

**Trasmissione del calore con applicazioni
numeriche: informatica applicata
a.a. 17/18**



Parte I: Panoramica generale sul MATLAB

Prof. Nicola Forgone

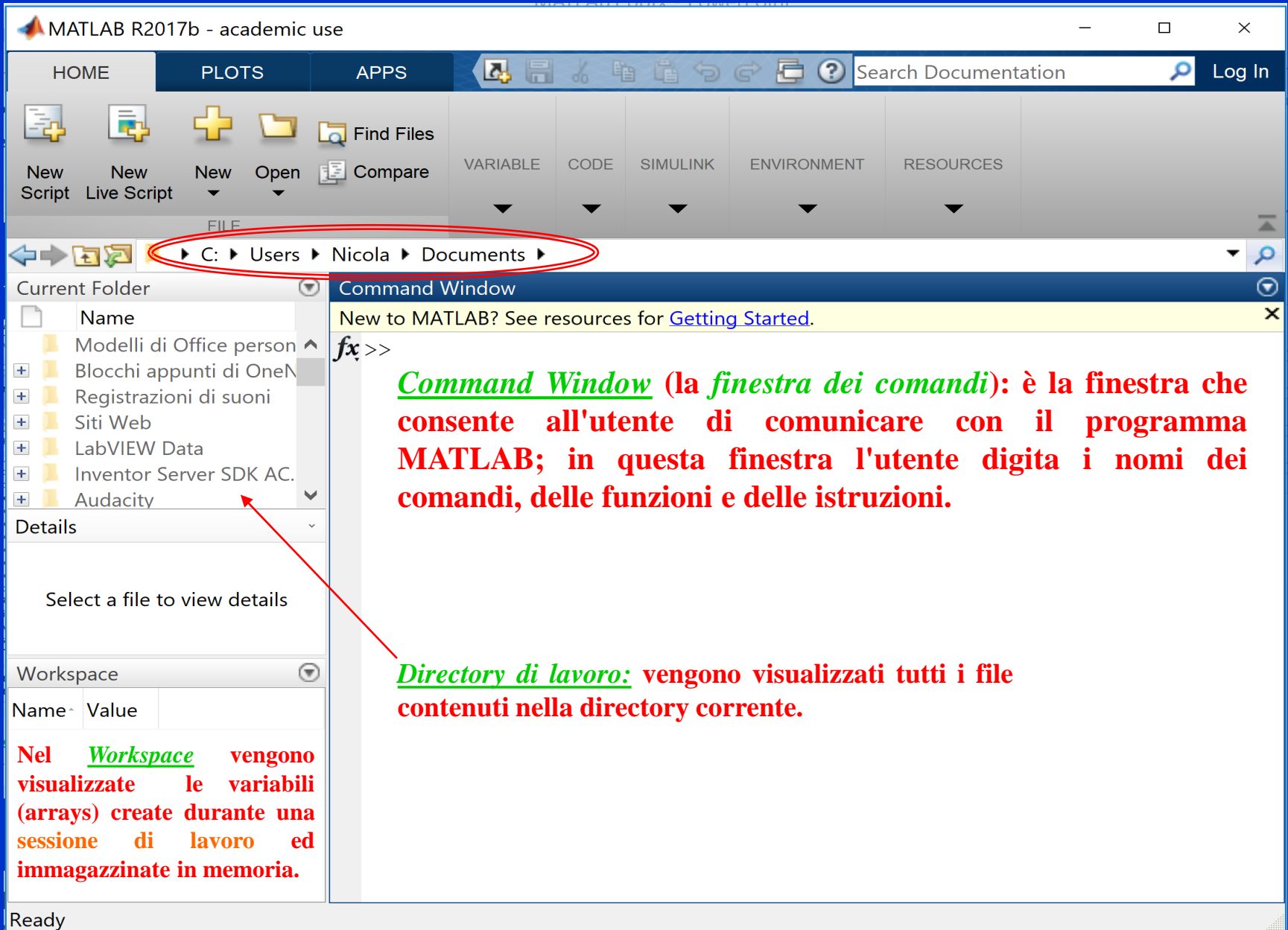
Dipartimento di Ingegneria Civile e Industriale

E-mail: nicola.forgione@unipi.it; tel. 0502218057

Introduzione

- MATLAB (MATrix LABoratory) è un *ambiente di programmazione* di alte prestazioni *per il calcolo tecnico e scientifico*.
- E' flessibile, facile da apprendere, facile da usare e veloce nei calcoli. Esso opera con le più perfezionate librerie esistenti (LINPAK ed EISPACK).
- Esso è anche un *linguaggio di programmazione*, molto simile al linguaggio C, basato su espressioni che rendono facile la programmazione stessa.
- *Effettua tutte le operazioni in doppia precisione* (tutte le variabili reali sono di tipo “double” per default, ovvero sono rappresentate internamente con 64 bit, cui corrispondono circa 16 cifre significative, $\pi=3.141592653589793$). L'uso di un numero limitato di cifre significative per la rappresentazione dei numeri porta a commettere il cosiddetto errore di arrotondamento (round-off error).
- La principale caratteristica è che *esso non opera con numeri, ma con matrici*: i vettori e gli scalari sono considerati come particolari matrici.
- Esso *consente la visualizzazione di figure a colori e di grafici bi- e tridimensionali* che possono essere stampati immediatamente o inclusi in documenti Word o TeX.
- MATLAB include vari *moduli software aggiuntivi, chiamati Toolbox* (es. Simulink, Signal Processing Toolbox, Symbolic Math Toolbox) che svolgono compiti particolari.
- Come ottenere MATLAB: www.sid.unipi.it/polo6/studenti/licenze-software/

Come avviare MATLAB



The screenshot shows the MATLAB R2017b - academic use interface. The top menu bar includes HOME, PLOTS, APPS, and a search bar for documentation. Below the menu bar are icons for New Script, New Live Script, New, Open, and Compare. The main workspace area is divided into several sections: Current Folder, Command Window, and Workspace. The Current Folder section shows a file browser with a path highlighted: C: > Users > Nicola > Documents. The Command Window displays a message: "New to MATLAB? See resources for [Getting Started.](#)" and a prompt "fx >>". The Workspace section shows a table with columns for Name and Value.

