

Introduzione a Matlab

Credits to A. Marchesi and F. Trovò

MATLAB

- **MATLAB** (MATrix LABoratory)
 - Ambiente di sviluppo e linguaggio di programmazione per il calcolo numerico
 - Ottimizzato per operare su matrici (ma include anche altre funzionalità matematiche)
- MATLAB è uno strumento commerciale, su licenza NON gratuita
 - Licenza fornita agli studenti dal Politecnico
 - <https://www.software.polimi.it/mathworks-matlab/>



Analisi dei dati



Sviluppo di algoritmi



Creazione di modelli e applicazioni

Octave

- **Octave** è una soluzione alternativa sviluppata dall'universo open-source GNU
- Identico nella concezione e molto simile nell'utilizzo
- Disponibile gratuitamente



Scientific Programming Language

- Powerful mathematics-oriented syntax with built-in 2D/3D plotting and visualization tools
- Free software, runs on GNU/Linux, macOS, BSD, and Microsoft Windows
- Drop-in compatible with many Matlab scripts

Caratteristiche di Matlab

- Linguaggio di alto livello
 - Abbastanza simile a linguaggio C
 - Possiede comandi sintetici per effettuare complesse elaborazioni numeriche
- **Linguaggio interpretato**, comandi e istruzioni
 - NON sono tradotte in codice direttamente eseguibile dalla macchina
 - Matlab invia istruzioni a un **interprete** (un altro programma) che li analizza ed esegue i comandi descritti nelle istruzioni
- Confronto MATLAB e C:
<https://www.geeksforgeeks.org/difference-between-matlab-and-c-language/>

Linguaggi interpretati

- L'interprete esegue le istruzioni contenute nel codice sorgente
- I programmi sono meno efficienti e meno portabili di quelli compilati
- Lo sviluppo è più facile
 - È possibile eseguire le istruzioni mentre si scrive il codice

Linguaggi dinamici

- Matlab è un **linguaggio dinamico** (non tipizzato)
 - Non è necessario dichiarare le variabili
 - Vengono definite automaticamente al primo assegnamento
 - Il tipo delle variabili è dinamico
 - A una variabile si possono assegnare valori di tipo diverso (es., scalari, stringhe, matrici)
 - Il tipo cambia automaticamente in base al valore assegnato

Matlab

- Matlab può far riferimento a tre cose diverse
 - Il linguaggi Matlab utilizzato per codificare i programmi
 - L'interprete Matlab invocato per eseguire i programmi
 - L'ambiente di sviluppo integrato (IDE) che permette di scrivere ed eseguire i programmi

Istruzioni e command window

- Le istruzioni possono essere inviate direttamente all'interprete se scritte nella command window (dopo il simbolo >>)
- Esempio di esecuzione di operazioni aritmetiche

```
>> 5 + 7
```

```
Ans = 12
```

```
>> 5 / 7
```

```
Ans = 0.7143
```

- **ans** è una variabile che contiene il risultato dell'ultima istruzione che non è un assegnamento

Altre istruzioni

```
>> 5 * 6
```

```
Ans = 30
```

```
>> 3 ^ 4
```

```
Ans = 81
```

```
>> 5 / 8
```

```
Ans = 0.625
```

