto utile. Sul *prompt* di MATLAB è sufficiente digitare `help` con il nome del comando specifico. Se non si conosce il nome del comando per eseguire una certa operazione si può tentare con l'istruzione `lookfor`.

Ad esempio: `>> help fft`
visualizza l'help del comando che esegue la trasformata di Fourier
mentre: `>> lookfor interpolation`
è un esempio di ricerca di funzioni per eseguire l'interpolazione

Per avere una spiegazione più dettagliata: `>> doc fft`

Nota = MATLAB ha molte funzioni matematiche già in-built, che possono essere ulteriormente estese con opportuni toolboxes. Per un elenco delle funzioni matematiche elementari si può digitare `help elfun` nella Command Window (ad esempio, tra le altre, `sin(x)` e `cos(x)` che saranno molto importanti nel corso).

Per salvare un'intero workspace oppure variabili individuali possiamo:

- usare l'interfaccia grafica (Home → Variable → Save Workspace);
- digitare nella Command Window il comando `save('filename.mat')` per intero workspace;
- digitare nella Command Window il comando `save('filename.mat', 'variable_names')` per salvare selettivamente delle variabili.

Se necessario, i file .mat prodotti (anche in altre sessioni/da altri utenti) possono essere caricati e riutilizzati in MATLAB attraverso il comando `load('filename.mat')`.

5) SCRIPT e FUNCTION

Invece di inserire i comandi dal prompt di MATLAB è possibile eseguire una serie di comandi scritti in files. Questi files sono chiamati m-files perché hanno estensione .m
Ci sono due tipi di m-files: gli *script files* e i *function files*.
Gli *script files* contengono una serie di comandi per l'interprete di MATLAB. Il set di comandi può essere inserito in un file di testo (ascii) usando un editor (le versioni 5.0 e successive di MATLAB hanno un proprio editor dedicato, consigliato da utilizzare). Ad esempio, se lo script file si chiama *pippo.m*, i comandi che esso contiene si eseguono digitando il nome del file senza estensione (*pippo*) dal prompt di MATLAB, oppure da Editor premendo Run. Attenzione: è necessario che la cartella dove ci troviamo sia la stessa in cui è contenuto il file.
Le istruzioni contenute in uno script file lavorano sulle variabili contenute nel workspace globale. Tutte le variabili utilizzate dallo script file rimangono disponibili una volta terminata l'esecuzione, fino a quando non si cancella tutto quanto.

I *function files* consentono, in ambiente MATLAB, la definizione di funzioni simili a quelle previste nei linguaggi di programmazione standard. Le variabili vengono passate per valore, quindi contengono vari comandi che necessitano di variabili di input per essere eseguiti e producono delle variabili di output. Questi devono essere salvati come file .m aventi lo stesso nome della function stessa, e devono sempre iniziare come:

```
function [output] = nome_function (input)
```

Tutte le variabili definite internamente sono locali, ovvero non vengono mostrate nel workspace tranne la variabile output. Quindi, una function differisce da uno script perché lavora su variabili locali e per il fatto che non accede alle variabili globali. Possono essere chiamate direttamente nel prompt o all'interno di uno script file (considerato come "main file").
Ad esempio, posso creare una function per creare una curva gaussiana, dati alpha e l'asse dei tempi:

```
function g1 = gaussian(alpha,t)

g1 = exp(-alpha*t.^2);
...COMMANDS

end
```

Questa verrà richiamata nello script principale o nella Command Window ad esempio come:

```
>> t = linspace(-30,30,1000);
>> alpha = 0.05;
>> curve = gaussian(alpha,t);
```

I commenti in MATLAB iniziano con il "%" e durano fino a fine della riga. Un doppio commento ha un significato particolare: delimita l'inizio di un blocco di codice che può essere eseguito singolarmente. È una caratteristica molto comoda per la programmazione in MATLAB via script. In alternativa ed altrettanto comodo, si può selezionare con il mouse esattamente la porzione di codice che si vuole far eseguire, mandandola in esecuzione con tasto dx oppure F9 (Windows).

6) ISTRUZIONI DI CONTROLLO

Tra le istruzioni di controllo che ci possono essere utili in MATLAB, ricordiamo:

`for` per ripetere un insieme di istruzioni per un numero predeterminato di iterazioni. Deve terminare con `end`, la sintassi generale:

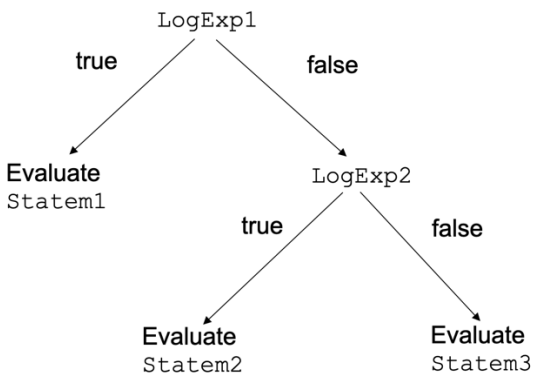
```
for variable = initial:step:final
    statements
end
```

`while` per ripetere un insieme di istruzioni fino a quando una condizione rimane vera. Deve terminare con `end`, la sintassi generale:

```
while condition is true
      statement
end
```

`if` come istruzione condizionale, ovvero esegue un'istruzione soltanto se una certa espressione logica è vera. Deve terminare con `end`. Si possono utilizzare anche else e elseif:

```
if logical_expression1
    statement1
elseif logical_expression2
    statement2
else
    statement3
end
```



Gli operatori relazionali a disposizione, che ritornano 1 se vero, 0 se falso, sono:

MATLAB Operator	Operation	MATLAB Expression
<	Less than	`x<y`
>	Greater than	`x>y`
<=	Less than or equal	`x<=y`
>=	Greater than or equal	`x>=y`
==	Equal	`x==y`
~=	Not equal	`x~=y`

Gli operatori logici a disposizione, che ci permettono ad esempio di valutare contemporaneamente più condizioni, sono invece:

MATLAB Operator	Operation	MATLAB Expression
~	Not	`~x`
&	And	`x&y`
|	Or	`x|y`
xor	exclusive-Or	`xor(x,y)`

x	y	~x	x&y	x|y	xor(x,y)
0	0	1	0	0	0
0	1	1	0	1	1
1	0	0	0	1	1
1	1	0	1	1	0

Esempio 1: ciclo for

```
m = 10;
n = 10;
for i=1:m
    for j=1:n
        A(i,j) = 1/(i+j-1);
    end
end
```

Esempio 2: ciclo while

```
while i < 100
    i = i+1
end
```

Esempio 3: ciclo if

```
a = 10;
max = -Inf;
if a>max
    max = a;
    disp(max)
end
```

Nota 1 = `Inf` rappresenta l'infinito (con segno), mentre `eps` rappresenta il numero più piccolo rappresentabile da MATLAB.
Nota 2 = È possibile effettuare operazioni logiche su matrici, in maniera immediata, ad esempio:

```
>> y = [-10 2 4 -1];
>> y>0
    ans =
        1x4 logical array

         0   1   1   0
```

Oppure:

```
>> y = [-10 2 4 -1];
>> idx = y>0;
>> y_pos = y(idx)
    y_pos =

            2     4
```