Command Window (la *finestra dei comandi*): è la finestra che consente all'utente di comunicare con il programma MATLAB; in questa finestra l'utente digita i nomi dei comandi, delle funzioni e delle istruzioni.

Directory di lavoro: vengono visualizzati tutti i file contenuti nella directory corrente.

Nel **Workspace** vengono visualizzate le variabili (arrays) create durante una sessione di lavoro ed immagazzinate in memoria.

Semplici espressioni numeriche

- È possibile valutare direttamente espressioni numeriche lavorando in *modalità interattiva* sulla Command Window.
- Ad esempio, per calcolare il valore di

$$\sqrt{\cos(\pi/4) + [\ln(15)]^2}$$

basta digitare sulla riga di comando:

```
>> sqrt(cos(pi/4)+log(15)^2)
ans =
    2.8356
```

- Il risultato viene memorizzato nella *variabile temporanea ans* che è l'abbreviazione di *answer*.
- E' possibile utilizzare un'altra variabile per immagazzinare il risultato del calcolo precedente, per esempio la variabile **a**:

```
>> a = sqrt(cos(pi/4) + log(15)^2)
a =
    2.8356
```

- **Il MATLAB è case sensitive.** Il primo carattere del nome di una variabile deve essere una lettera dell'alfabeto inglese. Non si devono utilizzare nomi di variabili che inizino con dei numeri o che contengano caratteri del tipo +, -, *, ecc..

Variabili e costanti speciali

<i>Nome</i>	<i>Descrizione</i>
<code>pi</code>	Il numero π
<code>ans</code>	Variabile temporanea che contiene il risultato più recente
<code>eps</code>	Precisione dei numeri decimali ($1+eps > 1$)
<code>i, j</code>	Unità immaginaria ($\sqrt{-1}$)
<code>realmin</code>	E' il più piccolo numero reale positivo rappresentabile
<code>realmax</code>	E' il più grande numero reale positivo rappresentabile
<code>Inf</code>	Infinito (overflow, ossia numero che in valore assoluto è maggiore di <code>realmax</code>)
<code>NaN</code>	Indica un risultato numerico indefinito

- Le variabili `Inf` e `-Inf` si ottengono per overflow, underflow o divisione per zero.
- `NaN` (Not a Number) si ottiene come risultato di operazioni aritmetiche indefinite: $\infty - \infty$, $0 / 0$.
- In MATLAB è possibile eseguire operazioni con i numeri complessi: es. `2+3i`

Formato di visualizzazione dei valori delle variabili

- Il comando **format** determina l'aspetto dei numeri sullo schermo. Il formato di visualizzazione standard è **short**. La tabella elenca le principali opzioni del comando **format**.

<i>Comando</i>	<i>Descrizione ed esempio</i>
format short	4 cifre decimali (formato standard): 10.1751
format long	16 cifre significative: 10.17512397561102
format short e	5 cifre (4 decimali) più l'esponente: (formato standard); 1.0175e+01
format long e	16 cifre (15 decimali) più l'esponente: 1.017512397561102e+01
format bank	2 cifre decimali (calcoli monetari): 10.17
format rat	approssimazione razionale: 2208/217
format +	Positivo, negativo o zero: +

Principali funzioni predefinite (intrinseche)

- Radice quadrata: `sqrt(x)`
- Arrotondamento all'intero più vicino: `round(x)`
- Parte intera di un numero: `fix(x)`
- Segno di un numero: `sign(x)`
- Esponenziale: `exp(x)`
- Logaritmo naturale: `log(x)`
- Logaritmo in base 10: `log10(x)`
- Funzioni trigonometriche: `sin(x)`, `cos(x)`, `tan(x)`, `cot(x)`, ...
- Funzioni iperboliche: `sinh(x)`, `cosh(x)`, `tanh(x)`, `coth(x)`, ...
- Funzioni trigonometriche inverse: `asin(x)`, `acos(x)`, `atan(x)`, `acot(x)`, ...
- Funzioni complesse: `abs(x)`, `conj(x)`, `imag(x)`, `real(x)`, ...

Una funzione deve avere i suoi argomenti racchiusi fra parentesi tonde.

Operatori aritmetici

<i>Simbolo</i>	<i>Descrizione</i>	<i>Formato in MATLAB</i>
^	Elevamento a potenza: a^b	a^b
*	Moltiplicazione: $a*b$	$a*b$
/	Divisione a destra o diretta: $a/b = a:b$	a/b
\	Divisione a sinistra o inversa: $b\backslash a = a:b$	$b\backslash a$
+	Addizione: $a+b$	$a+b$
-	Sottrazione: $a-b$	$a-b$

<i>Livello di precedenza</i>	<i>Operazione</i>
primo	Parentesi: sono valutate a partire dalla coppia più interna
secondo	Elevamento a potenza; valutata da sinistra a destra
terzo	Moltiplicazione e divisione con uguale precedenza; valutate da sinistra a destra
quarto	Addizione e sottrazione con uguale precedenza; valutate da sinistra a destra

Operatori relazionali o di confronto

- **Operatori relazionali o di confronto** (sono 6): servono a confrontare variabili ed array.
- Il risultato di un confronto è una **variabile di tipo logico** che può assumere due valori: **0** (**il confronto è falso**) oppure **1** (**il confronto è vero**).