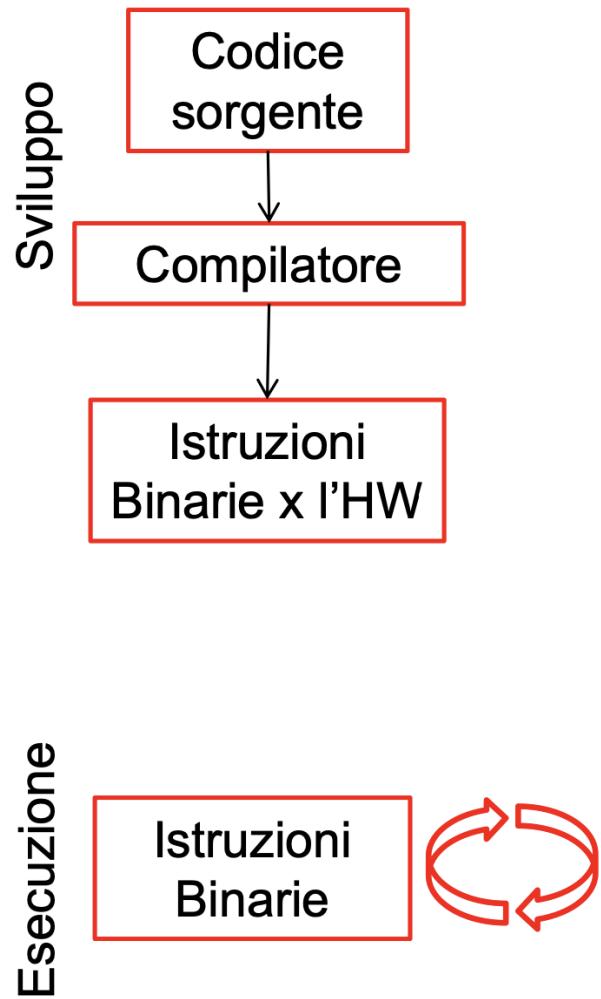
```
>> 'a' + 2
```

```
Ans = 99
```

- I caratteri alfanumerici si indicano con l'apice singolo
- Sono legati agli interi mediante la tabella ASCII

Istruzioni e codice sorgente

- Le istruzioni possono essere contenute in un file sorgente
- Sono eseguite in maniera sequenziale
- L'esecuzione di un codice sorgente può essere visto come l'inserimento sequenziale delle istruzioni della command window



Il terminatore ';'

- Le istruzioni possono terminare con ; ma non è obbligatorio
- Di default, il risultato di ogni istruzione viene visualizzato nella command window
- Terminare un'istruzione con ; blocca la visualizzazione del risultato
 - Maggiore velocità di esecuzione
 - Visualizzazione più compatta
- Inserire sempre il ; a meno che non si voglia ispezionare il valore di una variabile a scopo di debugging

Array (e variabili)

- Le variabili sono create mediante **inizializzazione**, cioè alla prima istruzione in cui compaiono
- Non occorre dichiarare le variabili come in C
- Modi di inizializzazione
 - **Assegnamento**
 - Lettura dati da tastiera
 - Lettura da file

Istruzione di Assegnamento

- Come in C,

NomeVariabile = espressione

- A differenza del C
 - Non si deve dichiarare la variabile **nomeVariabile** precedentemente
 - L'assegnamento comporta una dichiarazione contestuale della variabile **nomeVariabile**
 - È possibile eseguire l'assegnamento tra array
 - Non è richiesto il ; a fine istruzione
 - Il risultato di un'operazione che non comporta un assegnamento viene assegnato alla variabile **ans**

Assegnamento ed inizializzazione

- Quando assegno un valore ad una variabile che non è stata inizializzata, la variabile viene creata

```
>> a = 7
```

```
a = 7
```

- Non è possibile assegnare ad una variabile il valore di una variabile che non esiste

```
>> a = v
```

Undefined function or variable 'v'

- Ottengo un messaggio di errore

Array

- In Matlab tutto è un array
 - **Scalari**: array 1x1
 - **Vettori**: array con una sola riga o colonna
 - **Matrici**: array con due dimensioni
 - **Matrici multidimensionali**: array con più di 2 dimensioni
- Il **tipo** delle variabili è definito dal valore che contengono (definito al momento dell'assegnamento)

Workspace

- Tutte le variabili vengono salvate nel **workspace**, che corrisponde allo spazio di memoria del programma
- E' possibile visualizzare le variabili
 - Il comando **whos** visualizza tutte le variabili
 - Il comando **whos nomeVariabile** visualizza solo nomeVariabile
 - Nel pannello del workspace
- Il comando **clear** pulisce il workspace e rimuove tutte le variabili

```
>> clear
```

```
>> whos
```

```
>>
```

