MATLAB

- sta per MATrix LABoratory
- strumento specializzato per l'esecuzione di calcoli ingegneristici e scientifici
- obiettivo → fornire un ambiente di programmazione di facile utilizzo per l'esecuzione di calcoli complessi focalizzati sull'elaborazione di matrici di dati
- è in grado di eseguire programmi scritti nel linguaggio di programmazione ed è corredato da un ampio numero di librerie di funzioni predefinite che semplificano il lavoro di programmazione e migliorano l'efficienza del risultato

- permette di creare, trasformare, visualizzare e immagazzinare dati complessi
- offre la possibilità di creare dati strutturati, costrutti per modificare il flusso di controllo di un programma (selezione e cicli) e meccanismi di modularizzazione dei programmi (funzioni)

- un programma in MATLAB non deve essere trasformato in codice eseguibile dal calcolatore. Esso invece viene interpretato direttamente dall'ambiente MATHLAB
- in un programma MATLAB non occorre dichiarare le variabili → esse sono definite nel punto in cui vengono utilizzate per la prima volta. Il loro tipo è dinamico nel senso che esso può cambiare durante l'esecuzione del programma per effetto di assegnamenti di valori che appartengono a tipi diversi
- MATLAB è focalizzato sulla trattazione di problemi di natura numerica e offre meccanismi molto semplici da usare e sofisticati ed efficienti per il trattamento di array di dati con varie dimensioni

LINGUAGGI INTERPRETATI e COMPILATI

- linguaggi di programmazione non sono immediatamente comprensibili dal calcolatore quindi non è in grado di eseguirne direttamente le istruzioni. Ci sono due possibilità:

① trasformare i programmi in codice comprensibile dal calcolatore prima di tentare l'esecuzione → questo processo si chiama **COMPILAZIONE** e viene effettuato con l'ausilio di un programma che si chiama COMPILATORE

VANTAGGI:
- consente di ottenere programmi che vengono eseguiti più velocemente rispetto all'interpretazione perché non richiedono l'intermediazione dell'interprete
- l'approccio basato sulla compilazione consente di ottenere un programma eseguibile che può essere installato ed eseguito su un calcolatore indipendentemente dalla presenza dell'interprete purché tale programma eseguibile sia compatibile con il linguaggio macchina del calcolatore

② installare sul calcolatore un programma, chiamato **INTERPRETE**, che è in grado di eseguire direttamente le istruzioni dei programmi scritti nel linguaggio di programmazione di alto livello attuando le azioni appropriate

VANTAGGI:
- maggiore semplicità e velocità nella scrittura dei programmi
- possibilità di provare incrementalmente il programma durante la scrittura per verificare istruzioni

LINGUAGGI CON TIPI DINAMICI

- un linguaggio di programmazione supporta un sistema di tipi dinamico quando la maggior parte dei controlli di coerenza e coesistenza sui tipi dei valori coinvolti nelle espressione vengono effettuati durante l'esecuzione dei programmi
- il tipo non viene associato alle variabili ma ai valori che queste possono assumere → una variabile può cambiare tipo nel tempo, al variabile del suo valore

INTERFACCIA – MATLAB fornisce una serie di finestre e menù accessori che consentono di conoscere lo stato di ciascuna variabile, il contenuto della directory di lavoro, la storia dei comandi eseguiti fino al momento corrente

- in MATLAB l'unità fondamentale di dato è un ARRAY → sono collezioni ordinate di valori che si distinguono in VETTORI e MATRICI
- VETTORI – sono monodimensionali → una sola riga o una sola colonna
  - gli SCALARI sono un particolare tipo di vettori costituiti da una sola riga e una sola colonna
- MATRICI – possono avere due, tre o più dimensioni

- in MATLAB ogni valore numerico viene considerato come un numero reale
- risultato della divisione viene automaticamente assegnato alla variabile predefinita ans

Comando	Risultato	Commento
`1234/6`	`ans = 205.67`	calcolo di un valore scalare
`a=1234/6`	`a = 205.67`	assegnamento alla variabile a del risultato dell'espressione 1234/6
`5/0`	`ans = Inf`	divisione per zero
`5^2`	`ans = 25`	elevamento a potenza
`eps`	`eps = 2.2204e-16`	variabile predefinita: la più piccola differenza rappresentabile tra due numeri
`real(4+5j)`	`ans = 4`	real è una funzione predefinita che restituisce la parte reale di un numero complesso.
`1+1==2` `1+1~=2`	`ans = 1` `ans = 0`	1 = vero, 0 = falso, "==" uguale, "~=" diverso

Input	Output	Commento

`a=[1 2; 3 4]`	`a =`
	`1 2`
	`3 4`
		a è una matrice 2x2, ";" separa le righe

`x=[sqrt(3) (1+2)/5]`	`x =` `1.73205 0.60000`	gli elementi di un array possono essere calcolati a partire da espressioni; sqrt calcola la radice quadrata
`x(3)=abs(x(1))`	`x =` `-1.30000 1.73205 1.30000`	x è il vettore definito alla riga precedente; x(3) permette di accedere al terzo elemento del vettore; a questo si assegna il valore assoluto (abs) del primo elemento dello stesso vettore
`x(5)=4`	`x =` `-1.30000 1.73205 1.30000 0.00000 4.00000`	il vettore x viene esteso per includere nuovi elementi

