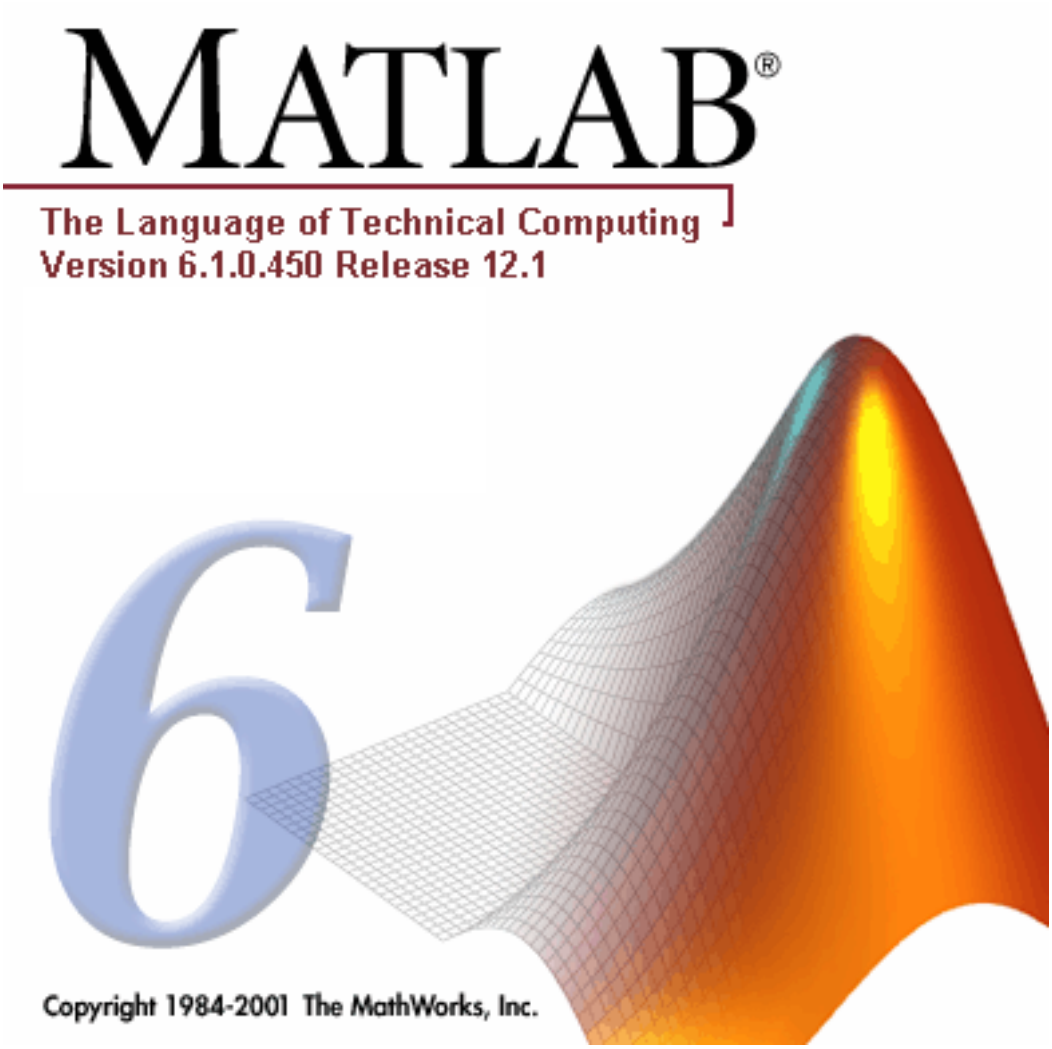


Introduzione all'uso di Matlab

Mario Cesarelli

Elaborazione di Dati e Segnali Biomedici

Corso di Laurea in Ingegneria Biomedica

Rev.1 Anno accademico 2004-05

Rev.2 Anno accademico 2005-06

Rev.3 Anno accademico 2012-13

Indice

Introduzione a Matlab

1.1 Matlab come ambiente di programmazione

Matlab non è un semplice linguaggio di programmazione, ma un ambiente interattivo che integra in modo efficiente il calcolo, la visualizzazione e la programmazione. L'ambiente Matlab consente di gestire variabili, importare ed esportare dati, svolgere calcoli, disegnare grafici, programmare istruzioni e sviluppare applicazioni (tra cui la costruzione grafica dell' interfaccia utente).

1.2 Matlab e l'algebra lineare

Matlab è stato sviluppato specificamente per applicazioni basate su matrici e algebra lineare, nell'ambito dell'analisi numerica. Il termine Matlab deriva appunto da **MAT**rix **LAB**oratory.
Tale specificità rende Matlab uno strumento particolarmente versatile ed efficiente per l'elaborazione dei dati e lo sviluppo di modelli. Infatti, Matlab permette di trattare grandi insiemi di dati come singole variabili chiamate array e di effettuare calcoli complessi tramite poche righe di codice. Per esempio, è possibile risolvere un sistema di equazioni lineari con sole tre righe di codice.

1.3 I moduli aggiuntivi di MATLAB

Matlab prevede moduli aggiuntivi denominati Toolbox sviluppati per svolgere compiti specializzati. La lista completa dei Toolbox attualmente disponibili è consultabile nel sito:
http://www.mathworks.it/products/products_by_category.shtml

1.4 La sessione di lavoro in Matlab

1.4.1 Avviare Matlab

Per avviare Matlab sotto MS Windows basta fare doppio clic sull'icona:

Fatto ciò, apparirà il desktop di Matlab, che prevede tre finestre:

- Launch Pad: strumenti di Matlab
- Command History: cronologia dei comandi
- Command Window: la finestra dei comandi

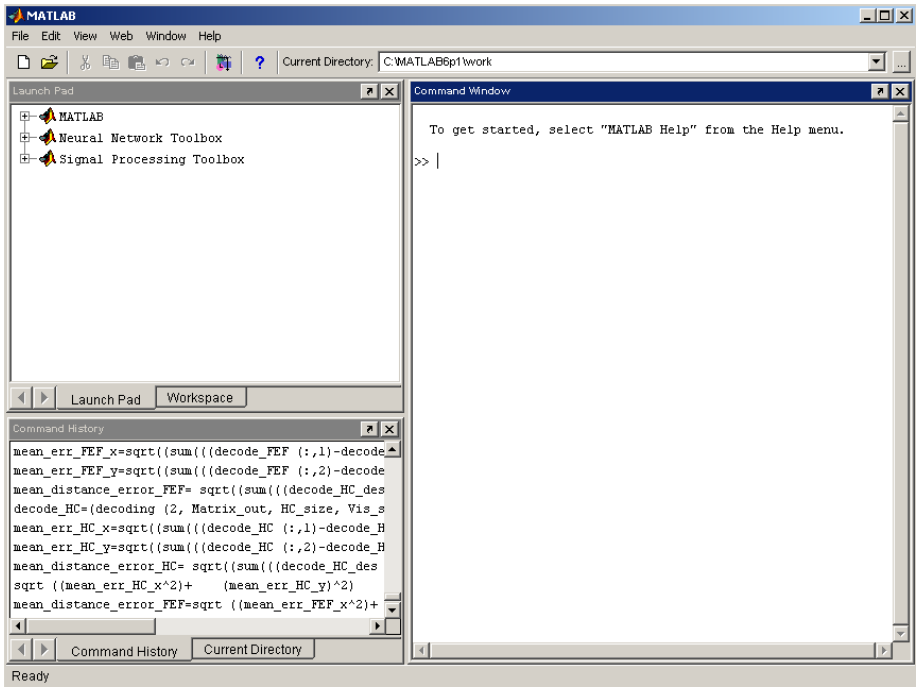


Figura 1. Il dekstop di Matlab.

La parte superiore del desktop presenta la barra dei menu (File, Edit, View, Web, Window, Help) e la barra degli strumenti costituita da una serie di icone, che permettono di accedere ad un particolare comando attraverso un singolo clic del mouse.
A destra della barra degli strumenti si trova un riquadro denominato "Current Directory", che indica la cartella di lavoro corrente in cui Matlab cerca e salva i file.

Figura 2. La barra dei menu e degli strumenti.

1.4.2 La finestra dei comandi

Permette all'utente di dialogare con il programma. Attraverso questa finestra, l'utente può digitare i comandi, le funzioni e le istruzioni che devono essere eseguite.
Matlab è un programma a riga di comando: il simbolo >> (EDU>>, nella versione per studenti), che compare nella finestra dei comandi, è il prompt di Matlab e indica che il programma è pronto a ricevere l'input dall'utente. Prima di immette un'istruzione è necessario accertarsi che il cursore si trovi subito dopo il prompt.

1.4.3 La finestra Launch Pad

Visualizza l'elenco dei Toolbox e di altri programmi Matlab installati nel computer, attraverso una finestra simile ad Esplora Risorse.

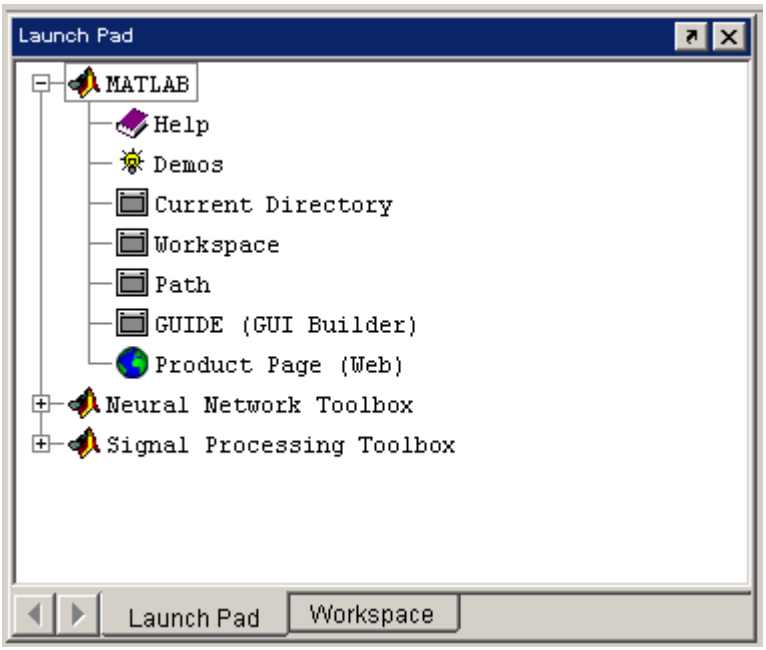


Figura 3. La finestra Launch Pad.

Facendo clic sul simbolo + a sinistra dell'icona di Matlab si aprirà una lista di cartelle, file e collegamenti il cui accesso è molto utile e frequente durante una sessione di lavoro.

1.4.4 La finestra Command History

Visualizza tutti i comandi precedentemente immessi nella finestra dei comandi.

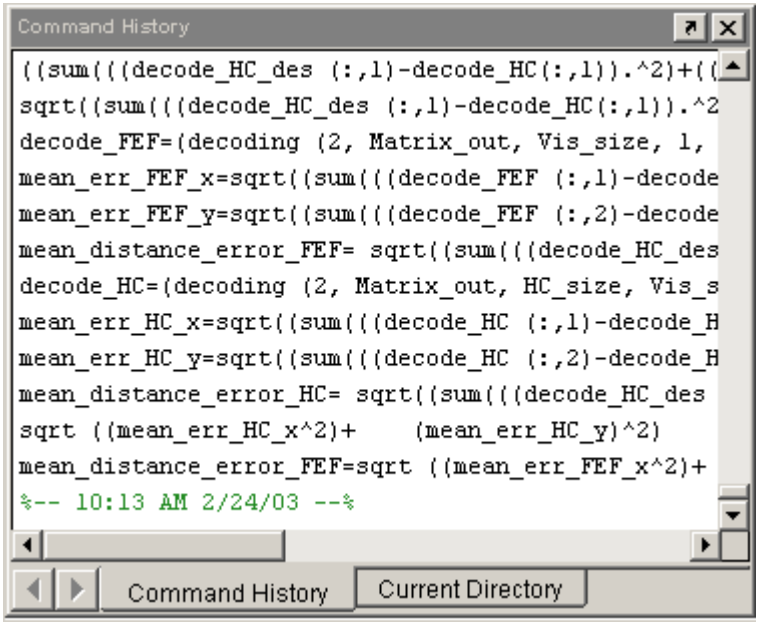


Figura 4. La finestra Command History.