Operazioni

Input	output	Commento
1234/6	ans = 205.6667	calcolo di un valore scalare
a=1234/6	a = 205.6667	assegnamento alla variabile a del risultato di 1234/6
2/5	ans = 0.4	divisione
5/0	ans = inf	divisione per zero
5^2	ans = 25	potenza
1+1==2 1+1~=2	ans = 1 ans = 0	notazione: 1 = vero, 0 = falso, == uguale, ~= diverso

Definizione di vettore

- I vettori sono definiti tra parentesi quadre
 - In un vettore riga gli elementi sono separati da virgole o spazi
 - In un vettore colonna gli elementi sono separati da ; (o andando a capo)
 - Gli indici partono da **1!**

```
>> a = [1 2 3]
```

```
a = 1 2 3
```

```
>> a = [1, 2, 3]
```

```
a = 1 2 3
```

```
>> a = [1; 2; 3]
```

```
a = 1
```

```
2
```

```
3
```

Trasposizione

- L'operatore ' esegue la trasposizione (trasforma vettore riga in vettore colonna e viceversa)

```
>> a = [1, 2, 3]
```

```
a = 1 2 3
```

```
>> a'
```

```
ans = 1
```

```
2
```

```
3
```

Vettori ad incremento regolare

- L'operatore : definisce vettori ad incremento regolare
[inizio : step : fine]
- Definisce un vettore che ha:
 - primo elemento inizio
 - Secondo elemento inizio + step
 - Terzo elemento inizio + 2*step
 - ...
 - Fino al più grande valore inizio + k*step che non supera fine (fine potrebbe non essere incluso)

Vettori ad incremento regolare

- Il valore step può essere qualsiasi, anche negativo
- Se non precisato step vale 1
- Le parentesi [] possono essere omesse

```
>> 1:3
```

```
ans = 1 2 3
```

Modifica di un array

- E' possibile modificare i valori di un array mediante assegnamento
 - Di un singolo elemento (come in C)
 - Di una parte dell'array
 - Di tutto l'array

Assegnamento tra array

- In Matlab è possibile eseguire assegnamenti tra array

```
nomeArray1 = nomeArray2
```

- Copia i valori contenuti in nomeArray2 in nomeArray1

- Non è necessario che entrambi gli array abbiano le stesse dimensioni

```
>> a = [1 2 3];
```

```
>> a = a + 1
```

```
a = 2 3 4
```

Accedere ad un vettore

- Per accedere ad un elemento del vettore
`nomeVettore(indice)`
- Restituisce il valore contenuto in `nomeVettore` alla posizione `indice`
- Come in C, una volta specificato l'indice si accede all'elemento del vettore come ad una variabile (per assegnamento e altre operazioni)
- Differenze da C
 - Si usano le parentesi tonde () invece che quadre []
 - Il primo elemento di `nomeVettore` è alla posizione 1 (**l'indice è sempre positivo**)

Accedere ed assegnamento

- È possibile modificare il valore di un elemento
 - Accedendo all'elemento
 - Assegnando un nuovo valore all'elemento

```
>> a = [1 : 3]
```

```
a = 1 2 3
```

```
>> a(3) = 6
```

```
a = 1 2 6
```

- È possibile eseguire l'assegnamento tra vettori anche quando due vettori hanno dimensioni diverse
 - Il vettore viene ridefinito

```
>> b = [1 : 4]
```

```
b = 1 2 3 4
```

```
>> a = b
```

```
a= 1 2 3 4
```

Accedere ed assegnamento

- Viene segnalato un errore quando si accede ad una posizione non corrispondente ad un elemento dell'array

```
>> a = [1 : 3]
```

```
a = 1 2 3
```

```
>> a(2)
```

```
a = 2
```

```
>> a(4)
```

```
Index exceeds matrix dimensions
```

```
>> a(1.3)
```

```
Subscript indices must either be real positive integers or logicals
```

Accedere ed assegnamento

- È possibile usare una variabile per definire l'indice

```
>> ii = 2;
```

```
>> a(ii)
```

```
ans = 2
```

```
>> a(ii) = a(ii-1) + a(ii+1)
```

```
a = 1 4 3
```