`b=a'`	`b =`
	`1 3`
	`2 4`
		a è la matrice definita nella prima riga della tabella; a b viene assegnata la matrice trasposta di a (scambiate righe e colonne)

`c=a+b`	`c =`
	`2 5`
	`5 8`
		a e b sono le matrici definite precedentemente nella tabella; a c si assegna la somma di matrici

`x=[-1 0 2]; y=x'`	`y =`
	`-1`
	`0`
	`2`
		il ";" blocca l'output, ma non impedisce l'esecuzione del comando

`x=1:5`	`x =` `1 2 3 4 5`	operatore ":" per produrre vettori di numeri
`y=0:pi/4:pi`	`y =` `0.00000 0.78540 1.57080 2.35619 3.14159`	operatore ":" con passo di incremento a valori non interi (pi corrisponde a π)
`v=10:-4:-3`	`v =` `10 6 2 -2`	valori negativi del passo (-4) e dell'estremo inferiore (-3)
`vuoto=[]`	`vuoto = [](0x0)`	consente di creare un array di tipo double che non contiene elementi

`a = [1 2; 3 4];`
`a^2`
	`ans =`
	`7 10`
	`15 22`
		elevamento a potenza tra matrici; corrisponde a scrivere a * a, dove '*' indica il prodotto righe per colonne tra matrici

`a = [1 2; 3 4];`
`a.^2`
	`ans =`
	`1 4`
	`9 16`
		elevamento a potenza dei singoli valori della matrice

→ vettori e matrici vengono delimitati da una coppia di parentesi quadre

→ elementi di vettori e matrici vengono separati da spazi o da virgole

→ le righe della matrici vengono separate da ;

→ possibilità di modificare dinamicamente la dimensione di matrici e vettori

→ oltre a due valori estremi è possibile specificare un passo che può essere un numero positivo o negativo

→ se il passo non è specificato, l'interprete lo considera uguale a 1

- un diagramma è un insieme di punti congiunti da una linea che serve per dare continuità al grafico → ciascun punto è caratterizzato da una coppia di valori che rappresentano le proiezioni del punto sui due assi
  ↳ funzione predefinita **plot** disegna il diagramma corrispondente
- per visualizzare sullo stesso grafico più di una curva si utilizzano i comandi **hold on** e **hold off**
  - hold on permette di bloccare la visualizzazione dei dati correnti mentre si eseguono altri comandi sul grafico
  - hold off sblocca la visualizzazione

- COMMENTI → non vengono interpretati
  → sono a esclusivo beneficio dei programmatori
  → inizia con il simbolo % e termina alla fine della riga corrente → nel caso in cui il commento occupi più righe, ciascuna riga deve iniziare con il simbolo %

CONTINUAZIONE DI LINEA
→ ogni istruzione deve essere scritta sulla stessa riga
→ utilizzando l'operatore di continuazione di linea si può scrivere un'istruzione su più righe
→ inserendo tre punti consecutivi alla fine di una riga per poter continuare l'istruzione nella riga successiva
→ operatore di continuazione di linea non può essere usato per spezzare su due righe il nome di un comando di MATLAB oppure una stringa

COMANDI UTILI → **help** richiama la guida in linea → può essere utilizzato per avere informazioni su funzioni e script
→ **who, whos, workspace** mostrano l'elenco delle variabili definite nello spazio di lavoro o workspace
→ **clear** ripulisce lo spazio di lavoro eliminando tutte le variabili che sono contenute in esso

SCRIPT
- semplice file di testo puro, senza alcuna forma di formattazione
- esso deve essere • testo senza formattazione → lo script può essere letto e interpretato correttamente da MATLAB
  • file con estensione ".m" → file viene automaticamente riconosciuto da MATLAB
- possibile invocare l'edit utilizzando il comando edit → edit 'nomeFile' (senza specificare estensione .m)
- per poter essere eseguito dall'interprete lo script dovrà necessariamente trovarsi • nella directory
  • in una delle directory in cui l'interprete cerca gli script che vengono invocati

- uno script ben strutturato dovrà avere la seguente struttura:

1. commento di intestazione
2. sezione di input
3. sezione di calcolo
4. sezione degli output

- Le operazioni di input sono le operazioni che consentono di acquisire i dati da processare all'interno del programma
- Le operazioni di output sono le operazioni che consentono di visualizzare o memorizzare i risultati del programma

- istruzione **input** visualizza la stringa `messaggio` e la memorizza nella variabile → dato = input('messaggio')
  - ↳ permette di inserire un valore scalare, stringa (tra apici), un vettore (tra quadre)

- funzione **disp** permette di stampare dei messaggi → disp('messaggio')
  - ↳ accetta solo argomenti di tipo stringa → attraverso funzione num2str si possono stampare anche numeri

```
risultato = 5
msg = ['il risultato è num2str(risultato)']
disp(msg)
```

- la funzione **fprintf** permette di stampare allo stesso tempo testo e dati → fprintf('formato', var1, var2, ...)
  dove 'formato' è una stringa che specifica come i dati devono essere visualizzati e var1, var2 sono le variabili che contengono i dati
  - ↳ stringa 'formato' può contenere:
    - CARATTERI NORMALI
    - CARATTERI SPECIALI → iniziamo con '\'
    - CARATTERI di CONVERSIONE o FORMATO → iniziamo con '%'