<i>Operatore</i>	<i>Significato</i>
<	Minore
<=	Minore o uguale
>	Maggiore
>=	Maggiore o uguale
==	Uguale
~=	Diverso (il simbolo tilde ~ si ottiene tenendo premuto Alt e digitando 126)

- **Gli operatori aritmetici hanno la precedenza su quelli relazionali.** Quindi l'istruzione **c=4<2+3** equivale a **c=4<(2+3)** e fornisce il risultato **c=1**, dove **c** è una variabile di tipo logico ed il suo valore **1** significa che il confronto è vero.

Sessione di lavoro

- Tutti i valori delle variabili utilizzate durante una *sessione di lavoro* vengono immagazzinati nella memoria del PC e le variabili stesse sono visualizzate nel *Workspace*.
- Il comando **save** (oppure il *pulsante save Workspace*) consente di salvare il contenuto dell'intero *Workspace* nel file avente il nome *matlab.mat*. E' possibile far seguire al comando **save** un particolare nome del file.
- In una successiva sessione di lavoro il contenuto di una sessione precedentemente salvata può essere ripristinato con il comando **load**.
- Tutto ciò che, durante una sessione di lavoro, appare nella finestra dei comandi (*Command Window*) può essere salvato in un file di testo (*file diario*): usando il comando **diary nomefile** viene avviato il salvataggio dei comandi digitati, mentre con **diary off** si interrompe il salvataggio.
- Tutti i file vengono salvati nella “*current directory*” (current folder) che può essere specificata da ciascun utente all'inizio della sessione di lavoro.

Principali comandi per gestire una sessione di lavoro

<i>Comando</i>	<i>Descrizione</i>
<code>clc</code>	Cancella il contenuto della finestra dei comandi
<code>clear</code>	Elimina tutte le variabili dalla memoria
<code>clear var1 var2</code>	Elimina le variabili <code>var1</code> e <code>var2</code> dalla memoria
<code>who</code>	Elenca le variabili che si trovano attualmente in memoria
<code>whos</code>	Elenca le variabili che si trovano attualmente in memoria indicandone per ognuna di esse la dimensione, il numero di bytes e la classe
<code>exist('nome')</code>	Determina se un file o una variabile hanno il 'nome' specificato
<code>help arg</code>	Visualizza un argomento della guida nella finestra dei comandi
<code>pause(n)</code>	Ferma per <i>n</i> secondi l'esecuzione del programma
<code>quit</code> (o <code>exit</code>)	Chiude MATLAB
<code>,</code>	Separa le istruzioni o gli elementi di una riga di un array
<code>;</code>	Esclude la visualizzazione del risultato di un'istruzione o separa le righe di un array
<code>:</code>	Genera un vettore di elementi regolarmente intervallati
<code>...</code>	Continua l'istruzione nella riga successiva

I comandi, come `clc` e `clear`, non richiedono necessariamente argomenti, ma quando sono richiesti gli argomenti non devono essere racchiusi fra parentesi.

Vettori e matrici

- Per creare un *vettore riga* in MATLAB basta digitare gli elementi all'interno di una coppia di parentesi quadre, separandoli con uno spazio o con una virgola.

```
>> x = [2, 3, 5]
```

```
x =
```

```
     2     3     5
```

```
>> y = [4 1 3]
```

```
y =
```

```
     4     1     3
```

- Per creare un *vettore colonna* si può separare gli elementi con un punto e virgola; in alternativa si può creare il vettore riga e poi utilizzare il simbolo di trasposizione (`'`).

```
>> z = [2; 3; 5]
```

```
z =
```

```
     2
```

```
     3
```

```
     5
```

Vettori e matrici

- *L'operatore due punti* (`:`) consente di creare un vettore di elementi ugualmente intervallati. Ad esempio, l'istruzione

```
>> x = [m:q:n] (se q viene omesso l'incremento è 1)
```

genera un vettore riga **x** formato da valori aventi un *incremento costante* pari a **q**. Il primo valore è **m**; l'ultimo valore è **n** se **n-m** è un intero multiplo di **q**, altrimenti l'ultimo valore è minore di **n**.

- *Un polinomio può essere descritto con un array* i cui elementi sono i coefficienti del polinomio, iniziando dal coefficiente della potenza più elevata. Per esempio il polinomio $4x^3 - 8x^2 + 7x - 5$ è rappresentato dall'array `[4, -8, 7, -5]`. Le radici del polinomio possono essere trovate con la funzione `roots(x)`; il risultato è un vettore colonna che contiene le radici del polinomio.

```
>> a = [4, -8, 7, -5];
```

```
>> roots(a)
```

```
ans =
```

```
1.3880
```

```
0.3060 + 0.8983i
```

```
0.3060 - 0.8983i
```

Vettori e matrici

- Il metodo più semplice per *creare una matrice* è quello di digitare le righe della matrice una dopo l'altra, separando gli elementi di ogni riga con uno spazio (o una virgola) e le righe con un punto e virgola. Per esempio:

```
>> x = [2, 3, 5; 4, 6, 0]
```

```
x =
```

```
    2    3    5
```

```
    4    6    0
```