È utile per richiamare velocemente un comando o un'istruzione digitata durante la sessione di lavoro. È possibile copiare un comando direttamente nella finestra dei comandi semplicemente selezionandolo e trascinandolo con l'aiuto del mouse.
Il desktop di Matlab può essere facilmente modificato in base alle proprie esigenze; per esempio, per nascondere la finestra Command History, è sufficiente fare clic sul simbolo nell'angolo superiore destro. È possibile, inoltre, disancorare una finestra dal desktop di Matlab cliccando sul simbolo . Per tornare alla configurazione standard bisogna aprire il menu View, cliccare sull'opzione Desktop Layout e selezionare Default.

1.4.5 Immissione dei comandi attraverso tastiera

In questo paragrafo utilizzeremo il simbolo (I) per indicare la pressione del tasto Invio. Tale convenzione sarà abbandonata nei paragrafi successivi.
Matlab può essere utilizzato come una semplice calcolatrice. Questa modalità viene chiamata sessione interattiva. Per esempio, digitando:

```
>> 10/3 (I)
```

sullo schermo comparirà la risposta di Matlab:

```
ans =
    3.3333
```

La risposta viene assegnata automaticamente ad una variabile temporanea che Matlab chiama ans (abbreviazione di answer). Una variabile è un simbolo utilizzato per contenere e memorizzare un valore.
Si può utilizzare la variabile ans per eseguire altri calcoli:

```
>> 10*ans (I)
```

```
ans =
    33.3333
```

Matlab normalmente visualizza il risultato di un'operazione con quattro cifre decimali (formato short). Successivamente verrà spiegato come modificare il formato di output.
Si noti che l'ultima istruzione ha modificato il valore contenuto nella variabile ans: attualmente non esiste alcuna variabile in cui è memorizzato il valore precedente (3.3333).
Per non perdere i dati è pertanto necessario utilizzare variabili definite dall'utente. Una variabile scalare (una variabile contenente un singolo valore) viene definita utilizzando l'operatore di assegnazione **=**. Per esempio l'istruzione:

```
>> A=2 (I)
A =
        2
```

assegna il valore '2' (a destra dell'operatore) alla variabile indicata a sinistra 'A'.
E' molto importante distinguere l'operatore di assegnazione ed il simbolo uguale utilizzato in matematica. La variabile nel lato sinistro dell'operatore è il "contenitore" di destinazione del valore generato dall'espressione del lato destro. Tale ordine non può MAI essere invertito; pertanto, l'espressione :

```
>> 6 = ans (I)
```

o l'espressione:

```
>> A + 1 = 2 (I)
```

generano un messaggio d'errore.
Matlab conserva in memoria una variabile finché non viene esplicitamente cancellata con il comando clear seguito dal nome della variabile:

```
>> clear A (I)
```

Il comando clear può essere seguito dal nome di più variabili:

```
>> clear A B C (I)
```

o da nessuna variabile:

```
>> clear (I)
```

in tal caso cancella tutte le variabili in memoria.
Si noti che Matlab distingue tra lettere maiuscole e minuscole: **A** e **a** sono due variabili diverse!
Le variabili possono essere utilizzate per scrivere espressioni matematiche. Per esempio, se utilizziamo la variabile B per registrare il risultato di un'operazione:

```
>> B = 8/10 (I)
B =
   0.8000
```

possiamo utilizzarla successivamente per altre operazioni:

```
>> C = 20*B (I)
C =
   16
>> D = ans + C (I)
D =
   49.3333
```

In Matlab è anche possibile utilizzare l'operatore di assegnazione per scrivere istruzioni auto-referenziali:

```
>> C = C+100 (I)
C =
   116
```

Questa istruzione permette di aggiungere 100 al valore corrente di C. Eseguita l'operazione, non rimarrà traccia in memoria del precedente valore di C.
Il lato destro dell'operatore di assegnazione deve essere calcolabile, se vi compare una variabile non definita, Matlab genera un messaggio d'errore.

Suggerimento: gli spazi nelle espressioni matematiche vengono ignorati da Matlab nell'esecuzione dei calcoli; è consigliabile, tuttavia, inserirli per semplificare la lettura delle istruzioni digitate.

L'operatore di assegnazione permette di utilizzare valori che non sono noti in anticipo o per cambiare il valore di una variabile attraverso una determinata procedura.
Se vogliamo visualizzare il valore corrente di in una determinata variabile, per esempio ans, basta digitarne il nome e premere invio:

```
>> ans (I)
ans = 33.3333
```

Matlab utilizza i simboli di programmazione standard **+ - * / ^** per eseguire rispettivamente la somma, la sottrazione, la moltiplicazione, la divisione e l'elevamento a potenza. Inoltre utilizza il simbolo \ (backslash) per la divisione sinistra o inversa. È possibile combinare a piacere tali simboli per scrivere espressioni matematiche di varia lunghezza:

```
>> x = C+5-1^2 (I)
x = 20
```

Si noti che è molto diverso digitare:

```
>> x = (C+5-1)^2 (I)
```

Il risultato in questo caso sarà:

```
 x =
  576
```

Regole di precedenza Le operazioni matematiche sono eseguite rispettando una serie di regole di precedenza:

1. elevazione a potenza (livello più alto di precedenza)

2. moltiplicazione e divisione
3. addizione e sottrazione

A parità di livello di precedenza, le espressioni matematiche sono valutate da destra a sinistra.
Nell'esempio non cambia l'ordine dei simboli, ma l'ordine di precedenza è modificato dalla presenza delle parentesi. La valutazione delle parentesi procede da quelle più interne a quelle più esterne.

Livello di precedenza	**Operazione**
Primo	Parentesi; dalla coppia più interna
Secondo	Elevazione a potenza; da sinistra a destra
Terzo	Moltiplicazione e divisione; da sinistra a destra
Quarto	Addizione e sottrazione; da sinistra a destra

Tabella 1. Regole di precedenza.

Suggerimento: per evitare errori quando si scrivono espressioni molto lunghe è sempre meglio utilizzare le parentesi tonde, anche quando non sono necessarie.

Per calcolare espressioni matematiche più complesse, Matlab include centinaia di funzioni. Una di queste è la funzione sqrt (square root) per il calcolo della radice quadrata.
In generale, per utilizzare una funzione bisogna digitarne il nome seguito dal valore (argomento) passato alla funzione racchiuso tra parentesi **tonde**:

```
nome_funzione (argomento)
```

La seguente istruzione permette di calcolare la radice quadrata di 9 e di registrare il risultato in una nuova variabile chiamata r:

```
>> r = sqrt (9)
r =
  3.0000
```

1.4.6 I nomi delle variabili

Matlab lascia ampia libertà nella definizione dei nomi delle variabili: devono iniziare con una lettera e possono contenere lettere, cifre e caratteri di sottolineatura, senza superare i 32 caratteri. Come ricordato in precedenza Matlab è case sensitive, ovvero considera diversi i caratteri maiuscoli dai minuscoli.

Suggerimento: per Matlab qualsiasi combinazione alfanumerica che rispetti la sintassi per nominare le variabili è identica. Tuttavia, è molto utile sfruttare la libertà lasciata per creare nomi di variabili significativi in modo da facilitare la comprensione del programma e le modifiche successive.

1.4.7 La finestra Workspace

Nella visualizzazione standard del desktop di Matlab, la finestra Workspace è nascosta dalla finestra Launch Pad ed è accessibile facendo clic sulla corrispondente etichetta:

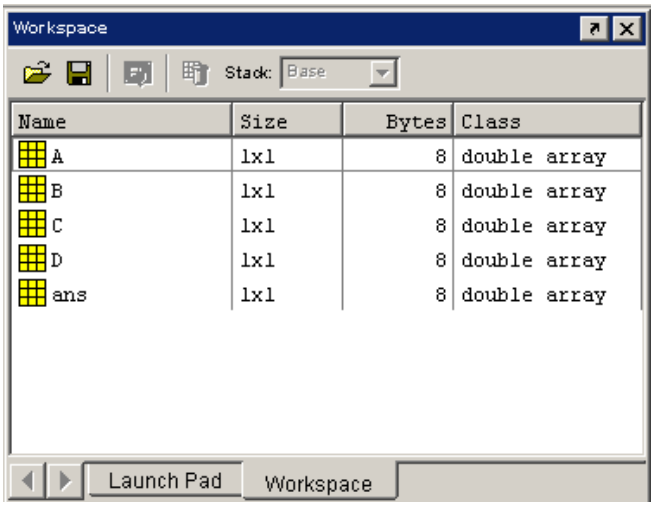


Figura 5. La finestra Workspace.

La finestra Workspace riporta tutte le variabili caricate nella sessione di lavoro corrente. Cliccando sul nome di una variabile è possibile visualizzarne il valore.

1.4.8 Gestire una sessione di lavoro

Alcuni comandi e simboli speciali sono molto utili nella gestione di una sessione di lavoro. Il simbolo ; (punto e virgola) digitato alla fine di un'espressione consente di non visualizzarne sullo schermo i risultati. La visualizzazione, infatti, richiede un tempo molto maggiore dei calcoli corrispondenti, col risultato di ritardare pesantemente una sessione di lavoro quando si effettuano calcoli su matrici con molti elementi.
È possibile digitare comandi diversi nella stessa riga separandoli con il simbolo , (virgola) se si vuole visualizzare il risultato, oppure con il simbolo ; se non si desidera visualizzarlo.
Per esempio, digitando la seguente riga di comando:

```
>> x = 9; y = x + 1, x = y*3
```

Matlab visualizzerà:

```
y =
  10
x =
  30
```

Il primo valore di x non viene visualizzato.
Se un'istruzione è molto lunga si può utilizzare l'operatore di continuazione di riga (tre puntini) per andare a capo:

```
>> E = sum ( dot (visibleState, log(visibleState + 0. 01)) …
+ dot (1-visibleState,log(1-visibleState + 0. 05)));
```

Matlab memorizza i comandi digitati in precedenza. Utilizzando i tasti ↑ e ↓ è possibile richiamare e scorrere tutti i comandi precedenti. Digitando i primi caratteri di una variabile e utilizzando i tasti ↑ e ↓, vengono richiamate tutte le istruzioni precedenti che iniziano con i caratteri immessi. Inoltre, è possibile completare automaticamente il nome di una variabile, funzione o file, digitando i primi caratteri e premendo il tasto Tab.

1.4.9 I menu di Matlab

Matlab prevede 6 menu: File, Edit, View, Web, Window e Help. Le opzioni presenti nei menu cambiano in funzione della finestra attiva. Alcune opzioni possono essere selezionate velocemente attraverso i tasti di scelta rapida indicati a destra dei nomi delle opzioni.
I menu più importanti sono: File e Edit.

1.4.9.1 Il menu file

New	apre una finestra di dialogo per la creazione di file di programma (M-file, Figure, Model, GUI)
Open	apre una finestra di dialogo per aprire un file di programma
Close nome_finestra	chiude la finestra dei comandi attiva
Import Data	avvia un'utilty per l'importazione dei dati
Save Workspace As	apre una finestra di dialogo per salvare tutte le variabili (nomi e valori) caricate in memoria in un unico file
Set Path	apre una finestra di dialogo per impostare il percorso di ricerca dei file
Preferences	apre una finestra di dialogo per impostare diversi parametri di Matlab
Print	apre una finestra di dialogo per stampare i testi presenti nella finestra di dialogo
Print Selcted	apre una finestra di dialogo per stampare le parti selezionate nella finestra di dialogo
File List	visualizza un elenco cronologico dei file precedentemente aperti
Exit Matlab	chiude il programma

1.4.9.2 Il menu Editor

Undo	annulla l'ultima operazione
Redo	annulla l'ultima operazione Undo
Cut	taglia e registra negli appunti di Windows (clipboard)
Copy	registra negli appunti di Windows
Paste	inserisce il contenuto del clipboard
Paste Special	inserisce il contenuto del clipboard sotto forma di una o più variabili
Select All	seleziona tutto
Delete	cancella la variabile selezionata nella finestra Workspace
Clear Command Windows	cancella tutti i testi della finestra dei comandi
Clear Command History	cancella tutti i testi della finestra Command History
Clear Workspace	cancella tutte le variabili memorizzate

1.4.10 La guida di Matlab

Trattare tutte le funzionalità e potenzialità di Matlab in un manuale è praticamente impossibile. Fortunatamente, Matlab mette a disposizione una guida molto dettagliata, che permette di eseguire ricerche e visualizzare informazioni avanzate su tutti gli argomenti relativi al suo funzionamento.
È possibile accedere alla guida di Matlab attraverso l' Help Browser, la funzione help, la funzione lookfor e la funzione doc.