Carattere	Output
`\n`	Salta alla riga seguente
`\t`	Inserisce una tabulazione
`%%`	Stampa il carattere %
`\\`	Stampa il carattere \
`%d`	Stampa il valore come un numero intero
`%f`	Stampa il valore come un numero con la virgola
`%e`	Stampa il valore con notazione esponenziale
`%g`	Stampa il valore usando la notazione più compatta fra quella esponenziale e quella in virgola mobile.
`%c`	Stampa il valore come carattere

- MATLAB essendo un linguaggio interpretato, esso ci stimola all'utilizzo di un approccio di programmazione che viene chiamato 'trial and error' → si procede per tentativi sperimentando l'uso dei vari comandi man mano che si rendono necessari
- È necessario una volta compreso come utilizzare le istruzioni che sono necessarie, consolidare il programma nella sua struttura e dare una forma definitiva alle sue istruzioni scrivendole in uno script che potrà essere usato più volte
- Si cerca di non riutilizzare la stessa variabile per scopi diversi o, se necessario, ci si deve assicurare di inserire commenti chiari e inequivocabili che permettano di ricordare il motivo di questa scelta anche nel futuro

VARIABILI

- rappresentano i dati su cui lavora un programma
- esse sono denotate mediante un nome, chiamato IDENTIFICATORE
- corrispondono a locazioni di memoria e ciascuna, oltre un identificatore, ha un tipo (insieme dei valori e operazioni ammissibili), una dimensione, un indirizzo e un valore
- è un elemento di memoria che contiene valori che possono cambiare durante l'esecuzione del programma
- in MATLAB una variabile può contenere dati di tipo diverso → il suo tipo può variare dinamicamente
- esistenza della variabile viene riconosciuta dall'interprete nel momento del suo primo utilizzo → non distingue tra la fase di dichiarazione e di utilizzo della variabile (variabili create al momento dell'inizializzazione)
- il comando whos permette di conoscere le informazioni principali su una variabile
- ogni variabile ha un NOME
  - deve iniziare con una lettera seguita da qualsiasi sequenza di lettere, numeri, caratteri underscore
  - si possono utilizzare fino a 63 caratteri → se i nomi di due variabili che intendiamo essere distinte hanno i primi 63 caratteri uguali, essi saranno corrispondenti alla stessa variabile
    - nomi troppo lunghi possono provocare problemi
  - è opportuno usare nomi di variabili che siano significativi → questo consente di capire il ruolo di una certa variabile nel programma → migliora leggibilità e modificabilità del programma
  - il fatto che non è necessario dichiarare le variabili rende il linguaggio molto flessibile MA rende i programmi di più difficile interpretazione → è quindi opportuno DEFINIRE per le variabili principali un DIZIONARIO da inserire all'inizio del programma

- i tipi più comuni per una variabile sono DOUBLE o CHAR
  - una variabile DOUBLE contiene uno scalare o un array di numeri espressi in 64 bit (8 byte) con doppia precisione
  - le parti reali e immaginarie possono essere positive o negative, con valore assoluto nell'intervallo $[10^{-308}; 10^{308}]$

- una variabile CHAR contiene uno scalare o un array di valori a 16 bit, ciascuno dei quali rappresenta un carattere
- un tale array viene chiamato STRINGA

- ASSEGNAMENTO → variabile = espressione → espressione può coinvolgere scalari, array, altre variabili, costanti, simboli matematici, funzioni

→ MATLAB offre una serie di funzioni predefinite che consentono di creare array speciali

Esempio di assegnamento	Risultato	Note
`a = [0 7+1];`	`[0 8]`	Gli operatori matematici possono essere utilizzati negli assegnamenti
`b = [a(2) 5 a];`	`[8 5 0 8]`	L'utilizzo dell'array `a` (definito nella riga 1) nell'assegnamento di `b` ha un impatto sulla dimensione di `b` oltre che sui suoi valori

`g = b';`	`[8`
	`5`
	`0`
	`8]`
		L'operatore ' consente di ottenere la trasposta di un array. In questo esempio `b` è il vettore definito alla riga 2

`x = 1:2:10;`	`[1 3 5 7 9]`	L'operatore : consente di costruire in modo sintetico un array di valori crescenti o decrescenti

`m = [x' x'];`	`1 1`
	`3 3`
	`5 5`
	`7 7`
	`9 9`
		Matrice 5 x 2 ottenuta da due vettori 5 x 1. In questo esempio `x` è il vettore definito alla riga 4

`n(1:4, 1:3) = 3;`	`3 3 3` `3 3 3` `3 3 3` `3 3 3`	A sinistra dell'assegnamento viene specificata la forma dell'array, in questo caso una matrice con 4 righe e 3 colonne. Il valore a destra è assegnato a tutti gli elementi dell'array.

`n(1:2, 2:3) = 4;`	`3 4 4`
	`3 4 4`
	`3 3 3`
	`3 3 3`
		L'assegnamento si applica anche a porzioni di array (sottoarray). `n` è l'array definito alla riga precedente.

		Si assegna il valore 4 al sottoarray che include le prime 2 righe e le colonne 2 e 3.
`c(2, 3) = 5`	`0 0 0` `0 0 5`	Gli elementi non specificati alla creazione dell'array assumono il valore 0.
`a = []`	`a = []`	Crea un array vuoto.