- E' possibile utilizzare *l'operatore di divisione a sinistra (\) per risolvere un sistema di equazioni algebriche lineari*. Per es. si voglia risolvere il seguente sistema lineare:

$$\begin{cases} 6x_1 - 4x_2 + 8x_3 = 112 \\ -5x_1 - 3x_2 + 7x_3 = 75 \\ 14x_1 + 9x_2 - 5x_3 = -67 \end{cases}$$

A tal fine bisogna creare la matrice dei coefficienti che chiameremo **A** ed il vettore dei termini noti che chiameremo **b**:

```
>> A = [6, -4, 8; -5, -3, 7; 14, 9, -5]
```

```
>> b = [112; 75; -67]
```

```
A\b
```

```
    2
```

```
   -5
```

```
   10
```

Operatori logici o booleani

- *Operatori logici* (sono 3):

<i>Operatore</i>	<i>Nome</i>	<i>Definizione</i>
&	AND	L'istruzione X&Y restituisce un array delle stesse dimensioni di X e Y ; gli elementi del nuovo array sono pari a 1 se i corrispondenti elementi di X e Y sono entrambi diversi da zero, altrimenti sono pari a 0.
 	OR	L'istruzione X Y restituisce un array delle stesse dimensioni di X e Y ; gli elementi del nuovo array sono pari a 1 se almeno uno dei due elementi corrispondenti di X e Y è diverso da zero, sono pari a 0 se entrambi gli elementi di X e Y sono nulli.
~	NOT	L'istruzione ~X restituisce un array delle stesse dimensioni di X ; gli elementi del nuovo array sono pari a 1 se quelli di X sono nulli, altrimenti sono pari a 0 (il simbolo ~ si ottiene tenendo premuto Alt e digitando 126).

- In aggiunta ai tre operatori logici precedenti, esiste la funzione **xor(X,Y)** che implementa l'operazione di "OR esclusivo". La funzione **xor(X,Y)** restituisce un array delle stesse dimensioni di **X** e **Y**; gli elementi del nuovo array sono pari a 1 se uno solo dei due elementi corrispondenti di **X** e **Y** è diverso da zero, sono pari a 0 altrimenti.

Generare i diagrammi

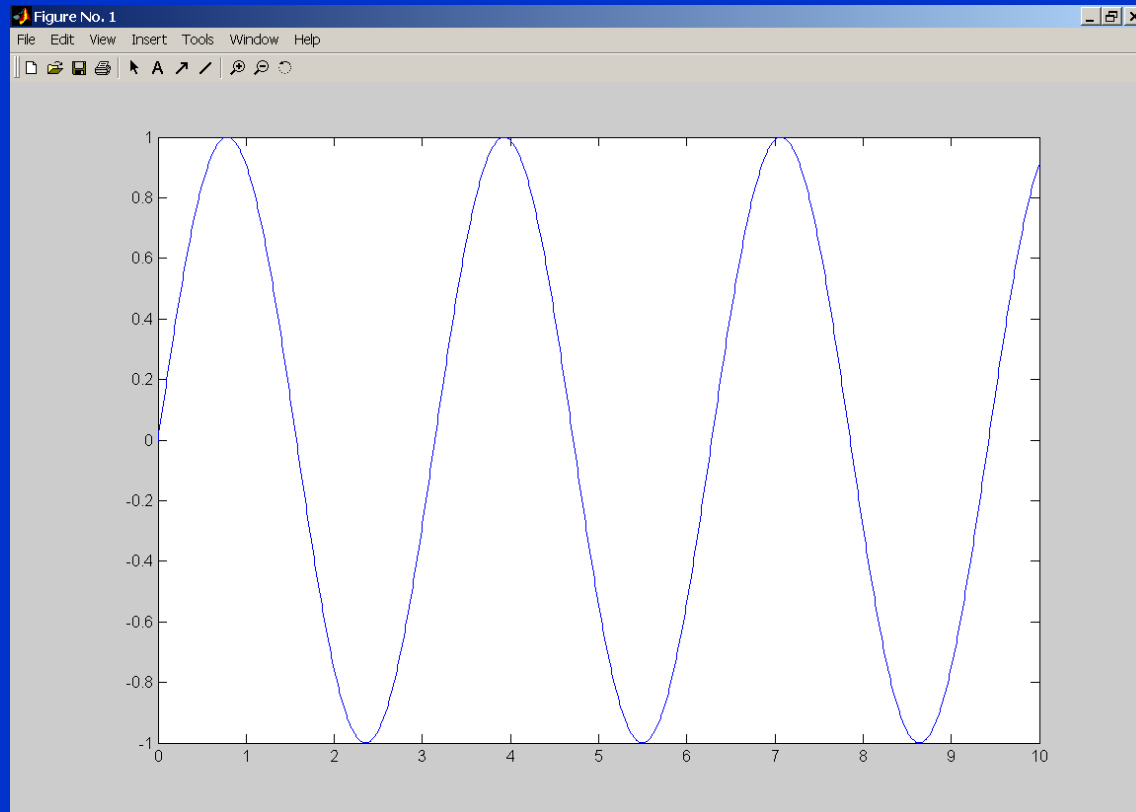
- Supponiamo di dover rappresentare graficamente la funzione $y=\sin(2x)$ per $0 \leq x \leq 10$:

```
>> x = [0:0.01:10];
```

```
>> y = sin(2*x);
```