1.4.10.1 L'Help Browser

Permette di eseguire ricerche nella documentazione di Matlab e di visualizzarne i risultati. Vi si accede dal menu Help, oppure cliccando l'icona  :

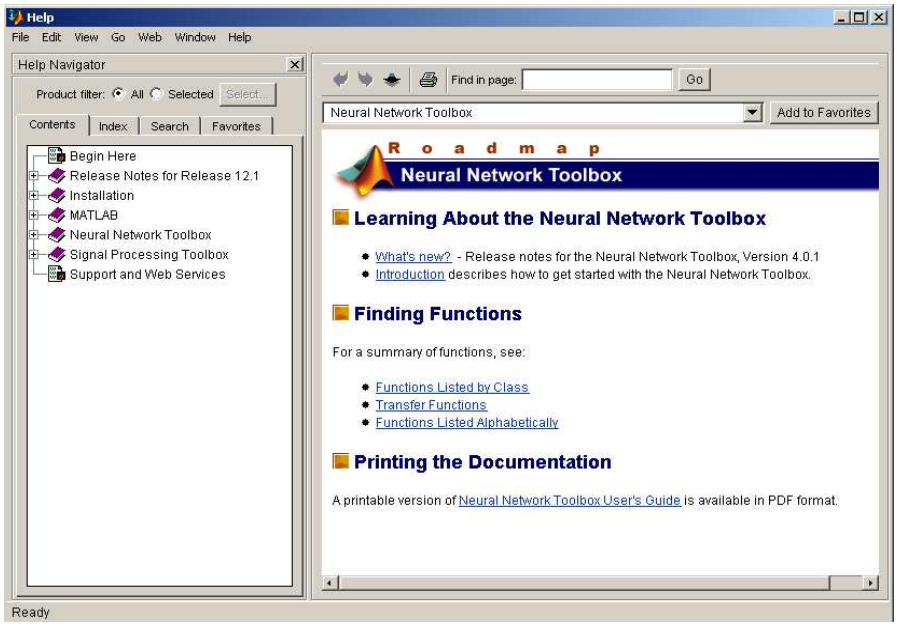


Figura 6. Figura 6. L'Help di Matlab.

La finestra Help Browser è suddivisa in due pannelli: l'Help Navigator, a sinistra, che consente di inserire i parametri di ricerca, ed il pannello di visualizzazione, a destra, in cui vengono visualizzati i documenti che soddisfano i criteri di ricerca.

1.4.10.2 La funzione Help

È il metodo più veloce per visualizzare informazioni riguardanti la sintassi ed il comportamento di una particolare funzione. Per accedervi è sufficiente digitare:

```
>> help nome_funzione
```

dalla finestra dei comandi.
Per esempio, digitando help sin, Matlab visualizzerà:

```
SIN    Sine.
       SIN(X) is the sine of the elements of X.
```

Esercizio Digitare help tanh per visualizzare le informazioni relative alla funzione che permette di calcolare la tangente iperbolica. Digitare help rand per visualizzare le informazioni relative alla funzione che permette di generare numeri casuali.

1.4.10.3 La funzione lookfor

Consente di effettuare ricerche basate su una parola chiave. È molto utile quando **non** si conosce il nome esatto di una funzione di Matlab. Per esempio, se cerchiamo informazioni sulla funzione che consente di calcolare la radice quadrata e digitiamo help square Matlab visualizza informazioni sulla funzione che permette di generare onde quadre. Utilizzando la funzione lookfor verrà visualizzato un elenco di funzioni con le relative descrizioni, tra cui la funzione sqrt che stiamo cercando. La ricerca avviene sulla prima riga di testo di ogni funzione, detta H1, e restituisce tutte le funzioni che contengono in H1 la parola chiave specificata. Per utilizzare la funzione digitare:

```
lookfor parola_chiave
```

dalla finestra dei comandi.

Esercizio Utilizzare la funzione lookfor per cercare il nome della funzione che permette di trasformare le coordinate Cartesiane in coordinate polari e utilizzare la funzione help per visualizzarne la sintassi.

1.4.10.4 La funzione doc

Digitando:

```
doc nome_funzione
```

dalla finestra dei comandi, Matlab apre automaticamente l'Help Browser visualizzando la prima pagina della documentazione relativa alla funzione indicata.

Esercizio Utilizzare la funzione doc per visualizzare la documentazione relativa alla funzione pol2cart che permette di trasformare le coordinate polari in coordinate Cartesiane.

Problemi

1.1) Avviate e chiudete una sessione di lavoro. Utilizzate Matlab per effettuare i seguenti calcoli dati x=10 e y=3:

1) u = x + y [13]
2) v = xy [30]
3) w = x/y [3.3333]
4) z = sin x [-0.5440]
5) r = 8 sin x [-4.3522]
6) s = 5 sin (2y) [-1.3971]
7) $p = x^{y}$ [1000]
8) $k = x^{y/x}$ [1.9953]
9) $f = \sqrt{(\sin(x + y)}$ [0.6482]
10) $g = \frac{x}{5\sqrt{y}}$ [1.1547]

1.2) Trovare le funzioni di Matlab che permettono di calcolare la media e la deviazione standard di un insieme di dati.

1.3) Calcolare il logaritmo naturale di 24 [3.1781] ed i logaritmo in base 2 di 73 [6.1898].

Vettori, Matrici, Strutture

Il principale punto di forza di Matlab è la capacità di trattare ampi insiemi numerici come singole variabili chiamate array. Tale peculiarità permette di scrivere programmi con poche righe di codice, pertanto facili da scrivere, correggere e documentare.
L'utilizzo degli array è la soluzione migliore per risolvere problemi che richiedono di analizzare e gestire ampi insiemi di dati.
Un array è un insieme, una lista ordinata di elementi. L'ordine degli elementi è la caratteristica fondamentale degli array. In altre parole, non conta solo l'identità degli elementi contenuti nell'array, ma anche la loro posizione. In generale, gli array sono costituiti da righe e colonne, che formano una griglia di indicizzazione, che permette di identificare in maniera univoca gli elementi in essi contenuti. Quando un array è formato da una singola riga o da una singola colonna si parla rispettivamente di **vettore riga**:

34 98 23

e **vettore colonna**:

12

1

128

Quando un array è formato da almeno due righe e due colonne prende il nome di **matrice**.
Ecco un esempio di matrice 3x2:

2 4
1 5
7 3

In questa sezione vedremo come creare e modificare gli array e come effettuare operazioni tra di essi. Per semplicità, tratteremo separatamente i vettori e le matrici.

2.1 I vettori

2.1.1 Creare i vettori

Supponiamo di misurare la temperatura ambientale per 10 giorni consecutivi. Possiamo rappresentare le nostre misurazioni attraverso un vettore, in modo tale che vi sia una corrispondenza tra il giorno in cui abbiamo effettuato la misurazione e la posizione del dato all'interno del vettore. Il terzo elemento del vettore ci fornirà la temperatura misurata il terzo giorno.
Per creare un vettore in Matlab basta digitarne gli elementi, racchiusi da una coppia di parentesi **quadre**. Separando i valori con uno spazio o una virgola (scelta consigliata), viene generato un vettore riga:

```
>> temp_riga = [14, 15, 12, 13, 13, 16, 19, 14, 15, 17]

temp =
     14   15   12   13   13   16   19   14   15   17
```

Per creare un vettore colonna, gli elementi devono essere separati da un punto e virgola:

```
>> temp_colonna = [14; 15; 12; 13; 13; 16; 19; 14; 15; 17]

temp =

    14
    15
    12
    13
    13
    16
    19
    14
    15
    17
```

Alternativamente, possiamo digitare una parentesi quadra aperta, immettere il primo valore, premere invio e così via fino all'ultimo elemento seguito da una parentesi quadra chiusa.
Se vogliamo trasformare un vettore riga nel vettore colonna corrispondente dobbiamo usare il comando di trasposizione (‘):

```
>> temp_riga = temp_colonna';
>> temp_colonna = temp_riga';
```

Recuperare un particolare elemento contenuto in un vettore è semplicissimo, basta comunicare al programma la sua posizione. Per esempio, se volgiamo recuperare la temperatura misurata l'ottavo giorno (per visualizzare o utilizzare il dato in un calcolo) dovremo digitare il nome del vettore seguito dal numero 8 racchiuso tra parentesi **tonde**.

```
>> temp_riga (8)

ans =

14
```

Se volgiamo calcolare la somma tra le temperature misurate il terzo ed il sesto giorno, digiteremo:

```
>> temp_riga (3) + temp_riga (6)

ans =

    28
```

Per calcolare la media del nostro insieme di dati basterà chiamare la funzione MEAN di Matlab passandole l'argomento:

```
>> mean (temp_riga)

ans =
```

14.8000

Il fatto di avere registrato tutti i dati in un'unica variabile ci consente di calcolare la media attraverso una singola riga di codice, indipendentemente dal numero totale di dati.
Possiamo creare un nuovo vettore **concatenando** due o più vettori.
Supponiamo di avere generato i seguenti vettori:

```
A = [1, 2, 3]
B = [4, 5, 6, 7]
```

Possiamo generare il vettore C semplicemente unendo A e B:

```
>> C = [A B]

C =

   1    2    3    4    5    6    7
```

2.1.2 Vettori equamente intervallati

2.1.2.1 L'operatore due punti

Se gli elementi di un array sono regolarmente intervallati, si può usare l'operatore due punti (:) per generare il vettore senza dover digitare tutti i valori:

```
A = [i:p:f]
```

dove i indica il valore iniziale della lista, f il valore finale e p il passo o incremento. Il passo può essere negativo, in tal caso i deve essere maggiore di f.
Per esempio, se vogliamo scrivere tutti i numeri pari compresi tra 2 e 20, basta digitare:

```
>> A = [2:2:20]

A =

    2    4    6    8   10   12   14   16   18   20
```

Omettendo p, Matlab utilizza di default incrementi unitari. Se il range (f-i) non'è un multiplo di p, l'ultimo elemento del vettore sarà minore di f. Il passo può essere negativo, in tal caso i deve essere maggiore di f. Si noti che le parentesi quadre possono essere omesse.

2.1.2.2 Il comando linspace

È possibile creare vettori linearmente intervallati anche attraverso il comando linspace. In tal caso non è richiesto il passo, ma il numero di elementi (n) da generare. La sintassi è:

```
linspace (i, f, n)
```

Per esempio:

```
>> linspace (2, 8, 20)

ans =

  Columns 1 through 4
```

```
    2.0000    2.3158    2.6316    2.9474

  Columns 5 through 8

    3.2632    3.5789    3.8947    4.2105

  Columns 9 through 12

    4.5263    4.8421    5.1579    5.4737

  Columns 13 through 16

    5.7895    6.1053    6.4211    6.7368

  Columns 17 through 20

    7.0526    7.3684    7.6842    8.0000
```

Omettendo n, Matlab genera automaticamente 100 valori.

2.1.2.3 Il comando logspace

E' possibile generare un vettore logaritmicamente intervallato utilizzando il comando logspace:

```
logspace (i, f, n)
```

dove n è il numero di elementi da generare, compresi tra 10^i e 10^f . Omettendo n, Matlab genera automaticamente 100 valori.

Per esempio, per generare un vettore di 10 elementi logaritmicamente intervallato tra 10 e 100, digitiamo:

```
>> logspace (1,2,10)

ans =

  Columns 1 through 4

   10.0000   12.9155   16.6810   21.5443

  Columns 5 through 8

   27.8256   35.9381   46.4159   59.9484

  Columns 9 through 10

   77.4264  100.0000
```

2.2 Le matrici

Le matrici sono array bidimensionali, formati da almeno due righe e due colonne. La dimensione di una matrice è data dal numero di righe e di colonne; per esempio, una matrice 3x2 è formata da tre righe e due colonne (il primo numero indica **sempre** le righe ed il secondo le colonne) e contiene 6 elementi.