Funzione	Risultato	Commento
`a = zeros(2);`	`0 0` `0 0`	`zeros` con un solo parametro crea una matrice quadrata della dimensione specificata
`b = zeros(2,3);`	`0 0 0` `0 0 0`	Il primo parametro è il numero delle righe, il secondo il numero delle colonne

`c = [1 2; 3 4; 5 6];`	`1 2`
	`3 4`
	`5 6`
		`size` restituisce la dimensione di un array. Tale funzione può essere usata per creare un array di 0 della stessa dimensione dell'array a cui si applica `size`. `size(c)` consente di ottenere il numero di righe e colonne di `c`. Tali valori vengono assegnati a `zeros` che genera la matrice 3x2 assegnata a `d`

`d = zeros(size(c));`	`0 0` `0 0` `0 0`	

Funzione	Significato
`zeros (n)`	Genera una matrice nxn di zeri
`zeros (m,n)`	Genera una matrice mxn di zeri
`zeros (size(arr))`	Genera una matrice di zeri della stessa dimensione di `arr`
`ones(n)`	Genera una matrice nxn di uno
`ones(m,n)`	Genera una matrice mxn di uno
`ones(size(arr))`	Genera una matrice di uno della stessa dimensione di `arr`
`eye(n)`	Genera la matrice identità nxn
`eye(m,n)`	Genera la matrice identità mxn
`length(arr)`	Restituisce la dimensione più lunga.
`size(arr)`	Restituisce due valori: il numero di righe e colonne dell'array
`v(end)` `m(end, end)`	Ultimo elemento del vettore v e ultimo elemento dell'ultima riga della matrice m; la costante end, come indice in un array, denota il più alto valore possibile dell'indice

- la lettura di dati da tastiera può avvenire mediante la funzione di INPUT → variabile = input ('inserisci');
  - quello che può essere inserito può essere uno scalare, un array racchiuso tra [], una stringa racchiusa tra apici semplici ''
  - dato inserito da tastiera viene memorizzata in variabile
- l'espressione valore2 = input ('inserisci', 's') interpreta come stringa qualsiasi sequenza di caratteri alfanumerici inseriti dall'utente

- MATLAB distingue tra file scritti nel suo formato proprietario (estensione .mat), file di testo e file in altri formati
  - il comando per caricare dati da file si chiama load
    - load 'nomefile' carica nello spazio di lavoro tutte le variabili e cui valori si trovano nel file nomefile.mat
    - load 'nomefile' < lista nomi variabili > carica nello spazio di lavoro solo le variabili il cui nome appare nella lista, se queste si trovano nel file chiamato nomefile.mat

- MATLAB definisce un insieme di variabili il cui valore è predefinito
  - è fortemente sconsigliato modificare il valore di una variabile predefinita

Variabile	Scopo
`pi`	contiene 15 cifre significative di π
`i, j`	contiene il valore ($\sqrt{-1}$)
`inf (o Inf)`	rappresenta l'infinito (ottenuto di solito come risultato di una divisione per 0)
`nan`	not-a-number è il risultato di una operazione matematica non definita, es 0/0
`clock`	contiene la data e l'orario corrente. E` un vettore di sei elementi (anno, mese, giorno, ora, minuti, secondi)
`date`	contiene la data corrente sotto forma di stringa
`eps`	la più piccola differenza rappresentabile tra due numeri (epsilon)
`ans`	contiene il risultato della valutazione di un'espressione dalla linea di comando quando questo non viene assegnato ad alcuna variabile, e.g.: `>>3*2` fa sì che `ans=6`

SCALARI → è un tipo particolare di array costituito da un unico elemento

→ operazioni definite per gli scalari sono quelle aritmetiche: somma, sottrazione, moltiplicazione, divisione, potenza

ARRAY

- possono avere una o più dimensioni
- accesso ai singoli elementi di un array X avviene indicando tra parentesi tonde il numero o i numeri che corrispondono alla posizione desiderata dell'array o in forma linearizzata

Operazione	Risultato	Commento
`a1(1) = a1(3);`	`a1 = [3 2 3 4]`	il valore dell'elemento in terza posizione in a1 viene assegnato alla posizione 1 dello stesso array
`a1(2) = a2(2,3);`	`a1 = [3 5 3 4]`	il valore dell'elemento di riga 2 e colonna 3 in a2 viene assegnato alla posizione 2 di a1
`a1(3) = a2(6);`	`a1 = [3 5 5 4]`	uso della *forma linearizzata* di a2 (vedi la spiegazione nel testo)

→ il sesto elemento della forma linearizzata di a2 corrisponde all'elemento di riga 2 e colonna 3

→ possibile usare forma classica o linearizzata

- in MATLAB la dimensione di un array può cambiare
- gli elementi per i quali non è stato definito esplicitamente un valore assumo il valore 0
- meccanismo per accedere all'ultimo elemento è → `nomearray(end)`

- le operazioni tra array si dividono in due categorie: **ARRAY OPERATIONS** e **MATRIX OPERATIONS**
- **ARRAY OPERATIONS** → vengono eseguite sugli elementi degli array coinvolti che si trovano nelle stesse posizioni
  → applicabili solo ad array con lo stesso numero di righe e di colonne
- **MATRIX OPERATION** → seguono le regole dell'algebra lineare