Operazioni aritmetiche

- Le operazioni aritmetiche sono quelle dell'algebra lineare
 - La **somma tra vettori** $c = a + b$ è definita per elemento
 - È possibile solo quando a e b hanno la stessa dimensione

$$c(i) = a(i) + b(i) \quad \forall i$$

- Prodotto tra vettori è il prodotto riga per colonna
- $c = a * b$ restituisce uno scalare

$$c = \sum_i a(i)b(i)$$

- a deve essere vettore riga e b colonna
- Devono avere lo stesso numero di elementi

Eccezione per la somma

- Se sommo un vettore riga ($1 \times n$) e un colonna ($m \times 1$) ottengo una matrice

```
>> a = [1 : 3];  
>> b = [1 : 4]';  
>> a + b  
ans =  
     2 3 4  
     3 4 5  
     4 5 6  
     5 6 7
```

- Replica per un numero di righe opportuno il vettore riga e per un numero di colonne opportuno il vettore colonna

Operazioni puntuale

- È possibile eseguire operazioni puntuale che si applicano ad ogni elemento del vettore separatamente

$c = a.* b$ restituisce $c(i) = a(i) * b(i) \quad \forall i$

$c = a./ b$ restituisce $c(i) = a(i) / b(i) \quad \forall i$

$c = a.^ b$ restituisce $c(i) = a(i) ^ b(i) \quad \forall i$

- Come in algebra lineare, le operazioni tra vettori e scalari corrispondono ad operazioni puntuali
- Se k è uno scalare e b un vettore

$c = k * b$ (o $k.*b$) restituisce $c(i) = k * b(i) \quad \forall i$

Elevamento a potenza

```
>> v1 = [2 3 5 4];  
>> v1^2  
Error using ^  
Inputs must be a scalar and a square matrix.  
To compute elementwise POWER, use POWER (.^)  
instead.
```

- L'elevamento a potenza fa riferimento al prodotto vettoriale ($v1 * v1$)
- Per elevare a potenza ogni singolo elemento

```
>> v1.^2  
ans = 4 9 25 16  
• Equivale a  $v1 .* v1$ 
```

Concatenare i vettori

- L'operatore , e ; permette di concatenare vettori compatibili (entrambi riga o colonna)

```
>> a = [1,2,3]
```

```
a = 1 2 3
```

```
>> b = [a, a + 3, a + 6]
```

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

```
>> b = [a, a+3]
```

```
b = 1 2 3 1 2 3 3
```

- Occhio agli spazi!

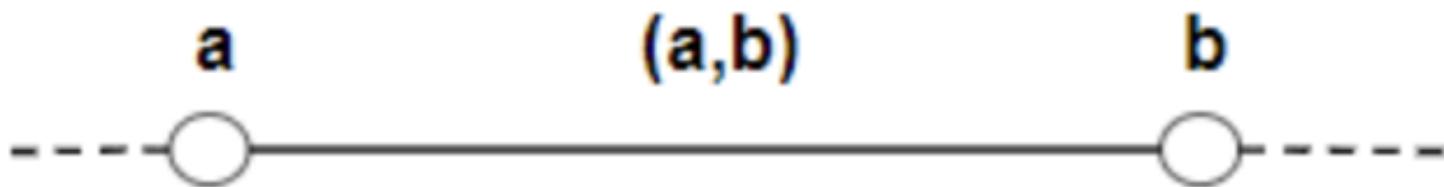
Concatenare i vettori: esempio

```
>> a = [0 7+1]
```

```
a = [0 8]
```

```
b = [a(2) 5 a]
```

```
b = [8 5 0 8]
```



WWW.ANDREAMININI.ORG

Matrici

- Le matrici sono definite affiancando vettori di dimensioni compatibili
 - Usiamo gli operatori , (o spazio) e ; (o vai a capo)
 - L'operatore ' di trasposizione inverte le righe e le colonne della matrice

```
>> a = [1, 2; 3, 4]
```

```
a = 1 2  
      3 4
```

```
>> a'
```

```
a= 1 3  
    2 4
```