Per generare una matrice si utilizzano le stesse regole viste nel caso dei vettori: è sufficiente digitare tutti i valori, una riga dopo l'altra, separando le righe con l'operatore (;) e i valori di ciascuna riga con l'operatore (,) o con uno spazio.
Per esempio, la seguente istruzione:

```
A= [7, 3, 5; 9, 1, 4]
```

genera una matrice 2x3, formata da due righe e tre colonne:

```
A =

   7   3   5
   9   1   4
```

Supponiamo di misurare la temperatura ambientale per 10 giorni consecutivi in 4 siti ambientali diversi. Possiamo registrare le nostre misurazioni in una matrice 4x10 in cui ogni riga corrisponde al vettore che raccoglie le misurazioni relative ad un particolare sito.
Un particolare elemento all'interno di una matrice è identificato attraverso due indici che rappresentano rispettivamente la riga e la colonna in cui si trova l'elemento. Per esempio, per recuperare la temperatura registrata il secondo giorno (colonne) nel 4 sito (righe) dovremo digitare:

```
>> T (4, 2)
```

Attenzione: il numero della riga viene sempre indicato per primo.

Nel nostro esempio, supponendo di avere precedentemente generato i vettori temp1, temp2, temp3 e temp4, in cui sono state registrate le temperature rilevate nei 4 siti, possiamo generare la matrice T concatenando per riga i vettori:

```
>> T = [temp1; temp2; temp3; temp4];
```

Supponendo di voler aggiungere a T le misurazioni relative ad un quinto sito, registrate nel vettore temp5, dovremo digitate:

```
>> T = [T; temp5];
```

Ecco un esempio della matrice T:

22	16	19	17	15	22	20	17	16	15
15	15	15	12	14	17	21	19	12	21
16	16	13	19	19	21	13	18	16	20
15	17	16	19	20	16	16	17	16	22
12	15	18	13	18	14	18	17	17	17

Per aggiungere un vettore riga ad una matrice, come nel caso precedente, è necessario che la matrice ed il vettore abbiano lo stesso numero di colonne. In altri termini il numero di elementi del vettore deve coincidere con il numero di colonne della matrice. In caso contrario Matlab genera un messaggio di errore:

```
??? Error using ==> vertcat
All rows in the bracketed expression must have the same
number of columns.
```

Supponiamo ora di procedere ad una nuova misurazione delle temperatura nei 5 siti e di registrarne gli esiti in un vettore chiamato t11:

12 10 23 12 14

Per aggiornare la nostra matrice T, dovremo digitare:

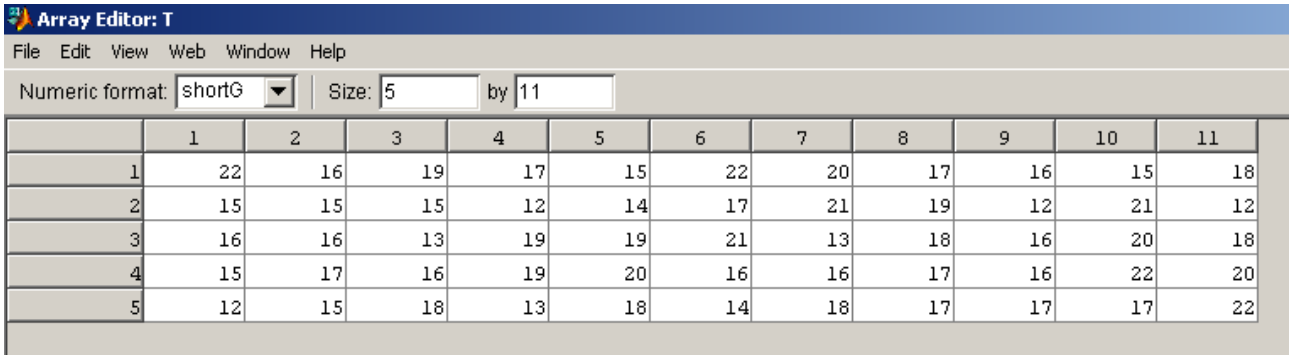
```
>> T=[T, t11']

T =

22  17  12  20  17  19  20  20  21  24  12
10  20  20  15  23  15  14  18  24  14  10
20  16  14  22  22  14  22  16  17  13  23
15  14  18  22  19  15  18  20  23  23  12
22  12  12  18  22  18  15  19  12  21  14
```

Per aggiungere un vettore colonna ad una matrice, come nell'esempio precedente, è necessario che la matrice ed il vettore abbiano lo stesso numero di righe. Pertanto, il numero di elementi del vettore deve coincidere con il numero di righe della matrice. In caso contrario Matlab genera il seguente messaggio di errore:

```
??? Error using ==> horzcat
All matrices on a row in the bracketed expression must have the
 same number of rows.
```

2.3 L'editor degli array

Come abbiamo visto in precedenza, la finestra Workspace permette di visualizzare le variabili caricate nell'area di lavoro di Matlab. Facendo doppio clic sul nome di una variabile, per esempio T, si aprirà l'Array Editor che visualizza i valori della variabile selezionata:



	1	2	3	4	5	6	7	8	9	10	11
1	22	16	19	17	15	22	20	17	16	15	18
2	15	15	15	12	14	17	21	19	12	21	12
3	16	16	13	19	19	21	13	18	16	20	18
4	15	17	16	19	20	16	16	17	16	22	20
5	12	15	18	13	18	14	18	17	17	17	22

Figura 7. L'Array Editor.

L'Array Editor può anche essere aperto digitando:

```
open ('nome_variabile')
```

dalla finestra dei comandi.
L'Array Editor consente di modificare singoli valori facendo clic sulle celle corrispondenti, digitando i nuovi valori e premendo invio. Il suo uso è tuttavia limitato ad array con non più di due dimensioni.

2.4 Array n-dimesionali e indicizzazione

Molto spesso, nelle applicazioni pratiche, i valori registrati in un array corrispondono a variabili specifiche. Nel nostro esempio, nella matrice T sono rappresentate due variabili: il giorno di misurazione ed il sito di rilevamento. Supponendo di ripetere le nostre misurazioni per 6 mesi, da Gennaio a Giugno, potremmo aggiungere una dimensione alla nostra matrice per rappresentare la variabile mese di misurazione. Otterremo in tal caso un array **T** di dimensioni 5x11x6.
Possiamo visualizzare l'array T come un libro suddiviso di 6 pagine. La prima pagina corrisponde al mese di Gennaio, la seconda a Febbraio e così via. Ogni pagina visualizza la matrice in cui sono riportate le temperature registrate nei diversi siti in giorni diversi. In altre parole, un array tridimensionale può essere immaginato come strati successivi di matrici.

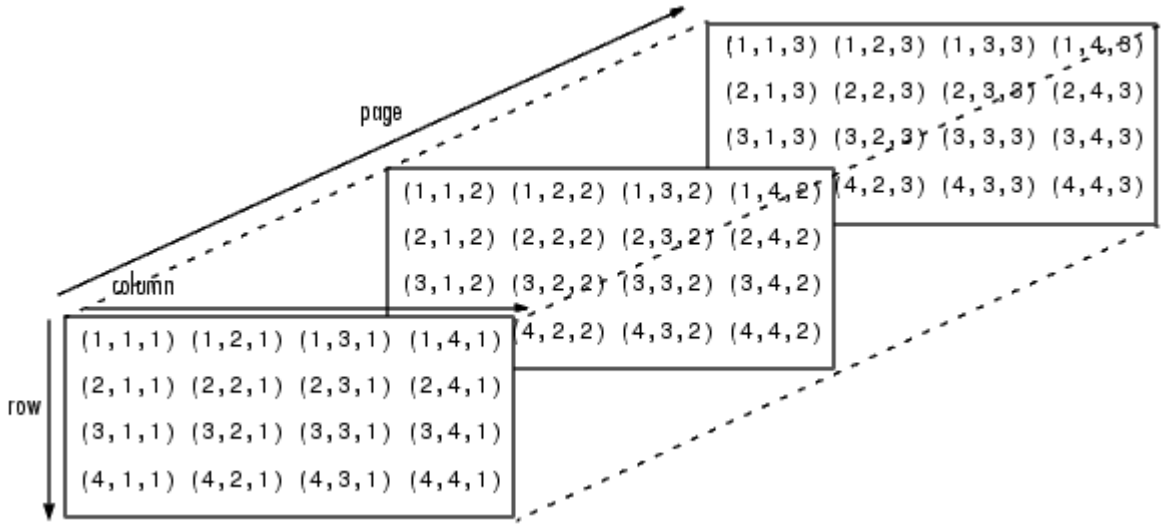


Figura 8. Array tridimensionali.

Gli **indici** sono valori numerici che permettono di identificare gli elementi di un array. In generale avremo bisogno di tanti indici quante sono le dimensioni dell'array. La sintassi prevede di indicare il nome dell'array seguito dagli indici racchiusi da parentesi **tonde** e separati dall'operatore virgola (,):

```
nome_array (indice1, … , indiceN)
```

dove N è il numero di dimensioni dell'array,
Se per esempio, vogliamo modificare la temperatura registrata il decimo giorno (colonne) di Gennaio (pagine) nel sito di rilevamento numero 3 (righe), registrando il valore 12, dovremo digitare:

```
>> T (3, 10, 1) = 12;
```

L'operatore due punti (:) permette di selezionare sottoinsiemi di valori all'interno di un array. Ricordando che l'operatore viene utilizzato per definire intervalli, è sufficiente estendere tale concetto agli indici di un array.
Ecco alcuni esempi:

```
>> t11 (2 : 4)

ans =
```

```
    10    23    12
```

Ricordando che:

```
T =

   22   17   12   20   17   19   20   20   21   24   12
   10   20   20   15   23   15   14   18   24   14   10
   20   16   14   22   22   14   22   16   17   13   23
   15   14   18   22   19   15   18   20   23   23   12
   22   12   12   18   22   18   15   19   12   21   14
```

Il comando:

```
>> T (1 : 3 , 1 : 3)

ans =

   22   17   12
   10   20   20
   20   16   14
```

seleziona la sottomatrice ottenuta prendendo gli elementi appartenenti contemporaneamente alle prime tre righe ed alle prime tre colonne.
Il comando

```
>> T (3 : 5 , 9 : 11)
ans =

   17   13   23
   23   23   12
   12   21   14
```

seleziona la sottomatrice ottenuta prendendo gli elementi appartenenti contemporaneamente alle ultime tre righe ed alle ultime tre colonne.
Il comando:

```
>> T( : , 1 : 2 )

ans =

   22   17
   10   20
   20   16
   15   14
   22   12
```

seleziona la sottomatrice ottenuta prendendo le prime due colonne dell'array. L'operatore (:) da solo indica a Matlab di considerare l'intero range dell'indice corrispondente.

```
>> T(: , :)
```

```
ans =

   22   17   12   20   17   19   20   20   21   24   12
   10   20   20   15   23   15   14   18   24   14   10
   20   16   14   22   22   14   22   16   17   13   23
   15   14   18   22   19   15   18   20   23   23   12
   22   12   12   18   22   18   15   19   12   21   14
```

In questo caso, abbiamo detto al programma di considerare tutta la matrice T (tutte le righe e tutte le colonne).
E' possibile, inoltre, selezionare righe o colonne specifiche non contigue.
Per esempio, il comando:

```
>> T([1 3 5] , :)

ans =

   22   17   12   20   17   19   20   20   21   24   12
   20   16   14   22   22   14   22   16   17   13   23
   22   12   12   18   22   18   15   19   12   21   14
```

seleziona le righe numero 1, 3, 5 della matrice T.