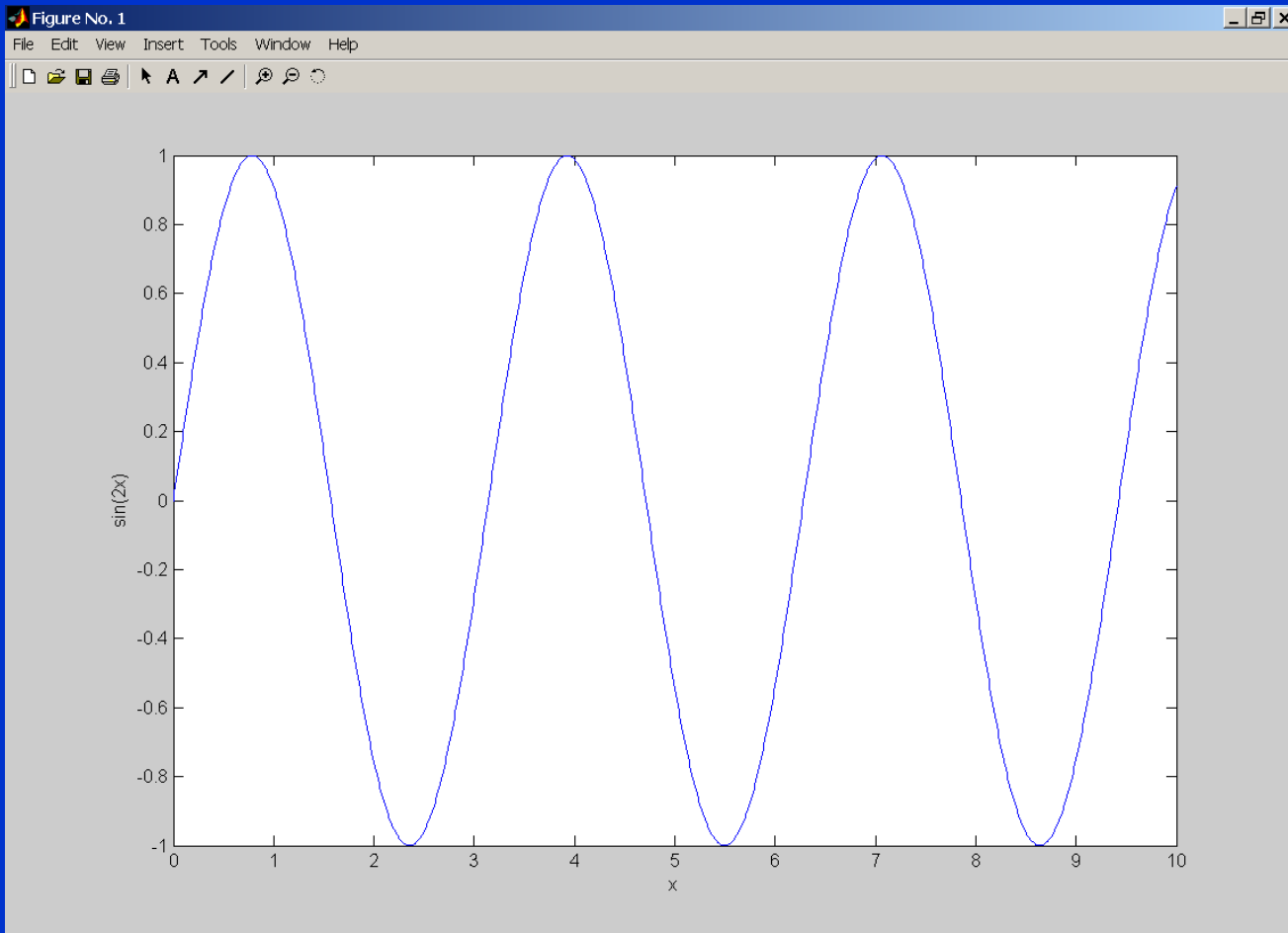
```
>> plot(x,y)
```

Il diagramma appare in un'apposita finestra:



Generare i diagrammi

- Per assegnare dei nomi agli assi ed ottenere lo stesso diagramma della figura precedente basta modificare l'ultima istruzione con la seguente riga:
`>> plot(x,y),xlabel('x'),ylabel('sin(2x)')`



M-file ed Editor/Debugger

- In MATLAB è possibile operare in due modi differenti:
 - *in modalità interattiva* tutti i comandi vengono immessi direttamente nella finestra dei comandi;
 - *eseguendo un M-file*, precedentemente realizzato, cioè un file con estensione *.m* (file script o file di funzione) contenente i comandi MATLAB.
- Operare in modalità interattiva è molto simile ad utilizzare una calcolatrice ed è conveniente soltanto per risolvere problemi semplici.
- Quando un problema richiede numerosi comandi, array con molti elementi, o se occorre ripetere più volte una serie di comandi è più conveniente riportare la sequenza di comandi all'interno di un *M-file*.
- Per creare un file di tipo *M* è possibile utilizzare l'utility *Editor/Debugger* di MATLAB.
- E' possibile eseguire i comandi immagazzinati in un *M-file* semplicemente digitandone il nome sulla linea di comando.
- *E' molto importante dare un nome significativo al file script, evitando di dare un nome coincidente con il nome di una variabile intrinseca o con quello di un comando MATLAB. Non bisogna inserire caratteri speciali nel nome del file e non bisogna iniziare il nome del file con un numero.*