Operatore CAT

- La concatenazione avviene mediante operatore CAT
- Richiede dimensioni consistenti dei vettori

```
>> a = [1 : 3]  
a = 1 2 3  
>> b = [4; 5; 6];  
>> A = [a;b]  
Error  
>> A = [a,b]  
Error  
>> A = [a;b']  
A= 1 2 3  
     4 5 6
```

Accedere agli elementi

- Per accedere agli elementi specificare un valore per ogni indice

```
>> nomeMatrice(indice1, indice2)
```

- Seleziona il valore alla riga indice1 della colonna indice2 nella variabile nomeMatrice

```
>> A = [1:3; 4:6; 7:9];
```

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

```
>> A(2,3)
```

```
ans = 6
```

```
>> A(3,5)
```

```
Error
```

Operazioni Aritmetiche

- Array operation: elemento per elemento

```
>> a = [1, 2; 3, 4];
```

```
>> b = [2, 3; 5, 7];
```

```
>> a.*b
```

```
ans = 2 6
```

```
      15 28
```

- Matrix operation: regole dell'algebra lineare

```
>>a*b
```

```
ans = 12 17
```

```
      26 37
```

Divisione sinistra

- Serve per risolvere sistemi di equazioni lineari
- $A x = b$
- La soluzione è $x = A^{-1} b$ oppure $x = A \setminus b$ che corrisponde a calolare la matrice inversa (l'inversa di A rispetto al prodotto matriciale)

Estrarre una riga/colonna

- Per estrarre la seconda riga da una matrice A
riga= A(2, :)
- Per estrarre la terza colonna da una matrice A
colonna= A(:, 3)

Creare matrici

- Esistono alcune funzioni per creare matrici comuni
 - Creare una matrice $A \ n \times n$ in cui tutti gli elementi sono 0

`A=zeros (n)`

- Creare una matrice $A \ n \times m$ in cui tutti gli elementi sono 0

`A=zeros (n, m)`

- Per creare una matrice A in cui tutti gli elementi sono 1

`A=ones (n)`

`A=ones (n, m)`

Creare matrici

- Esistono alcune funzioni per creare matrici comuni
 - Per creare una matrice diagonale

```
>> diag(1,2,3)
```

```
ans = 1 0 0  
      0 2 0  
      0 0 3
```

- Per creare una matrice i cui elementi sono numeri casuali tra 0 e 1

```
rand(n)
```

```
rand(n,m)
```

Creare matrici

- Esistono alcune funzioni per creare matrici comuni
 - Per creare una matrici i cui elementi sono numeri casuali interi tra a e b

`randi ([a,b] , n)`

`randi (([a,b] , [n,m])`

Determinare la dimensione di una matrice

- Per determinare la dimensione di una matrice si può usare size

```
>> A = [1,2,3;4,5,6];
```

```
>> size(A)
```

```
ans = 2 3
```

- Per determinare il numero di righe

```
>> size(A,1)
```

```
ans = 2
```

- Per determinare il numero di colonne

```
>> size(A,2)
```

```
ans = 3
```

Calcolare la media di una matrice

```
>> A = [1, 2, 3; 4, 5, 6];
```

- Per determinare la media di ogni colonna

```
>> mean (A)
```

```
ans = 2.5000      3.5000      4.5000
```

- Per determinare la media di ogni riga

```
>> mean (A, 2)
```

```
ans = 2
```

```
5
```

- Per determinare la media della matrice

```
>> mean (A, "all")
```

```
ans = 3.5000
```

Sommare elementi di una matrice

```
>> A = [1, 2, 3; 4, 5, 6];
```

- Per sommare ogni colonna

```
>> sum(A)
```

```
ans = 5 7 9
```

- Per sommare ogni riga

```
>> sum(A, 2)
```

```
ans = 6
```

```
15
```

- Per determinare la somma di tutti gli elementi della matrice

```
>> sum(A, "all")
```

```
ans = 21
```

Cercare elementi in una matrice

- `find` trova gli elementi di una matrice che soddisfano una condizione

```
>> A = [1, 2; 2, 3];  
>> [i,j] = find(A==2);  
>> i  
i = 2  
     1  
>> j  
j = 1  
     2
```