2.5 Cancellare righe e colonne
L'array vuoto si indica in Matlab con il simbolo [ ]. Per cancellare righe o colonne bisogna selezionarle ed assegnare l'array vuoto alla selezione. Per esempio, se abbiamo:

```
Z =

   19   24   15   21
   14   18   16   17
   17   16   13   19
   10   17   18   13
```

possiamo cancellare la prima colonna digitando:

```
>> Z (:,1) = [ ]

Z =

   24   15   21
   18   16   17
   16   13   19
   17   18   13
```

e l'ultima riga digitando:

```
>> Z(4,:)=[ ]

Z =

   24   15   21
```

```
    18    16    17
    16    13    19
```

2.6 Comandi utili per gli array

Matlab implementa alcuni comandi molto utili nell'utilizzo degli array. I più comuni sono:

max (x) restituisce il valore più grande di x se x è un vettore; se x è una matrice restituisce un vettore riga i cui elementi corrispondono ai valori più grandi delle corrispondenti colonne di x

min (x) equivalente a max(x) con la differenza che restituisce i valori più bassi

ndims (x) restituisce il numero di dimensioni di un array

size (x) restituisce un vettore riga con due valori [i j] uguali al numero di righe e colonne di x

length (x) restituisce il numero degli elementi di x

sum (x) restituisce la somma dei valori di x se x è un vettore; se x è una matrice restituisce un vettore riga i cui elementi corrispondono alla somma degli elementi di ciascuna colonna di x

sort (x) restituisce un array della stessa dimensione di x; se x è un vettore, ordina gli elementi in senso crescente; se x è una matrice, ordina ogni singola colonna

2.7 Operazioni con gli array

Nel trattare le operazioni con gli array è utile distinguere tre casi, in base al tipo di operandi:

1. array-scalare e scalare-array
2. vettore-vettore o matrice-matrice
3. matrice-vettore

2.7.1 Moltiplicazione array-scalare

La moltiplicazione di un array A per uno scalare restituisce un nuovo array C i cui elementi sono il prodotto dei corrispondenti elementi di A per lo scalare. Per esempio:

```
>> Z*3

ans =

    57    72    45    63
    42    54    48    51
    51    48    39    57
    30    51    54    39
```

2.7.2 Divisione array-scalare e scalare-array

Analogamente alla moltiplicazione, la divisione array-scalare restituisce un nuovo array i cui elementi sono il quoziente tra i corrispondenti elementi di A ed il valore scalare.
Matlab consente anche la divisione scalare array, attraverso un apposito operatore: **./**. Per esempio, se d=[1 10 100], digitando 1./ si ottiene [1/1 1/10 1/100].

```
>> 1./d

ans =

    1.0000    0.1000    0.0100
```

2.7.3 Elevazione a potenza array-scalare e scalare-array

Analogamente alla moltiplicazione, l'elevazione a potenza array-scalare restituisce un nuovo array i cui elementi sono dati dall'elevazione a potenza dei corrispondenti elementi di A. Tale operazione richiede l'operatore **.^**.

```
G =

    1     2     3

>> G.^3

ans =

    1     8    27
```

Matlab consente anche di elevare uno scalare ad array di potenze. Per esempio:

```
>> 3.^G

ans =

    3     9    27
```

2.7.4 Addizione e sottrazione array-scalare

Analogamente alla moltiplicazione, l'addizione (sottrazione) array-scalare restituisce un nuovo array i cui elementi sono la somma (sottrazione) dei corrispondenti elementi di A ed il valore scalare.

```
>> G+50

ans =

   51    52    53
```

2.7.5 Moltiplicazione tra array elemento-per-elemento

La moltiplicazione tra l'array A e l'array B elemento-per-elemento (.*) restituisce un nuovo array C i cui elementi sono il prodotto dei corrispondenti elementi di A e di B. Dalla definizione deriva che A, B e C hanno la stessa dimensionalità. Sintassi:

```
A.*B
```

Formalmente, C=A.*B implica $c_{ij}=a_{ij}*b_{ij}$.

Esempio Siano:

```
A =

    15    18    10    10
    21    21    16    21
    20    10    14    24
    16    19    23    24
```

e

```
B =

    21    19    16    24
    16    14    21    20
    17    24    14    13
    13    20    16    22
```

La moltiplizione elemento-per-elemento si ottiene digitando:

```
>> C=A.*B

 C =

   315   342   160   240
   336   294   336   420
   340   240   196   312
   208   380   368   528
```

2.7.6 Divisione tra array elemento-per-elemento

È analoga alla moltiplicazione tra array elemento-per-elemento, e si effettua digitando il simbolo **./**. Formalmente, C=A./B implica $c_{ij}=a_{ij}/b_{ij}$.

2.7.7 Addizione e sottrazione di array

Analogamente alla moltiplicazione di array elemento-per-elemento, l'addizione (sottrazione) tra l'array A e l'array B restituisce un nuovo array C i cui elementi sono la somma (sottrazione) dei corrispondenti elementi di A e di B. Dalla definizione deriva che A, B e C hanno la stessa dimensionalità.
Formalmente, C=A±B implica $c_{ij}=a_{ij}\pm b_{ij}$.

2.7.8 Prodotto scalare

Il prodotto scalare è un'operazione tra un vettore riga $[a_1, \ldots, a_n]$ ed un vettore colonna $[w_1, \ldots, w_n]$ aventi entrambi n elementi. E' definito come il valore scalare ottenuto dalla somma dei prodotti degli elementi corrispondenti dei due vettori: $a_1 w_1 + \ldots + a_n w_n$.
Per esempio, dato un vettore riga vr = [12, 4, 7] ed un vettore colonna vc = [3, 6, 7], il prodotto scalare sarà:

$$\begin{bmatrix} 12 & 4 & 7 \end{bmatrix} \begin{pmatrix} 3 \\ 6 \\ 7 \end{pmatrix} = 12*3 + 4*6 + 7*7 = 109$$

In Matlab basterà digitare:

```
>> vr*vc

ans =

   109
```

Alternativamente, si può usare la funzione dot:

```
>> dot(vr,vc)
```

```
ans =

   109
```

2.7.9 Moltiplicazione matrice-vettore colonna

È una generalizzazione del prodotto scalare. Richiede che il numero di colonne della matrice coincida con il numero di elementi del vettore colonna. Ogni riga della matrice fornisce il vettore riga su cui viene calcolato il prodotto scalare. Il risultato è un vettore colonna avente tanti elementi quante sono le righe della matrice.

$$\begin{pmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} = \begin{pmatrix} m_{11}*v_1 + m_{12}*v_2 \\ m_{21}*v_1 + m_{22}*v_2 \end{pmatrix}$$

$$\begin{pmatrix} 2 & 7 \\ 6 & -5 \end{pmatrix} \begin{pmatrix} 3 \\ 9 \end{pmatrix} = \begin{pmatrix} 2*3 + 7*9 \\ 6*3 + (-5*9) \end{pmatrix} = \begin{pmatrix} 69 \\ -27 \end{pmatrix}$$

2.7.10 Moltiplicazione matrice-matrice

È un'estensione della moltiplicazione matrice-vettore, con l'unica differenza ogni colonna della seconda matrice fornisce il vettore che sarà proiettato sulle righe della prima matrice per il calcolo del prodotto scalare. La moltiplicazione della matrice A con la matrice B richiede che il numero di colonne di A sia uguale al numero di righe di B e fornisce una matrice C con un numero di righe uguale ad A e di colonne uguale a B. In Matlab si esegue utilizzando l'operatore *****.

Esempio Date le seguenti matrici:

```
A (3x4) =

    0.6288    0.6072    0.5751    0.0272
    0.1338    0.6299    0.4514    0.3127
    0.2071    0.3705    0.0439    0.0129

B  (4x6) =

    0.3840    0.6124    0.1901    0.6315    0.4544    0.6756
    0.6831    0.6085    0.5869    0.7176    0.4418    0.6992
    0.0928    0.0158    0.0576    0.6927    0.3533    0.7275
    0.0353    0.0164    0.3676    0.0841    0.1536    0.4784
```

Calcolare la moltiplicazione AB:

```
>> C=A*B

C =

    0.7106    0.7641    0.5190    1.2335    0.7613    1.2808
    0.5346    0.4775    0.5360    0.8755    0.5466    1.0088
    0.3371    0.3532    0.2641    0.4281    0.2753    0.4371
```

C risulta avere dimensionalità 3x6.

2.8 Matrici speciali

Ecco alcuni dei principali comandi Matlab ideati per creare matrici speciali:

eye (n)	crea una matrice identità nxn
ones (n)	crea una matrice nxn i cui elementi sono uguali a 1
zeros (n)	crea una matrice nxn i cui elementi sono uguali a 0
ones (m,n)	crea una matrice mxn i cui elementi sono uguali a 1
zeros (m,n)	crea una matrice mxn i cui elementi sono uguali a 0

2.9 Funzioni elementari

round	arrotonda all'intero più vicino
fix	arrotonda al più vicino intero verso lo zero
floor	arrotonda per difetto
ceil	arrotonda per eccesso
rem	resto di una divisione intera
abs	valore assoluto

2.10 Tipi di dati strutturati

Le ***strutture*** sono **array MATLAB**® con *campi* (fields) che possono contenere ogni tipo di dati. Per esempio un campo potrebbe contenere una stringa di testo rappresentante un nome, un secondo campo un valore numerico rappresentante una la cifra di un conto da pagare ed un terzo campo una matrice di risultati di test medicali. E' sempre possibile definire un array a partire da una struttura così come si fa per un numero. Si possono costruire array di strutture di dimensioni e forme qualunque purché valide, includendo anche array multidimensionali.
Una semplice struttura può essere costruita assegnando valori ai singoli campi. MATLAB automanticamente crea la struttura. Per esempio:

- paziente.nome = 'Mario Rossi';
- paziente.conto = 127.00;
- paziente.test = [79 75 73; 180 178 177.5; 220 210 205];

Per visualizzare i dati contenuti nella struttura basta scrivere "paziente" e si ottiene:

- nome = 'Mario Rossi';
- conto = 127.00;
- paziente.test = [3x3 double]

Si può considerare paziente come un array contente una struttura contenente tre campi. Per aggiungere un nuovo elemento dell'array possiamo scrivere:

- paziente(2).nome = 'Ann Lane';
- paziente(2).conto= 28.50;
- paziente(2).test = [68 70 68; 118 118 119; 172 170 169];

In questo caso scrivendo il nome della struttura non si visualizzano i dati contenuti ma una semplice ricapitolazione: paziente = 1x2 struct array with fields: nome, conto, test
Un metodo alternativo per creare la struttura è:

```
paziente = struct('nome','Mario Rossi','conto',127.00,…
                  'test',[79 75 73; 180 178 177.5; 220 210 205])
```

a video non avendo inserito il punto e virgola si avrà:

```
paziente =
    nome: 'Mario Rossi'
    conto: 127
test: [3x3 double]
```

per aggiungere un campo si procede nel seguente modo:

```
paziente.via = 'Roma'
```

2.11 I cellarray

Il cellarray è un insieme di contenitori chiamati celle in cui è possibile memorizzare diversi tipi di dati. La figura sottostante rappresenta un array di celle di dimensione 2 per 3. Le celle nella prima riga in possesso di una serie di numeri interi senza segno, un array di stringhe, e una serie di numeri complessi. Seconda fila contiene tre altri tipi di array, di cui l'ultimo è un cellarray di cellarry

<table>
<tr>
<td>Cell 1,1
<table><tr><td>3</td><td>4</td><td>2</td></tr><tr><td>9</td><td>7</td><td>6</td></tr><tr><td>8</td><td>5</td><td>1</td></tr></table></td>
<td>Cell 1,2
<table><tr><td>‘Mario Rossi’</td></tr><tr><td>‘11/8/1902’</td></tr><tr><td>‘via Bove’</td></tr><tr><td>‘ob1’</td></tr></table></td>
<td>Cell 1,3
<table><tr><td>3+3i</td><td>.8+1.3i</td></tr><tr><td>5+.6i</td><td>.4+.3i</td></tr></table></td>
</tr>
<tr>
<td>Cell 2,1
<table><tr><td>2.43</td><td>1.21</td><td>6</td></tr><tr><td>3.4</td></tr></table></td>
<td>Cell 2,2
<table><tr><td>2.3</td><td>4</td><td>3.6</td></tr><tr><td>5</td><td>10.2</td><td>8</td></tr></table></td>
<td>Cell 2,3
<table><tr><td><table><tr><td>3</td><td>4</td><td>2</td></tr><tr><td>9</td><td>7</td><td>6</td></tr></table></td><td>‘text’</td></tr><tr><td>3.1+2i</td><td><table><tr><td>2.3</td><td>4</td></tr><tr><td>5</td><td>10.2</td></tr></table></td></tr></table></td>
</tr>
</table>

Ogni cella di un cellarray contiene un tipo di matrice MATLAB. I dati contenuti nella matrice possono appartenere a una o più classi di dati del MATLAB, anche ad una classe definita dall'utente, e possono avere una qualsiasi dimensione valida per un array, questo include un array scalare 1-by-1, o un array con una o più dimensioni o anche un un array con dimensione nulla (array vuoto). Una altra cella della stessa matrice può contenere dati di tipo completamente differente, e può anche avere dimensioni diverse rispetto dalle altre. La capacità di memorizzare array di classi miste per tipo e dimensione è la caratteristica più significativa di un cellarray. Un altro uso comune dei cellarray è quello di memorizzare stringhe di caratteri di lunghezza diversa. Il cellarray che viene utilizzato a questo scopo è definito col nome di cellarray di stringhe. Come tutte le matrici MATLAB, il cellarray deve essere di forma rettangolare. Cioè, la lunghezza di tutte le righe deve essere la stessa, la lunghezza di tutte le colonne della stessa, e così via per ogni dimensione della matrice. Per molti aspetti, i cellarray sono molto simili a array di strutture. Nella tabella seguente si riportano gli operatori utili per creare, concatenare e indirizzare una cella in un cellarray.