Operazione	Sintassi MATLAB	Commenti
Array addition	`a + b`	Array e matrix addition sono identiche
Array subtraction	`a - b`	Array e matrix subtraction sono identiche
Array multiplication	`a .* b`	Ciascun elemento del risultato è pari al prodotto degli elementi corrispondenti nei due operandi
Matrix multiplication	`a * b`	Prodotto di matrici
Array right division	`a ./ b`	risultato(i,j)=a(i,j)/b(i,j)
Array left division	`a .\ b`	risultato(i,j)=b(i,j)/a(i,j)
Matrix right division	`a / b`	a*inversa(b)
Matrix left division	`a \ b`	inversa(a)*b
Array exponentiation	`a .^ b`	risultato(i,j)=a(i,j)^b(i,j)

→ questa operazione potrebbe risolvere $Ax=B$

infatti $x = A^{-1}B = inversa(A)*B = A\backslash B$

- MATLAB offre molte funzioni aggiuntive che consentono di operare sugli scalari e sugli array

Funzione	Scopo
`ceil(x)`	Ceiling = soffitto. Approssima `x` all'intero immediatamente maggiore cioè al più piccolo intero $\geq x$. Se `x` è un array, tale approssimazione viene effettuata per ciascuno degli elementi di `x`.
`floor(x)`	Floor = pavimento. Approssima `x` all'intero immediatamente minore cioè al più grande intero $\leq x$. Se `x` è un array, tale approssimazione viene effettuata per ciascuno degli elementi di `x`.
`fix(x)`	Approssima `x` all'intero più vicino verso lo zero. Se `x` è un array, tale approssimazione viene effettuata per ciascuno degli elementi di `x`.
`round(x)`	Approssima `x` all'intero più vicino. Se `x` è un array, tale approssimazione viene effettuata per ciascuno degli elementi di `x`.
`max(x)`	Restituisce il valore massimo nel vettore `x` e, opzionalmente, la collocazione di questo valore in `x`.
`min(x)`	Restituisce il valore minimo nel vettore `x` e, opzionalmente, la collocazione di questo valore nel vettore .
`mod(x,y)`	Restituisce il modulo di `x` e `y` che viene definito come `x - y .* floor (x ./ y)` per `y ~= 0` .
`rand(N)`	Genera una matrice di NxN numeri casuali compresi tra 0 e 1.
`sum(x)`	Se x è un vettore, restituisce la somma di tutti i suoi elementi. Se x è una matrice, restituisce un vettore riga contenente la somma degli elementi appartenenti a ciascuna colonna della matrice. Oltre al comando `sum(x)`, è possibile usare anche il comando `sum(x, DIM)`, dove DIM rappresenta la dimensione lungo cui eseguire la somma. Se si vogliono sommare gli elementi di cuascuna riga si scrive `sum(x, 2)`. Il risultato viene immagazzinato in un vettore colonna.

SUBARRAY

- meccanismo dei subarray consente di utilizzare porzioni di array in modo indipendente dall'array di cui fanno parte
- viene identificato inserendo a fianco del nome dell'array, tra parentesi tonde, l'indicazione degli indici dei valori nell'array di partenza che fanno parte del sottoarray
  - ↳ vettore (2) → sottoarray costituito dal secondo elemento
  - vettore [1 4] → sottoarray costituito dai valori in posizione 1 e 4
  - vettore [2:4] → sottoarray contenente i valori di tutte le posizioni comprese tra 2 e 4
  - matrice ([1 4], [2 3]) → matrice costituita da quatto elementi: matrice (1,2), matrice (1,3) matrice (4,2) e matrice (4,3)
- in una matrice a due dimensioni il simbolo : può essere usato al posto dell'indice di riga o di colonna per indicare che quell'indice varia da 1 e la sua dimensione massima
  - ↳ matrice (1:2:5, :) → sottoarray contenente le righe di indice tra 1 e 5 con passo di incremento 2 e le colonne di indice tra 1 e il numero di colonne della matrice
- è possibile attribuire nuovi valori agli elementi dell'array che fanno parte del sottoarray
  - ↳ matrice (2:2:4, :) = [-1 -2 3; -4 -5 -6]; → gli elementi del sottoarray di matrice che include righe 2 e 4 diventano pari ai valori specificati a destra dell'assegnamento

CANCELLAZIONE

- assegnamento della costante [] può essere utilizzato per cancellare elementi di un array → se si assegna [] a un elemento di un vettore non si crea un buco ma il numero degli elementi del vettore diminuisce di uno e il vettore viene ricompattato lasciando gli elementi originali nello stesso ordine
- [] non è assegnabile ai singoli elementi di matrice → esso è assegnabile a intere righe o colonne

TIPI DI DATO LOGICO

- può assumere solo due valori: true o false che corrispondono ai valori 1 e 0
- per creare una variabile di tipo logico basta assegnare a una variabile un valore logico
- le variabili di tipo logico occupano solo un byte di memoria
- si possono creare array i cui elementi sono di tipo logico
- MATLAB permette di usare le variabili logiche insieme a quelle numeriche
  - se una variabile logica è usata al posto di una variabile numerica, il valore logico 1 (true) è convertito nel valore numerico 1, mentre il valore logico 0 (false) nel valore numerico 0
  - se una variabile numerica è usata al posto di una variabile logica, il valore numerico 0 è convertito nel valore logico 0 (false), mentre tutti i valori diversi da 0 vengono convertiti nel valore logico 1 (true)

- ESPRESSIONI LOGICHE possono essere basate su
  - operatori relazionali
  - operatori logici

  entrambe offrono come risultato un valore logico ma le prime utilizzano operatori relazionali per confrontare due operandi (possono essere numerici, caratteri, stringhe) mentre le seconde utilizzano operatori logici applicati a operandi con valore logico