M-file ed Editor/Debugger

- **Il simbolo % indica un commento** e tale simbolo può essere posto in un punto qualsiasi della riga di un *M-file*. MATLAB ignora tutto ciò che è posto a destra del carattere %.
- **Il simbolo % di commento vale solo per una riga**. Se il commento sta su più righe occorre mettere il simbolo % all'inizio di ciascuna riga.
- Ogni file inizia generalmente con una serie di commenti che ne presentano il contenuto: è la **parte dichiarativa del file**. Se al termine della parte dichiarativa, prima dell'inizio dei comandi o istruzioni, si lascia una riga vuota (senza neanche il simbolo % all'inizio), dalla finestra dei comandi si potrà vedere il contenuto di questa parte digitando **help nome** essendo **nome.m** il nome del file.
- Realizzare, come **esempio di M-file**, il grafico della funzione:

$$y = \frac{\sin(x)}{x}$$

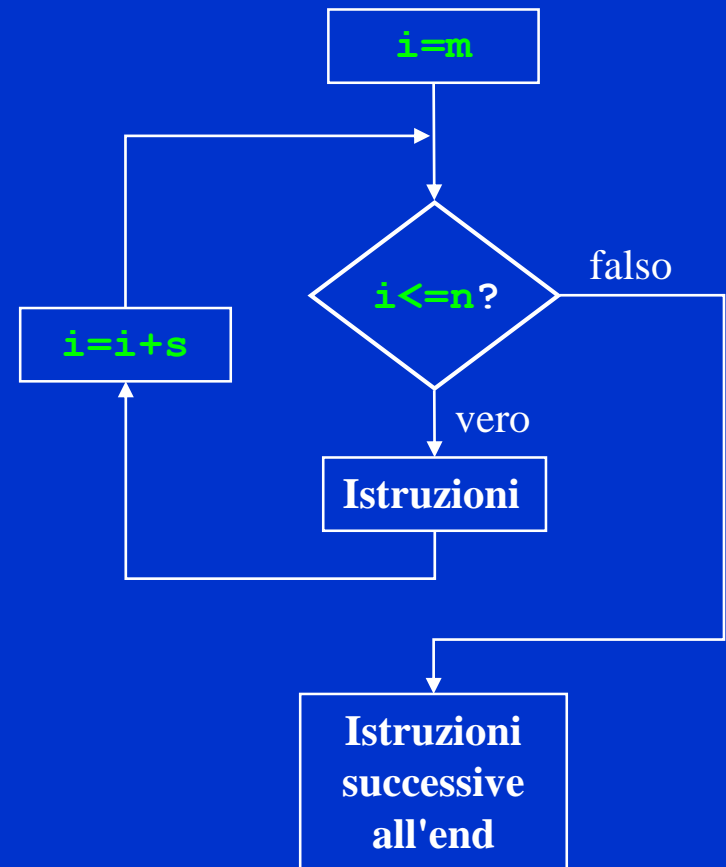
nell'intervallo $[\pi/100; 10 \pi]$.

Cicli

- **Un ciclo o loop è una struttura che permette di ripetere i calcoli un certo numero di volte.** Ogni ripetizione del ciclo si chiama passaggio. La forma tipica di un **ciclo for** è la seguente:

```
for variabile di ciclo = m:s:n  
  istruzioni  
end
```

L'espressione **m:s:n** assegna il valore iniziale **m** alla **variabile di ciclo**, che viene incrementata del valore **s** (valore di step). Le istruzioni vengono eseguite una volta per ogni passaggio, utilizzando il valore corrente della variabile di ciclo. L'elaborazione continua finché la variabile di ciclo non supera il valore finale **n**.



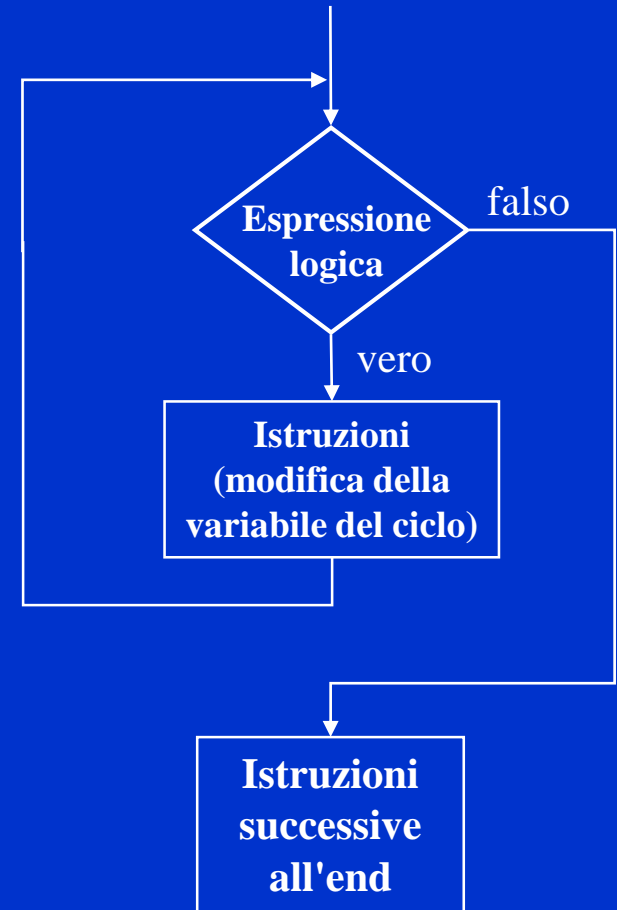
Cicli

- Il **ciclo while** si usa quando non si conosce in anticipo il numero di passaggi da effettuare. La forma tipica di un ciclo while è la seguente:

```
while espressione logica
  istruzioni
end
```

Se l'espressione logica è vera le istruzioni vengono eseguite. Affinché il ciclo while possa operare correttamente devono verificarsi due condizioni:

- la **variabile di ciclo** deve avere un valore prima che venga eseguita l'istruzione while;
 - la variabile di ciclo deve essere modificata in qualche maniera dalle istruzioni interne al ciclo.
- L'istruzione **"break"** termina forzatamente l'esecuzione di un ciclo for o di un ciclo while. Nel caso vi siano più cicli annidati break termina l'esecuzione del ciclo più interno.