Operatore	Sintassi	Descrizione
creazione	C = {A B D E}	crea un cellarray formato da i dati A B D E anche di quattro differenti tipi
concatenazione	C3 = {C1 C2}	concatena i cellarray C1 and C2 in un cellarray C3 a due elementi in modo tale che C3{1} = C1 and C3{2} = C2.
	C3 = [C1 C2]	concatena il contenuto dei cellarrays C1 and C2, assumendo che le dimensioni di questi array sono compatibili.
Indirizzamento	X = C(s)	restituisce un cellarray X contenente la sola cella corrispondente all’indice s estratta dal cellarray C
	X = C{s}	restituisce i dati contenuti nella cella del cellarray corrispondente all’indice s
	X = C{s}(t)	Fa riferimento ad uno o più elementi di un array memorizzato in una cella del cellarry. L’indice s seleziona la cella, l’indice t seleziona l’elemento dell’array.

Problemi

21.) Creare il vettore i cui elementi sono linearmente intervallati con incrementi di 0.4 tra 2 e 14. Utilizzare due metodi diversi.

2.2) Creare il vettore che ha 100 elementi linearmente intervallati tra 5 e 28. Utilizzare due metodi diversi.

2.3) Creare il vettore che ha 50 elementi linearmente intervallati tra -2 e 5. Utilizzare due metodi diversi.

2.4) Creare il vettore che ha 50 elementi logaritmicamente intervallati tra 10 e 1000.

2.5) Creare il vettore che ha 20 elementi logaritmicamente intervallati tra 100 e 10000.

2.6) Definito il vettore x=[-3, 4, 2, 1, 0, 2, 3, 5, 10] calcolare:

1. length (x)
2. size (x)
3. media di x
4. assegnare agli elementi con indici pari il valore 7, digitando un unico comando (suggerimento: usare l'operaotore :)

2.7) Definito il vettore x=[6, 7, 5, 1, 0, 2, 13, 5, 12], costruire a partire da esso la matrice:

```
  15   7  15   1  15   2  15   5  15  15   7  15   1  15   2  15   5  15  15   7  15   1  15
   2  15   5  15
  15   8  15   2  15   3  15   6  15  16   7  16   1  16   2  16   5  16  15   8  15   2  15
   3  15   6  15
```

Soluzione

```
>> x(1:2:9)=15
x =
   15    7   15    1   15    2   15    5   15
>> x=[x x x;x x x]
x =
  Columns 1 through 13
   15    7   15    1   15    2   15    5   15   15    7   15    1
   15    7   15    1   15    2   15    5   15   15    7   15    1
  Columns 14 through 26
   15    2   15    5   15   15    7   15    1   15    2   15    5
   15    2   15    5   15   15    7   15    1   15    2   15    5
  Column 27
   15
   15
>> x(2,2:2:26)= x(1,2:2:26)+1
x =
  Columns 1 through 13
   15    7   15    1   15    2   15    5   15   15    7   15    1
   15    8   15    2   15    3   15    6   15   16    7   16    1
  Columns 14 through 26
   15    2   15    5   15   15    7   15    1   15    2   15    5
   16    2   16    5   16   15    8   15    2   15    3   15    6
  Column 27
   15
   15
```

2.8) Generare il vettore x=[2, 3, 4, 5, 6, 7, 12], si costruisca il vettore y=[2, 3, 4, 5, 6, 7, 12, 12, 7, 6, 5, 4, 3, 2].

2.9) A partire dai vettori:

a=[0, 3, 6, 9]
b=[45, 40, 35, 30]

costruire i vettori:

```
c=[0, 45, 3,  40, 6, 35, 9, 30]
d=[0, 90, 6,  80, 12, 70, 18, 60]
e=[0, 90, 3, 80, 6, 70, 9, 60]

>> c(2:2:8)=b
c =
    0    45     0    40     0    35     0    30
>> c (1:2:7)=a
c =
    0    45     3    40     6    35     9    30
>> d=c*2
d =
    0    90     6    80    12    70    18    60
>> e=d
e =
    0    90     6    80    12    70    18    60
>> e(1:2:7)=d(1:2:7)/2
e =
    0    90     3    80     6    70     9    60
```

2.10) Data la seguente matrice:
A=

$$\begin{pmatrix} 1 & 0 & 6 & -3 \\ -1 & 2 & 0 & 2 \\ 0 & 3 & -1 & -2 \\ -6 & 0 & 4 & 1 \end{pmatrix}$$

1. si valuti A(: , 2)
2. si valuti le media della seconda colonna e della terza riga
3. si generi la matrice composta dagli elementi appartenenti alla seconda e quarta riga ed alla terza e quarta colonna
4. si generi la matrice composta dagli elementi appartenenti alla prima e terza riga ed alla seconda e quarta colonna (Utilizzare i vettori negli indici di riga e colonna: X=A([1 3],[2 4]))

2.11) Data la matrice:
A=

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

si estragga la sottomatrice di A composta dagli elementi A_{ij}, tali che i=2, 3 e j=2,3.

2.12) Tabellare la funzione log x nell'intervallo [1,5] con passo 0.1. Con il termine "tabellare" si intende creare una tabella, tale che la prima colonna rappresenti la variabile x e la seconda la variabile y.

2.14) Tabellare la funzione:
$y=e^{3x} \sin 5\pi x$
Il π in Matlab si indica con pi.

2.15) Creare un array x con 8 valori compresi tra 2 e 16. Creare una matrice A tale che:

1. la prima riga sia contenga i valori 3x
2. la seconda riga contenga i valori $(3x)^2$
3. la terza riga contenga i valori x-8
4. Scambiare la prima e la terza riga di A.

2.16) Creare la matrice C tale che:

1. la prima riga corrisponda alla seconda colonna di A
2. la seconda riga corrisponda alla terza colonna di A
3. la terza riga corrisponda alla prima colonna di A

2.17) Date le seguenti matrici:

A =
-7 16
4 9

B =
6 -5
12 -2

C =
-3 -9
6 2

Usare Matlab per:

1. Calcolare A+B+C
2. Calcolare A-B+C
3. Verificare la proprietà associativa:
   (A+B)+C=A+(B+C)
4. Calcolare il prodotto elemento-per-elemento di A per B
5. Elevare al cubo gli elementi di B
6. Elevare 3 alla prima riga di C

2.18) La seguente tabella riporta il consumo medio, i km percorsi e l'indice di affidabilità di 5 diverse automobili.

	1	2	3	4	5
Consumo km/l	11,2	8,8	9,1	12	16,4
Km percorsi	240	550	302	667	980
Affidabilità	0.921	0.454	0.980	0.777	0.602

Usare Matlab per:

1. Calcolare i litri di benzina consumati da ogni automobile
2. Calcolare il consumo medio delle 5 automobili prese in esame
3. Calcolare la percorrenza media delle 5 automobili prese in esame
4. Usare le funzioni corrette per determinare l'auto più affidabile e quella che consuma di meno

2.19) Data la matrice quadrata:

A =

$$\begin{pmatrix} 12 & 34 & 4 \\ 77 & 21 & 1 \\ 3 & 5 & 9 \end{pmatrix}$$

Dimostrare che IA=AI=A, dove I indica la matrice identità.

2.20) Date le seguenti matrici:
A =

$$\begin{pmatrix} 12 & 34 & 4 \\ 77 & 21 & 1 \\ 3 & 5 & 9 \end{pmatrix}$$

B =

$$\begin{pmatrix} 6 & -5 \\ 12 & -2 \\ 3 & 7 \end{pmatrix}$$

Calcolare:

1. Il prodotto A*B
2. la somma di tutti gli elementi di A*B
3. il prodotto scalare tra la seconda colonna di A e la prima riga di B
4. la media del prodotto scalare tra la prima colonna di A e di B, ed il prodotto scalare della seconda colonna di A e di B

2.21) Data la matrice:
A=

$$\begin{pmatrix} 3 & 7 & -4 & 12 \\ -5 & 9 & 10 & 2 \\ 6 & 13 & 8 & 11 \\ 15 & 5 & 4 & 1 \end{pmatrix}$$

1. Trovare i valori minimi e massimi di ogni colonna.
2. Trovare i valori minimi e massimi di ogni riga.

2.22) Data la matrice:

A=

$$\begin{pmatrix} 3 & 58 & -4 & 3 \\ -5 & 42 & 80 & 2 \\ 6 & -19 & 9 & 55 \\ 17 & -35 & 44 & 1 \end{pmatrix}$$

1. Ordinare gli elementi di ogni colonna e assegnare il risultato all'array B
2. Ordinare gli elementi di ogni riga e assegnare il risultato all'array C
3. Sommare gli elementi di ogni colonna e assegnare il risultato all'array E
4. Sommare gli elementi di ogni riga e assegnare il risultato all'array D

2.23) Dati i seguenti array:
A=

$$\begin{pmatrix} 1 & 4 & 2 \\ 2 & 4 & 100 \\ 7 & 9 & 7 \\ 3 & \pi & 42 \end{pmatrix}$$

B= ln (A)

1. Calcolare la somma degli elementi della seconda riga di B
2. Calcolare il prodotto scalare tra la seconda colonna di B e la prima colonna di A
3. Calcolare il valore massimo del vettore risultante dalla moltiplicazione elemento-per-elemento dei primi tre elementi della seconda colonna di A e la prima riga di B
4. Calcolare la somma degli elementi della terza riga di A dopo averli divisi elemento-per-elemento per, per i primi tre elementi della terza colonna di B

2.24) Creare un array tridimensionale D le cui tre pagine siano le seguenti matrici:

A=

$$\begin{pmatrix} 3 & -2 & 1 \\ 6 & 8 & 5 \\ 7 & 9 & 10 \end{pmatrix}$$

B=

$$\begin{pmatrix} 6 & 9 & -4 \\ 7 & 5 & 3 \\ -8 & 2 & 1 \end{pmatrix}$$

C=

$$\begin{pmatrix} -7 & -5 & 2 \\ 10 & 6 & 1 \\ 3 & -9 & 8 \end{pmatrix}$$

Trovare l'elemento massimo in ogni strato di D e l'elemento massimo di D.

2.25) creare una struttura che abbia i seguenti campi:

- Nome
- Cognome
- Data di nascita
- Paga

2.26) a) creare un cellarray D che contenga i seguenti elementi:
A = [1 2 3]; B = 'nome file'; C = [3.2+3i 2.4+6i];
b) estrarre la prima cella del cellarray
c) estrarre il primo valore dell'array contenuto nella terza cella

I file in Matlab

Matlab utilizza tre tipi di file: gli M-file, i MAT-file ed i file dati. Gli M-file hanno estensione m e sono utilizzati per scrivere sequenze di comandi (script e funzioni); essendo file ASCII (American Standard Code for Information Interchange) possono essere scritti utilizzando qualsiasi editor di testi. Matlab, tuttavia, prevede un editor molto utile che include funzioni di controllo del funzionamento dinamico (debug).
I MAT-file, estensione mat, permettono di salvare i nomi ed i valori delle variabili generate durante una sessione di lavoro. Sono file binari e pertanto possono essere letti soltanto dal software che li ha creati.
Infine, Matlab è in grado di utilizzare file dati ASCII creati da altri tipi di programma.