Strutture controllo e funzioni

Plot di funzioni

- Come visualizzare una funzione $f(x)$ tra a e b
- Esempio con funzione $f(x) = x^2$

```
x = [a:0.01:b]
```

```
y = x.^2
```

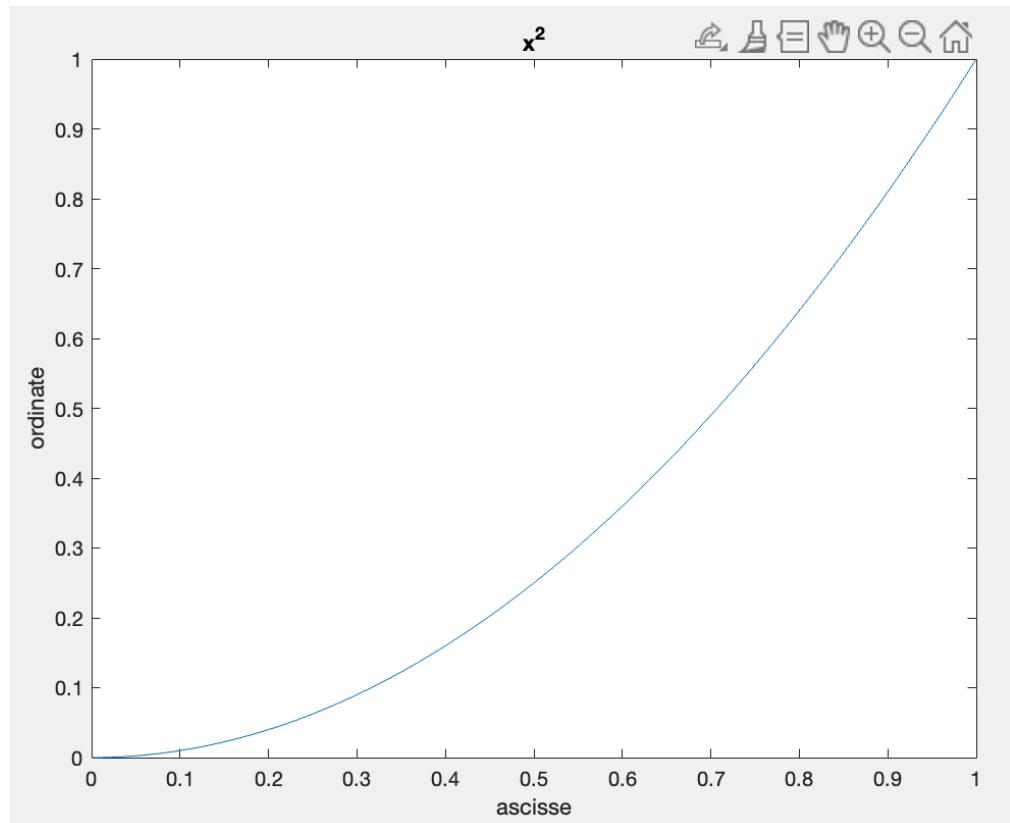
```
figure
```

```
plot(x,y)
```

```
title ('x^2')
```

```
xlabel('ascisse')
```

```
ylabel('ordinate')
```



Tipo di dati logico

- E' un tipo di dato con soli due valori
 - True 1
 - False 0
- I valori di questo tipo possono essere generati
 - Direttamente da due funzioni speciali (true e false)
 - Dagli operatori relazionali
 - Dagli operatori logici

Operatori Relazionali

- Operano su tipi numerici
- Possono confrontare
 - Due scalari
 - Due vettori della stessa dimensione
- Forma generale a OP b
- Possibili operazioni OP: ==, ~=, >, >=, <, <=

3 < 4 true(1)

3==4 false(0)

3~=4 true(1)

Note

- Come in C non confondere == e =
 - == è un operatore di confronto
 - = è un operatore di assegnamento
- La precisione finita può produrre errori

`sin(0) == 0` true (1)

`sin(pi) == 0` false (0)