Istruzioni condizionali

- **Istruzione if:** la forma tipica è la seguente:

```
if espressione logica
    istruzioni
end
```

Esempio:

```
if x >= 0
    y = sqrt(x);
end
```

L'espressione logica può essere un'espressione composta e le istruzioni possono essere formate da una serie di comandi eventualmente disposti su più righe.

```
if (x >= 0) & (y >= 0)
    z = sqrt(x) + sqrt(y);
    w = z + 5;
end
```



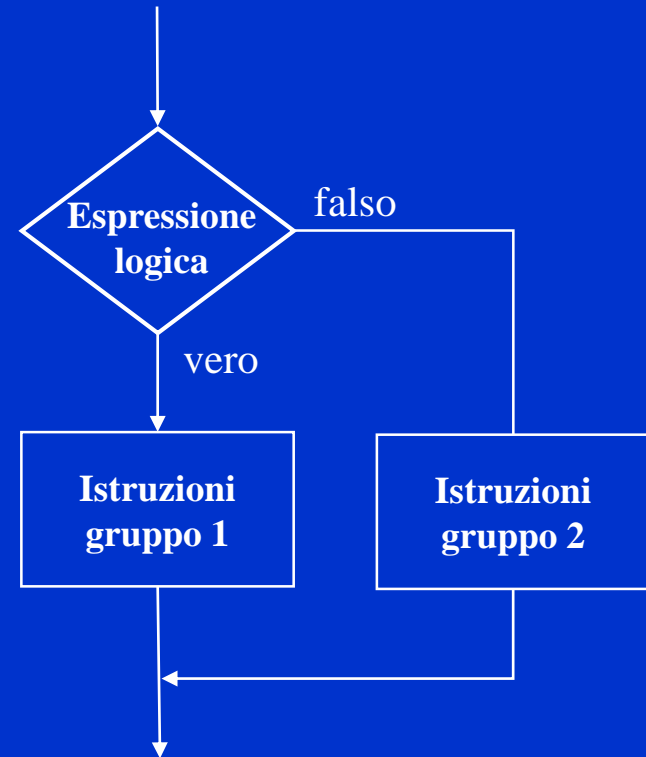
Istruzioni condizionali

- **Istruzione if+else**: la forma tipica è la seguente:

```
if espressione logica
    istruzioni del gruppo 1
else
    istruzioni del gruppo 2
end
```

Esempio:

```
if x >= 0
    y = sqrt(x);
else
    y = exp(x) - 1;
end
```



Istruzioni condizionali

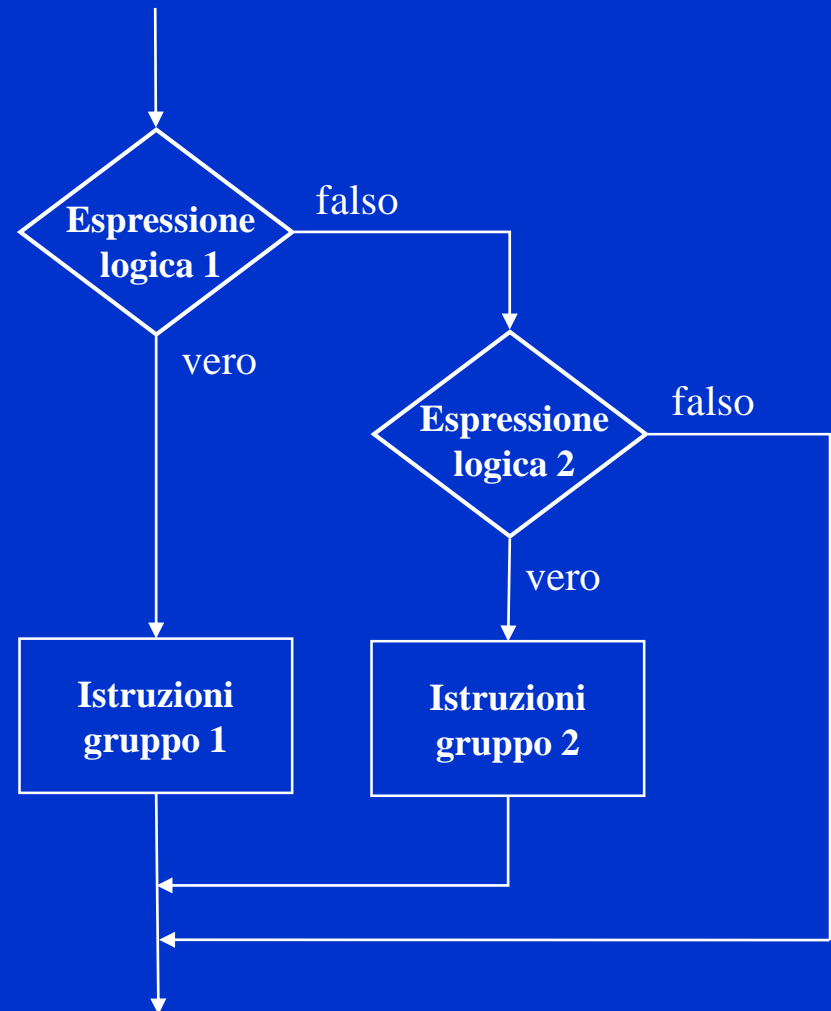
- *Istruzione if+elseif*: la forma tipica è la seguente:

```
if espressione logica 1
  istruzioni del gruppo 1
elseif espressione logica 2
  istruzioni del gruppo 2
end
```