3.1 I file script e le funzioni

I file script

Fino ad ora abbiamo utilizzato Matlab in modalità interattiva, digitando le istruzioni direttamente nel finestra dei comandi. Questa modalità, tuttavia, è utile solo in casi limitati, quando i problemi da risolvere sono molto semplici.
Nelle applicazioni pratiche è molto più conveniente scrivere e salvare i propri programmi contenenti i comandi di Matlab in un file tipo M (M-file). Eseguire un M-file corrisponde a digitare in sequenza tutti i comandi in esso contenuti nella finestra dei comandi. Per eseguire un M-file basta digitarne il nome dopo il prompt di Matlab.
Gli M-file sono semplici file di testo che contengono sequenze di comandi che l'utente intende far eseguire a Matlab. Esistono due categorie di M-flie: i file script ed i file funzione. Inizialmente ci limiteremo a considerare i file script. Gli script sono il tipo più semplice di file di programma perché non hanno argomenti di input o di output. Sono utili per automatizzare una serie di comandi MATLAB, come calcoli che si deve eseguire ripetutamente dalla riga di comando.
La proprietà principale di tali file è che i valori delle variabili create durante la loro esecuzione sono caricate nella sessione di lavoro e pertanto rimangono in memoria fino a quando non vengono esplicitamente cancellate attraverso un comando. Tali variabili sono definite "globali". Gli script condividono "workspace" con la sessione interattiva di MATLAB e con altri script. Tutte le variabili che creano gli script rimangono nell'area di lavoro dopo lo script termina in modo da poterli utilizzare per ulteriori calcoli. L'esecuzione di uno script può inavvertitamente sovrascrivere i dati memorizzati nel workspace.

Le funzioni

La differenza principale tra uno script ed una funzione è che una funzione accetta dati in ingresso restituisce dati in uscita alla funzione o allo script chiamante, mentre gli script non lo fanno. È possibile definire funzioni di MATLAB in un file la cui prima riga contiene la parola chiave "*function"*. Non è possibile definire una funzione all'interno di un file di script o dalla riga di comando di MATLAB. Le funzioni iniziano sempre con una linea di definizione di funzione e terminano o alla fine del file o in corrispondenza della parola chiave "*end*" (opzionale in alcune versioni ma sempre utile per la chiarezza del codice) o in presenza di una linea contenente una nuova definizione di funzione. L'utilizzo della parola chiave "*end*" per segnare la fine di una definizione di funzione è necessaria quando la funzione in corso di definizione contiene una o più funzioni nidificate.
Le funzioni operano su variabili contenute all'interno di uno specifico spazio di lavoro ("function workspace"). Questa area di lavoro è separata dall'area di lavoro di base che è l'area di lavoro a cui si accede al prompt dei comandi MATLAB e negli script.

Tipi di funzioni

MATLAB fornisce differenti tipi di funzioni. La *funzione primaria* è la prima funzione in un file di programma e contiene tipicamente il programma principale. Le *sottofunzioni* che normalmente sono subroutine alla funzione principale e che permettono di definire funzioni multiple in un singolo file. Le *funzioni nidificate* sono funzioni definite all'interno di un'altra funzione. Essi possono contribuire a migliorare la leggibilità del programma e anche dare un accesso più flessibile per le variabili del file. Le *funzioni anonime* che forniscono un modo rapido di creare una funzione da una qualsiasi espressione MATLAB. È possibile comporre funzioni anonime o all'interno di un'altra funzione o al prompt dei comandi MATLAB. Le *funzioni di overload* utili quando è necessario creare una funzione che risponde in modo diverso a tipi di ingresso diversi. Sono simili alle funzioni polimorfe in qualsiasi linguaggio orientato agli oggetti. Le *funzioni private* che permettono di limitare l'accesso alla funzione. È possibile chiamare solo da una funzione che è memorizzata nella cartella a livello superiore. Nella documentazione MATLAB si usa poi il termine funzione di funzioni "function functions". Con questo termine non si vuole indicare un altro tipo di funzione bensì ci si riferisce alle funzioni che accettano un'altra funzione come argomento di input. È possibile passare una funzione ad un'altra funzione utilizzando un "handle".

3.1.1 Creare un file script o una funzione

Per creare un nuovo file script è sufficiente cliccare sulla prima icona a sinistra della barra dei comandi o selezionare l'opzione New/M-file del menu File:

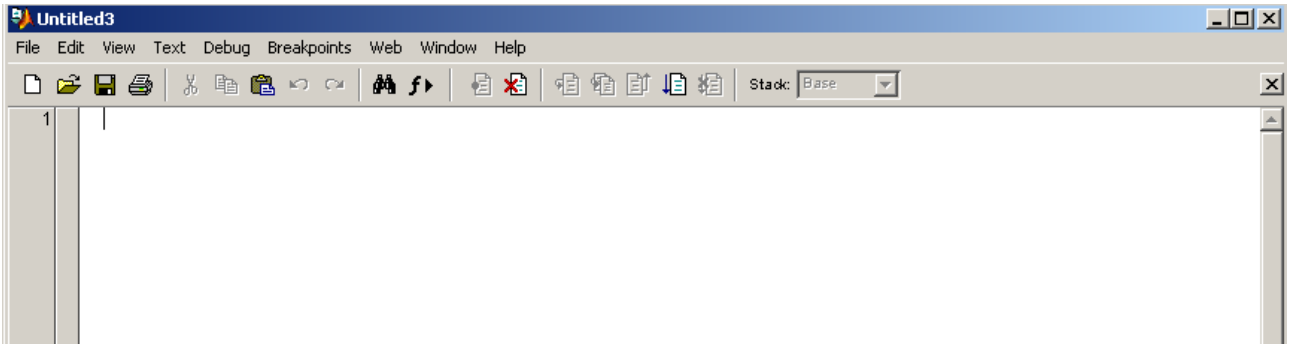


Figura 9. La finestra dell'Editor/Debugger.

Comparirà una nuova finestra di editing dell'utlity Editor\Debugger di Matlab, il cui utilizzo è molto simile a quello di qualsiasi editor di testi.
Il codice seguente permette di creare un file script per calcolare il seno della radice quadrata di un numero:

```
%       esempio1.m
%       calcola il seno della radice quadrata
%       visualizza il risultato
x = sqrt (6);
y = sin (x)
```

Il simbolo % (percentuale) indica a Matlab di non eseguire la riga di comando e può essere usato per inserire un commento nel programma. È buona norma indicare nella prima riga di un file script il suo nome con l'estensione (.m) e successivamente una spiegazione sintetica del suo scopo.
Dopo aver digitato il codice, bisogna salvare il file cliccando sull'icona o selezionando l'opzione Save del menu File. Il nome del file deve rispettare le regole previste per le variabili di Matlab. Il nome del file, inoltre, deve essere diverso da quello delle variabili, altrimenti Matlab visualizza automaticamente il valore della variabile digitata senza eseguire il file script.

A questo punto possiamo eseguire il programma digitando il suo nome nella finestra dei comandi:

```
>> esempio1
```

Matlab visualizzerà il valore della variabile y, poiché la riga di comando corrispondente non termina con il simbolo (;):

```
y =
  0.6382
```

3.1.2 Utilizzo del debugger

Il debugger di Matlab consente di ricercare e correggere gli eventuali errori (bug) presenti in un M-file. Gli errori riconosciuti dal debugger sono di due tipi:

1. Errori di sintassi: Matlab visualizza un messaggio nella finestra dei comandi che descrive l'errore e la sua posizione.
2. Errori di runtime: sono dovuti a procedure di calcolo sbagliate. Tipicamente si verificano per determinati valori delle variabili di input.

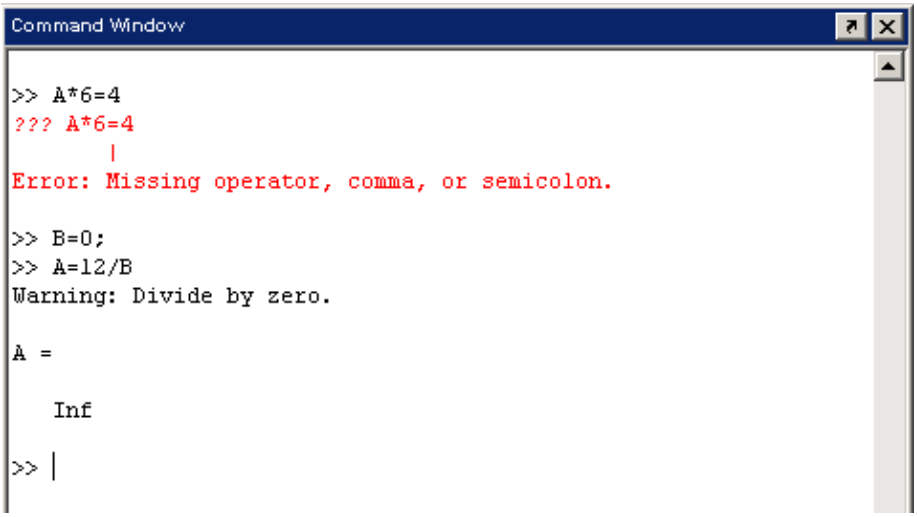


Figura 10. Errore di runtime.

3.1.3 Il menu Text

Permette di:

a. inserire (Comment) o eliminare (Uncomment) commenti;
b. aumentare (Increase Indent) o diminuire (Decrease Indent) i rientri (indentazione);
c. evidenziare (Bilance Delimiters) il codice delimitato dalle parentesi (delimitatori bilanciati);
d. impostare la funzione di rientro automatico (Smart Indent) delle righe di codice contenute all'interno di un'istruzione condizionale;
e. visualizzare nella finestra dei comandi i valori delle variabili selezionate (Evaluate Selection).

Suggerimento: il metodo più veloce per visualizzare il valore di una variabile è quello di posizionare il puntatore del mouse sopra il suo nome; il valore corrispondente comparirà all'interno di una piccola finestra chiamata Datatips.

3.1.4 Il menu Breakpoints

Un breakpoint in Matlab è un marcatore, visualizzato come un cerchietto rosso, in corrispondenza del quale viene interrotta temporaneamente l'esecuzione del programma. Tale interruzione permette all'utente di esaminare eventuali errori di sintassi o di runtime e di valutare i valori delle variabili correnti. Per impostare un breakpoint basta posizionare il cursore in una riga di testo e fare clic sull'icona dei breakpoint  .
Il menu prevede opzioni che permettono di:

1. impostare (Set Breakpoint) o eliminare un breakpoint (Clear Breakpoint);
2. eliminare tutti i breakpoint (Clear All Breakpoint);
3. interrompere il programma quando viene generato un messaggio di errore (Stop If Error), di runtime (Stop If Warning), valori indefiniti (Stop If NaN), valori infiniti (Stop If Inf).

3.1.5 Il menu Debug

Dopo aver impostato un breakpoint è possibile utilizzare le opzioni Step, Step In e Step Out per eseguire il codice interrompendolo in corrispondenza del breakpoint impostato.

Step	il codice viene eseguito un passaggio alla volta
Step In	il codice viene eseguito fino alla prima riga eseguibile della funzione che viene chiamata (utenti esperti)
Step Out	esegue il codice della funzione chiamata (utenti esperti)
Run	permette di eseguire interamente il codice del M-file fino al primo breakpoint impostato
Go Unitl Cursor	imposta un breakpoint temporaneo in corrispondenza della riga in cui si trova il cursore
Exit Debug Mode	permette di uscire dalla modalità di debugging e tornare alla modalità di editing normale

Esercizio Modificare il file *esempio1.m* aggiungendo una variabile chiamata input_sqrt, che sarà utilizzata come argomento della funzione sqrt, assegnandole il valore 8. Impostare un breakpoint sulla riga chiamante la funzione sin. Effettuare il debug attraverso l'opzione Run e verificare che il valore della variabile x sia uguale 2.8284. Terminare l'esecuzione del codice. Verificare che il valore di y sia uguale a 0.3081.
Aggiungere una variabile chiamata input assegnandole il valore 12. Assegnare il valore 5 alla variabile input_sqrt. Aggiungere la variabile input_sin che sarà utilizzata come argomento della funzione sin assegnandole il risultato dell'operazione: x + input. Salvare il file come *empio1b.m*. Impostare un breakpoint sulla riga chiamante la funzione sin. Effettuare il debug attraverso l'opzione Step e verificare che il valore della variabile x sia uguale 2.2361. Terminare l'esecuzione del codice. Verificare che il valore della variabile y sia uguale 0.9951.