- Per i numeri piccoli usate una soglia
`abs(sin(pi) - 0) <= eps`

Operatori logici

- Operatori binari
 - AND (`&&`, `&`, `o and`)
 - OR (`||`, `|`, `o or`)
 - XOR (`xor`)
- $a \text{ OP } b$ per la notazione simbolica
- `OP(a,b)` per la notazione testuale
- Operatori unari
 - NOT (`~`)
- $\text{OP } a$
- Valori numerici di a, b vengono interpretati come logici
 - 0 come falso
 - Tutti i numeri diversi da 0 come vero

&& vs & e || vs |

- $\&\&$ ($||$) funziona con gli scalari
 - Valuta prima l'operando di sinistra
 - Se è sufficiente a decidere il valore dell'espressione si ferma

$a/b > 10$

- Non voglio eseguire la divisione se b è 0
- $b \neq 0 \&\& (a/b > 10)$
- $\&$ ($|$) funziona con scalari e vettori
- Valuta sempre tutti gli operandi

Esempi

- Hai tra 25 e 30 anni?

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

- Con i vettori

```
>> Voto = [12, 15, 8, 29, 23, 24, 27];
```

```
>> C = (Voto > 22) & (Voto < 25)
```

```
C = [0 0 0 0 1 1 0]
```

```
>> D = (mod(Voto, 2)==0 | (Voto > 18))
```

```
D = [1 0 1 1 1 1 1]
```

- Contare quanti elementi soddisfano la condizione

```
>> nVoti = sum( (Voto > 22) & (Voto < 25) )
```

Costrutto if

- Permette di eseguire istruzioni a seconda del valore di un'espressione booleana
- Utilizza `if`, `else`, `elseif`, `end`

```
if expression1
    statement1
elseif expression2
    statement2
else
    statement3
end
```

- I rami `elseif` e `else` non sono obbligatori!

Ciclo while

while expr

 istruzioni da ripetere finchè expr è vera

end

- Il valore di expr deve cambiare nelle ripetizioni per garantire la terminazione

Ciclo while

- Calcoliamo gli interessi composti fino al raddoppio del capitale, con un interesse annuo del 8%

```
value = 1000;  
year = 0;  
while value < 2000  
    value = value * 1.08  
    year = year +1  
end
```

Ciclo for

```
for variabile = array  
    istruzioni  
end
```

- Tipicamente array è un vettore, quindi variabile assume valori scalari
 - Alla prima iterazione variabile=array (1)
 - Alla seconda iterazione variabile=array (2)
 - ...

Ciclo for

```
vector = [1,2,3]
```

- Per sommare tutti gli elementi del vettore (funzione sum)

```
somma=0;  
for n = vector  
    somma = somma + n;  
end
```

Funzioni

- Come in C è possibile definire funzioni

```
function f = fattoriale (n)
```

```
    f = 1
```

```
    for ii= 1:n
```

```
        f = f * ii
```

```
    end
```

- La testata (*header*) inizia con la parola chiave function e definisce
 - Nome della funzione
 - Argomenti (input)
 - Valore restituito (output)
- Il corpo (*body*) include le istruzioni da eseguire quando la funzione viene chiamata
 - Utilizza gli argomenti e assegna il valore restituito

Funzioni

- Una funzione può essere invocata in un programma attraverso il suo nome, seguito dagli argomenti fra parentesi tonde
 - può avere più argomenti separati da virgola

`f (x, y)`

- Quando è necessario restituire più valori, possiamo usare la notazione degli array

`function [v1, v2, v3] = f (x, y)`

- Esempio

`function [s, p] = sumProd(a, b)`

`s = a + b;`

`p = a * b;`

File Funzione

- Le funzioni sono scritte in file di testo sorgenti
 - Devono avere estensione .m
 - Devono avere lo stesso nome della funzione
 - La prima riga del file deve contenere l'header della funzione

The diagram shows a MATLAB function definition within a yellow box. A red bracket labeled "testata" spans from the start of the function keyword to the end of the parentheses. Another red bracket labeled "corpo" spans from the assignment of f=1 to the end of the loop body.

```
function f=fattoriale(n)
    f=1
    for i=1:n
        f = f*i
    end
```