Esempio:

```
if x >= 0
  y = sqrt(x);
elseif x > -10
  y = exp(x) - 1;
end
```

N.B.: *elseif non deve essere scritto else if cioè con lo spazio tra else ed if.*



La struttura switch

- **Struttura switch:** la forma tipica è la seguente:

```
switch espressione di input %(scalare o stringa)
    case valore1
        istruzioni del gruppo 1
    case valore2
        istruzioni del gruppo 2
    ....
    otherwise
        istruzioni del gruppo n
end
```

Esempio:

```
switch angolo
    case 90
        disp('Nord')
    case 270
        disp('Sud')
    otherwise
        disp('Direzione sconosciuta')
end
```

Funzioni definite dall'utente (function)

- Un altro tipo di *M-file* sono i *file di funzione*. Diversamente dai **file script** tutte le *variabili di un file di funzione sono locali*; questo significa che i loro valori sono disponibili soltanto all'interno della funzione.
- La prima riga di un file di funzione *deve* iniziare con la *definizione della funzione* che può contenere un elenco di *variabili di input (racchiuse tra parentesi tonde) e variabili di output (racchiuse tra parentesi quadre)*:

```
function [out1,out2,...]=nome_funzione(in1,in2,...);
```
- Il nome della funzione (*nome_funzione*) deve essere uguale al nome del file in cui sarà salvata la funzione.
- Una funzione può avere una o più variabili di output, oppure può non avere variabili di output.

<i>Riga di definizione della funzione</i>	<i>Nome del file</i>
<pre>function [area]=quadrato(lato);</pre>	quadrato.m
<pre>function area=quadrato(lato);</pre>	quadrato.m
<pre>function [area,circonf]=cerchio(raggio);</pre>	cerchio.m
<pre>function grafquad(lato);</pre>	grafquad.m

Funzioni definite dall'utente

- La seguente funzione, di nome *drop*, calcola la velocità e la distanza percorsa da un corpo che viene lasciato cadere con una velocità iniziale v_0

```
function [dist,vel] = drop(g,v0,t);  
% Calcola la velocità e la distanza percorsa da un corpo in caduta  
% in funzione di g, della velocità iniziale v0 e del tempo t  
vel = g * t + v0;  
dist = 0.5 * g * t.^2 + v0 * t;  
end
```

I seguenti esempi mostrano i modi di chiamare la funzione drop:

1. non è necessario utilizzare nella chiamata gli stessi nomi delle variabili specificati nella definizione della funzione:

```
acc = 9.81;  
vel0 = 10;  
tempo = 2;  
[distanza,veloc] = drop(acc,vel0,tempo)
```

2. non occorre necessariamente assegnare dei valori alle variabili di input prima di chiamare la funzione:

```
[distanza,veloc] = drop(9.81,10,2)
```

Trovare gli zeri di una funzione

- Il comando `fzero` consente di trovare lo zero di una *funzione di una sola variabile* (**x**). Una forma della sua sintassi è:

```
x = fzero('nome_funzione',x0);
```

Il comando `fzero` tenta di trovare uno zero della funzione in prossimità dello scalare `x0`. Esso non può trovare zeri complessi. Per le funzioni senza zeri validi, `fzero` continua la ricerca finché non trova un valore `Inf`, `NaN` o un valore complesso della funzione.

- Se `x0` è un vettore di lunghezza 2 `fzero` presuppone che `x0` sia un intervallo dove il segno della funzione in `x0(1)` differisce dal segno della funzione in `x0(2)`. Se questo non è vero esso restituisce un messaggio di errore.

- Il formato alternativo:

```
[x,fval] = fzero('nome_funzione',x0);
```

restituisce anche il valore della funzione (`fval`) in corrispondenza di `x`.

- Il formato:

```
[x,fval,exitflag] = fzero('nome_funzione',x0);
```

restituisce anche un valore di `exitflag` che descrive la condizione di uscita di `fzero`. Un valore positivo di `exitflag` indica che `fzero` ha trovato uno zero della funzione, mentre un valore negativo indica che il comando `fzero` non è riuscito a trovare uno zero.

- `fsolve` è usato al posto di `fzero` per risolvere sistemi di equazioni non lineari.

Trovare il minimo (massimo) di una funzione

- Il comando `fminbnd` consente di trovare il minimo di una *funzione di una sola variabile* (x). Una forma della sua sintassi è:

```
x = fminbnd('funzione', x1, x2);
```

Il comando `fminbnd` restituisce un valore di x che rende minima la funzione nell'intervallo $x1 \leq x \leq x2$. Essa prima ricerca un punto di minimo in corrispondenza del quale la pendenza della funzione è nulla; se ne trova uno si ferma. Se non lo trova esamina i valori della funzione agli estremi dell'intervallo specificato.

- Per esempio `fminbnd('cos', 0, 4)` restituisce il valore $x=3.1416$.
- Se la funzione è definita all'interno di un file di funzione, ad esempio:

```
function y = f2(x);
```

```
y = 1 - x .* exp(-x);
```

per trovare il valore di x che rende minima tale funzione nell'intervallo $0 \leq x \leq 5$, basta porre:

```
x = fminbnd('f2', 0, 5);
```

```
y = f2(x);
```

Si ottiene il minimo ad $x=1$, mentre `y=f2(x)` fornisce il valore assunto dalla funzione in corrispondenza del suo minimo: il risultato è $y=0.6321$.

- `fminsearch` è usato per trovare il minimo di una funzione di più variabili.