- OPERATORI RELAZIONALI

`==`	uguale
`~=`	diverso
`>`	maggiore
`>=`	maggiore o uguale
`<`	minore
`<=`	minore o uguale

- a seconda che la condizione espressa sia o non sia rispettata dagli operandi, l'espressione assume il valore 1 o 0 rispettivamente
- quando si confrontano due array essi devono avere le stesse dimensioni

- Esistono in MATLAB funzioni che permettono di confrontare di lunghezza diversa
  - strcmp → applicata a due stringhe restituisce 1 se le due stringhe sono identiche, 0 altrimenti
  - strcmpi → applicata a due stringhe restituisce 1 se le due stringhe contengono gli stessi caratteri in posizioni corrispondenti, indipendentemente dal fatto che siano minuscoli o maiuscoli, 0 altrimenti
  - strncmp → presi come parametri due stringhe e un intero n, restituisce 1 se le due stringhe hanno i primi n caratteri uguali, 0 altrimenti

- OPERATORI LOGICI

  && AND → vero quando entrambi operandi sono veri

  || OR → falso quando entrambi operandi sono falsi

  Xor XOR → vero quando solo uno dei due operandi è vero

  ~ NOT → opposto del valore dell'operando

- FUNZIONI LOGICHE

`all` applicata a un array `x`, restituisce un vettore riga con lo stesso numero di colonne di `x`, che contiene `1 (true)`, se la corrispondente colonna di `x` contiene tutti elementi pari a `1 (true)`, `0 (false)` altrimenti;

`any` applicata a un array `x`, restituisce un vettore riga con lo stesso numero di colonne di `x`, che contiene `1 (true)` , se la corrispondente colonna di `x` contiene almeno un elemento pari a `1 (true)`, `0 (false)` altrimenti;

`isinf` applicata a un array `x`, restituisce un array delle stesse dimensioni di `x` con `1 (true)` in corrispondenza degli elementi di `x` che sono `inf`, `0 (false)` altrove.

`isempty` applicata a una variabile x, restituisce `1 (true)` se `x` è vuota (cioè uguale a []), `0 (false)` altrimenti;

`isnan` applicata a un array `x`, restituisce un array delle stesse dimensioni di `x` con `1 (true)` in corrispondenza degli elementi di `x` che sono `NaN`, `0 (false)` negli altri posti;

`finite` applicata a un array `x`, restituisce un array delle stesse dimensioni di `x` con `1 (true)` in corrispondenza degli elementi di `x` finiti, `0 (false)` negli altri posti;

`ischar` applicata a una variabile `x`, restituisce `1 (true)` se `x` è di tipo `char`, `0 (false)` altrimenti;

`isnumeric` applicata a una variabile `x`, restituisce `1 (true)` se `x` è di tipo `double`, `0 (false)` altrimenti;

`isreal` applicata a un array `x`, restituisce `0 (false)` se almeno una delle componenti di `x` è un numero con parte immaginaria non nulla, `1 (true)` altrimenti.

STRUTTURE

- consentono di associare più elementi a una singola variabile
- gli elementi di una struttura sono identificati da un nome
- gli elementi di una struttura possono appartenere a diversi tipi di dato
- una struttura è un tipo di dato formato da più elementi, chiamati CAMPI e ciascuno è identificato da un nome
  - per accedere ai campi di una struttura si utilizza la NOTAZIONE PUNTATA → si specifica il nome della variabile struttura seguito da un punto e dal nome del campo a cui si vuole accedere → nome_struttura.nome_campo
- il nome della variabile struttura può essere utilizzato senza specificare il campo per indicare intera struttura

- per creare una struttura ci sono due approcci:
  - creare un campo della struttura alla volta, mediante una serie di assegnamenti
  - creare e inizializzare tutti i campi della struttura con una sola istruzione, mediante la funzione struct

    nome_variabile = struct('campo1', val1, 'campo2', val2, ...)

- si può creare anche un array di strutture → per accedere a un elemento si utilizza la notazione standard nome_array().nome_campo

- l'aggiunta di un campo in un array di strutture avviene mediante assegnamento
  - l'aggiunta di un campo in un singolo elemento di un array di strutture ha come risultato l'aggiunta di quel campo anche in tutti gli altri elementi dell'array di strutture

- per rimuovere un campo da un array di strutture è necessario utilizzare la funzione rmfield

  ris = rmfield(nome_array, 'nome_campo') → dove nome_array è l'array di strutture da cui si desidera rimuovere un campo, 'nome_campo' è il nome del campo che si desidera rimuovere e ris è il nome della variabile in cui viene memorizzato il risultato, ovvero l'array di strutture da cui è stato rimosso il campo 'nome.campo'

- In molti casi è utile accedere al contenuto di uno specifico campo di tutti gli elementi di un array di strutture utilizzando la notazione [nome_array.nome_campo]
  - può essere utilizzata solo per accedere ai dati in lettura e quindi non può essere utilizzata per modificare i valori dei campi

- il campo di una struttura può essere a sua volta un array di strutture

STRUTTURE DI CONTROLLO

- sono costrutti sintattici che consentono di modificare l'ordine di esecuzione delle istruzioni
- le strutture di controllo sono:
  - **SEQUENZA** → consiste in una lista ordinata di istruzioni
  - **SELEZIONE** → costrutti che permettono, tramite il valore di una condizione, di scegliere quali istruzioni eseguire
  - **CICLO** → schema che permette di ripetere più volte una o più istruzioni

SELEZIONE

- costrutti di selezione permettono di scegliere tra una o più sequenze di istruzioni quella da eseguire, a seconda del valore di una o più condizioni rappresentate da espressioni logiche

COSTRUTTO IF

```
if espressione 1
   istruzione 1
   istruzione 2 ...        ] blocco 1

elseif espressione 2
   istruzione 1
   istruzione 2            ] blocco 2

else
   istruzione 1
   istruzione 2 ...        ] blocco k

end
```

espressioni logiche

blocchi di istruzioni possono contenere altri costrutti if

COSTRUTTO SWITCH

- permette di scegliere fra diverse alternative a seconda dei valori di un singolo intero, carattere o espressione logica

```
switch (espressione_switch)

   case caso 1
      istruzione 1
      istruzione 2 ...

   case caso 2
      istruzione 1
      istruzione 2 ...

   otherwise
      istruzione 1
      istruzione 2 ...

   end
```

- può rappresentare sia valori numerici che stringhe
- se espressione_switch non risulta uguale a nessun caso e sia presente il ramo otherwise, allora verrà eseguito il blocco corrispondente a quel ramo
- la parola end termina il costrutto
- se espressione_switch risultasse uguale a più di uno dei casi proposti, verrà eseguito solo il blocco di istruzioni corrispondenti al primo caso trovato sequenzialmente per poi passare direttamente alla prima istruzione dopo la parola end

CICLI

- struttura di controllo che permette di ripetere la stessa sequenza di istruzioni più volte, a seconda della valutazione di una data condizione

- **WHILE**

. permette di eseguire più volte la stessa sequenza di istruzioni

```
while espressione
      istruzione 1
      istruzione 2 ...
end
```

ciclo continua finché espressione non assume il valore 0 (False)

- **CICLO FOR**

```
for indice = espressione
    istruzione 1
    istruzione 2 ...
end
```

espressione è l'espressione di controllo del ciclo

indice è una variabile, chiamata indice del ciclo

indice assume il valore della prima colonna dell'array generato

dopo l'esecuzione, indice assume il valore della colonna successiva dell'array generato e, con quel valore, viene eseguito nuovamente il blocco

- quando si usa il ciclo for bisogna → non modificare l'indice all'interno del ciclo
  → preallocare gli array utilizzati all'interno del ciclo — creare array prima del ciclo

- **BREAK** → interrompe il ciclo e porta alla prima istruzione dopo il ciclo
- **CONTINUE** → porta immediatamente a valutare la condizione del ciclo

VETTORIZZAZIONE

- qualora all'interno del ciclo si eseguano solo operazioni sull'indice, che possono essere sostituite dall'equivalente operazioni sui vettori, il ciclo può essere evitato tramite il processo di vettorizzazione

ARRAY LOGICI

- una proprietà degli array è che possono funzionare da 'maschere' per le operazioni aritmetiche
- una maschera è un array che serve per selezionare elementi in un altro array della stessa dimensione, in modo da applicare un'operazione solo sugli elementi selezionati e non sugli altri
- per rappresentare un costrutto if/else si possono utilizzare gli array logici → un array logico seleziona tutti gli elementi di un altro array per i quali una certa condizione è vera

FIND

- funzione predefinita che permette di effettuare ricerche all'interno di un array → essa fornisce un vettore costituito dagli indici degli elementi non nulli dell'array passatole come parametro
- funzione find può essere anche applicata a matrici, nel qual caso i valori da essa restituiti fanno riferimento alla versione linearizzata della matrice

SOTTOPROGRAMMI

- sono moduli di codice che svolgono operazioni particolarmente significative o di uso frequente
- sono importanti per questi motivi:
  - RIUSABILITÀ → un'operazione viene svolta frequentemente
  - ASTRAZIONE → un sottoprogramma che svolge una ben identificata operazione è più semplice e astratto
  - ESTENDIBILITÀ → sottoprogramma sono un'estensione del linguaggio di programmazione

- sottoprogrammi vengono scritti in appositi file
- un sottoprogramma è asservito a un altro programma, detto il PROGRAMMA PRINCIPALE
- il sottoprogramma lavora solo a seguito di una CHIAMATA o INVOCAZIONE da parte del programma principale

- ogni sottoprogramma è caratterizzato da due elementi principali: l'OPERAZIONE da esso realizzata e il MODO in cui esso va utilizzato
  - ↳ include: - MODO in cui esso viene identificato dal programma chiamante
    - DATI che utilizza come punto di partenza
    - RISULTATI prodotti

- il termine usuale con cui si indicano i sottoprogrammi è quello di FUNZIONE

- la definizione della funzione viene data una sola volta, mentre le chiamate della funzione possono essere effettuate un numero arbitrario
- nella definizione si distinguono:
  - TESTATA - interfaccia della funzione
    - contiene l'informazione necessaria a chi scrive nel programma principale la chiamata della funzione
    - nome dei PARAMETRI FORMALI IN INGRESSO vanno scritti tra le parentesi tonde che seguono il nome della funzione
      - il termine formali si riferisce al fatto che al momento della definizione della funzione non hanno un valore proprio ma assumeranno un valore solo all'atto dell'esecuzione del sottoprogramma
      - il termine in ingresso per l'evidente motivo che il loro valore viene fornito dal programma chiamante e costituisce il punto di partenza
      - parametri formali in ingresso possono essere un numero qualsiasi, maggiore o uguale a zero

- nome dei PARAMETRI FORMALI IN USCITA vanno scritti tra le parentesi quadre, separati da virgole, che precedono il nome della funzione
  - il termine formale si riferisce al fatto che al momento della definizione della funzione non hanno un valore proprio ma assumeranno un valore solo alla fine dell'esecuzione del sottoprogramma
  - il termine in uscita per il motivo che il loro valore verrà trasmesso al programma chiamante al termine dell'esecuzione della funzione
  - parametri formali in uscita possono essere un numero qualsiasi, maggiore o uguale a zero

. CORPO - contiene le istruzioni che permettono di ottenere i risultati partendo dagli argomenti

- i PARAMETRI ATTUALI IN INGRESSO e IN USCITA sono le variabili del programma chiamante
  - il termine attuale si riferisce al fatto che questi parametri sono quelli che effettivamente entrano in gioco, assumendo un preciso valore

- Nella chiamata devono essere presenti tanti parametri attuali quanti sono i parametri formali
  - nella chiamata i parametri attuali sono racchiusi tra una coppia di parentesi, tonde per quelli in ingresso e quadre per quelli in uscita

- i parametri attuali in ingresso devono avere, al momento della chiamata della funzione, un valore coerente con le operazioni che, durante l'esecuzione della funzione, vengono applicate al corrispondente parametro formale

- le funzioni stesse devono essere scritte in appositi file, chiamati FILE DI FUNZIONE
  - . sono chiamati M-file perché il loro nome deve avere estensione ".m"
  - . devono avere lo stesso nome della funzione un ora definita
  - . prima parola deve essere "function"
  - . il file deve trovarsi in una cartella che sta nella variabile PATH

- con il comando HELP si possono ottenere informazioni su qualsiasi funzione definita in un M-file, a patto che in esso siano stati inseriti opportuni commenti seguendo alcune semplici convenzioni

- gli script, invece, non hanno un proprio ambiente di esecuzione → passaggio informazioni avviene per il fatto che le variabili del programma principale possono essere lette dalle istruzioni dello script

- nel definire una funzione avente un array come parametro in ingresso, occorre spesso far riferimento alla dimensione di tale array → funzione predefinita length (nome_array)
- nel definire una funzione avente una matrice come parametro in ingresso, occorre spesso far riferimento alla dimensione di tale matrice → ndims (nome_matrice) restituisce il numero n delle dimensioni dell'array
  → size (nome_matrice) restituisce un array con n valori uguali al numero di elementi per ognuna delle dimensioni della matrice

- l'istruzione **RETURN** permette di terminare l'esecuzione della funzione e di restituire immediatamente il controllo al programma chiamante
  ↳ l'esecuzione delle eventuali istruzioni che la seguono non avviene

- La ricorsione è una tecnica di programmazione che si basa sulla possibilità che una funzione chiami sé stessa, direttamente o indirettamente
  ↳ durante l'esecuzione ogni chiamata ricorsiva determina la sospensione dell'esecuzione della chiamata corrente e l'attivazione di una nuova chiamata, con il parametro attuale n decrementato rispetto alla chiamata precedente

- L'uso della ricorsione comporta il rischio di catene infinite di chiamate → ciò accade se:
  - una funzione ricorsiva contiene una chiamata a sé stessa in cui i parametri attuali sono uguali a quelli formali
  - l'esecuzione della funzione ricorsiva porta, indipendentemente dal valore dei parametri formali, a un'ulteriore chiamata ricorsiva

- Le chiamate ricorsive devono avere **ARGOMENTI RIDOTTI** e devono essere **CONDIZIONATE**
- una funzione ricorsiva può contenere chiamate ricorsive multiple

- un tipo predefinito di dati è il tipo funzione che permette di introdurre nei programmi variabili che rappresentano funzioni e di seguire due principali operazioni:
  - trasmettere tali valori di tipo funzione da un programma a un sottoprogramma
  - applicare tale valore a opportuni argomenti

- i valori di tipo funzione vengono memorizzati in variabili dette **HANDLE**

- a una variabile handle possono essere assegnati valori di tipo funzione in due modi:
  - indicando l'identificatore di una funzione già esistente → nome_variabile = @ nome_funzione
  - definendo ex novo una funzione distinta da tutte le altre → nome_variabile = simbolo @ + parametri in ingresso tra parentesi tonde + espressione che fornisce il risultato come funzione dei parametri in ingresso

INPUT e OUTPUT

- con Input/Output (I/O) si intendono tutte quelle operazioni che consentono a un programma di scambiare informazioni con l'esterno

operazioni di INPUT - programma riceve in ingresso le richieste dell'utente e tutti i dati su cui svolgere le elaborazioni

operazioni di OUTPUT - programma comunica il risultato delle proprie elaborazioni all'utente

DIAGRAMMI

- è possibile associare a ciascuna curva un colore, un tratto o un marcatore diverso
- è opportuno inserire una leggenda che chiarisca la corrispondenza tra ciascuna curva e la funzione rappresentata
  - legend ('nome_funzione 1', ..., 'nome_funzione N')

Colore	Stile del marcatore	Stile della linea
r (rosso)	O	- (linea solida)
b (blu)	X	: (linea punteggiata)
g (verde)	*	-- (linea tratteggiata)
k (nero)	+	<niente> (nessuna linea)