3.2 I MAT-file

3.2.1 Salvare e caricare le variabili

Il comando save permette di salvare le variabili registrate nell'area di lavoro in un file binario MAT-file). Per salvare tutte le variabili in nome_file.mat, digitate

```
save nome_file.mat
```

Per caricare successivamente le variabili salvate digitate

```
load nome_file
```

Se volete salvare soltanto alcune variabili (per esempio nome_var1 nome_var2) digitate

```
save nome_file.mat nome_var1 nome_var2
```

Per salvare in formato ASCII in singola precisione digitare save nome_file.mat –ASCII; in doppia precisione save nome_file double.

3.3 Importare i dati

3.3.1 Importare i file ASCII

Quando i dati da elaborare vengono generati da applicazioni esterne a Matlab, per esempio da un'apparecchiatura da laboratorio, bisogna importare i dati. La maggior parte delle applicazioni supportano il formato ASCII. Per caricare questi dati digitare

```
load nome_file.estensione
```

I dati saranno caricati in una variabile che ha lo stesso nome del file (senza estensione).

3.3.2 Importare i file di Excel

Per caricare i dati di una cartella Excel nella matrice A, digitare

```
A = xlsread ('nome_file')
```

Il comando

```
[A, B]= xlsread ('nome_file')
```

importa i dati numerici nell'array A ed i testi nell'array B.

3.3.3 L'Import Wizard di Matlab

Matlab include un'applicazione chiamata Import Wizard per facilitare l'importazione dei dati, ASCII o binari. L'Import Wizard è accessibile selezionando Import Data dal menu File del desktop di Matlab. Apparirà un finestra che permette di selezionare il file che contiene i dati da importare.
L'Import Wizard visualizzerà un'anteprima dei dati contenuto nel file. Dopo aver impostato il separatore di colonna appropriato (di default è selezionato lo spazio) premere Next. Apparirà una finestra di dialogo che consente di selezionare le variabili da importare (data, textdata, colheaders). Premere Finish per terminare il processo di importazione.

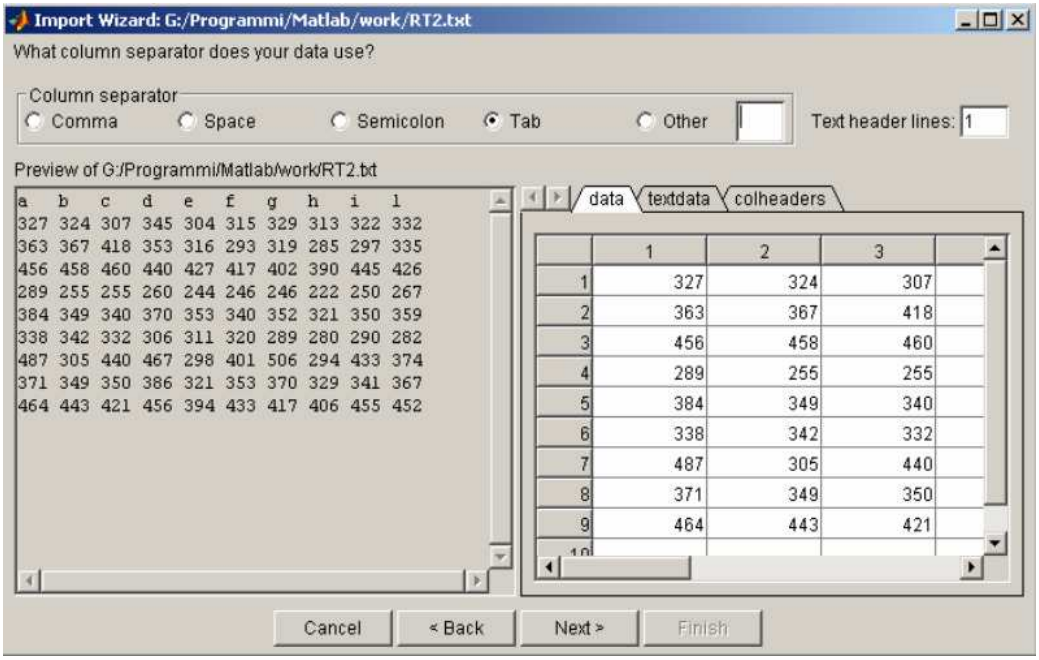


Figura 11. L'Import Wizard.

3.3.4 Le funzioni di import del Matlab

Le funzioni di import dei dati ad alto livello permettono di leggere file dati standard (vedi tabella 2). Se i dati da importare sono contenuti in file di formato non specificato nella tabella 2 è necessario utilizzare le funzioni di I/O di livello basso per importare i dati.

Tabella 2: le funzioni di import dei dati

Formato Dati	Estensione del File	Funzione
File audio	.au .wav	wavread
Audio-video Interleaved (AVI)	.avi	aviread
Common Data Format (HDF)	.cdf	cdfread
Image files	.bmp .cur .gif .hdf .ico .jpg (.jpeg) .pbm .pcx .pgm .png .pnm .ppm .ras .tif (.tiff) .xwd	Imread
MATLAB proprietary format (MAT-files)	.mat	load
Spreadsheets (Excel or Lotus 123)	.xls .wk1	xlsread

3.4 Esportare i file dati in formato ASCII

Matlab prevede due opzioni per esportare un array come file ASCII delimitato: il comando save nome_file –ASCII, oppure la funzione dlmwrite. Quest'ultima è da preferirsi, poiché permette di specificare il delimitatore e di esportare sottoinsiemi di array.
Per esportare la matrice T con il comando save digitate:

```
save nome_file T –ASCII
```

Il comando save utilizza lo spazio come delimitatore di default; per usare il carattere di tabulazione specificare –tab.
Per usare la funzione dlmwrite specificando il punto e virgola come delimitatore, digitate:

```
dlmwrite ('nome_file.out', T, ';')
```

La funzione dlmwrite utilizza la virgola come delimitatore di default, per impostare lo spazio digitare (' ').

3.5 Importare ed esportare i dati in formato binario non standard.

MATLAB include un insieme di funzioni di I/O di basso livello basate sullo standard C.
Per leggere o scrivere un file è necessario seguire il seguente schema:
Aprire il file utilizzando la funzione *fopen*. Tale funzione restituisce un identificatore di file da usare con tutte le altre funzioni.
operare sul file:
leggere dati binari con la funzione *fread*
scrivere dati binari con la funzione *fwrite*
leggere stringhe di testo linea per linea utilizzando le funzioni *fgets/fgetl*
leggere dati formattati ASCII usando la funzione *fscanf*
scrivere dati formattati ASCII usando la funzione *fprintf*
chiudere il file usando la funzione *fclose*

Ad esempio, per leggere 1000 interi a 16 bit memorizzati in formato binario si utilizzano le seguenti istruzioni:

```
fid = fopen('nome file');
[dati, contatore] = fread(fid, 1000, 'int16')
fclose
```

i dati sono memorizzati nel vettore **dati** e il numero di interi effettivamente letto è contenuto in contatore.

Problemi

3.1) Creare uno script per simulare un modello computazionale in grado di computare la funzione logica AND.
Suggerimento: utilizzare 2 unità di input, una di bias ed una di output con attivazione bipolare (-1, 1). La funzione AND richiede una risposta positiva quando le unità di input sono attive e risposta opposta in tutti gli altri casi.

3.2) Scrivere un file script per calcolare l'ipotenusa di un triangolo rettangolo i cui cateti misurano 12 cm e 16 cm. Il file deve avere due variabili di input ed una variabile di output. Inserire un commento che illustri lo scopo del codice. Salvare il file come teorema_pitagora.m.

3.3) Importare i dati dal file RT1.xls, generato dal programma Excel. Calcolare la somma di tutti i dati e cancellare i dai dall'area di lavoro.

3.4) Importare i dati dal file RT2.txt, generato dal programma E-Prime e calcolare la media e la deviazione standard di ogni colonna.

3.5) Creare un array RT2b i cui elementi sono dati dalla radice quadrata dei corrispondenti elementi di RT2. Esportare la variabile RT2b in un file ASCII utilizzando il punto e virgola come delimitatore.

3.6) Scrivere un file binario per memorizzare il vettore di interi [1:1000].

3.7) Leggere il file scritto per l'esercizio 3.6.

Input e output

Matlab dispone di vari strumenti che permettono di ottenere l'input dall'utente e di formattare l'output.

4.1 Il comando disp

Permette di visualizzare sullo schermo il contenuto di una variabile calcolata attraverso uno script. La sintassi è:

```
disp (nome_variabile)
```

Possiamo utilizzare il comando disp anche per visualizzare testi. In questo caso la sintassi prevede di racchiudere il testo tra due apici:

```
disp ('testo da visualizzare')
```

Per esempio, supponiamo che il nostro file script calcoli la media di un determinato insieme di dati e la registri nella variabile MEDIA. Digitando

```
>> disp ('La media dei dati è:'), disp (MEDIA)
```

nell'ipotesi che MEDIA sia uguale a 23, verrà visualizzato sullo schermo:

```
>> La media dei dati è:
    23
```

4.2 Comandi di formattazione

Il comando format permette di modificare l'aspetto dei dati sullo schermo. Per default Matlab utilizza il formato short che visualizza quattro cifre decimali.

Format short	4 cifre decimali
Format long	16 cifre decimali
Format short e	4 cifre decimali più l'esponente
Format long e	15 cifre decimali più l'esponente
Format bank	2 cifre decimali
Format +	positivo, negativo o zero

4.3 Input esterno

4.3.1 Il comando input

Permette di caricare dall'esterno il valore di una variabile. Il comando input visualizza un testo, definito dal programmatore, attende che l'utente digiti un valore e registra l'input nella variabile specificata dal programmatore. La sintassi è:

```
nome_variabile = input ('testo da visualizzare')
```

Matlab visualizza sullo schermo il testo specificato e attende che venga digitato un valore. Se viene digitato 16, tale valore verrà registrato in nome_variabile.

4.3.2 Il comando menu

Si utilizza per vincolare l'input esterno ad un insieme di opzioni. Il comando menu genera un menu di opzioni per l'input esterno:

```
g = menu ('testo' 'opzione1' 'opzione2' 'opzione3')
```

Il valore di g dipende dalla scelta dell'utente: 1 se viene scelta l'opzione 1 e via dicendo.

Elementi di programmazione in Matlab

5.1 Operatori relazionali

Permettono di confrontare variabili e array. Il risultato del confronto può essere 0 (falso) o 1 (vero).
Quando si confrontano due array, vengono confrontati i singoli elementi di un array con i corrispondenti elementi dell'altro. Nel caso di confronti tra uno scalare ed un array, il confronto viene eseguito tra lo scalare e tutti gli elementi dell'array.
Un'istruzione relazionale (confronto) può comparire a destra dell'operatore di assegnazione (=). In tal caso il risultato del confronto viene assegnato alla variabile indicata a sinistra dell'operatore di assegnazione. Tutti gli operatori relazionali hanno lo stesso livello di precedenza; un'istruzione relazionale viene valutata da sinistra a destra. È consigliabile racchiudere un'istruzione relazionale tra parentesi tonde per agevolare la lettura del codice.
Operatori relazionali:

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

Per esempio, siano:
a = [44 56 73]
b= [56 11 73]

Istruzione	Risultato
x = = b	0
a(3)= =b(3)	1
a(2)= = b(2)	0
a(2)= = b(1)	1

Esercizio Dati i seguenti vettori:
a = [13 6 22]
b= [5 6 4]
Valutare:

1. a>b
2. a<b
3. a>=b
4. a<=b
5. a= =b
6. a~ =b

5.2 Operatori logici

Gli operatori logici utilizzati da Matlab sono 3:

~	NOT	~A restituisce un array avente le stesse dimensioni di A, i cui elementi sono pari a 1 se quelli di A sono nulli; pari a 0, altrimenti
&	AND	A&B restituisce un array delle stesse dimensioni di A e B, i cui elementi sono pari a 1 se i corrispondenti elementi di A e B sono ENTRAMBI diversi da 0; pari a 0 se almeno uno tra i due elementi di A e B è uguale a 0.
|	OR	A|B restituisce un array delle stesse dimensioni di A e B, i cui elementi sono pari a 1 se almeno uno tra i due elementi corrispondenti di A e B è diverso da 0; pari a 0 se sono ENTRAMBI uguali a 0.

La presenza di operatori relazionali e logici introduce nuovi livelli di precedenza:

Primo	parentesi tonde
Secondo	operatori aritmetici e operatore NOT; da sinistra a destra
Terzo	operatori relazionali; da sinistra a destra
Quarto	AND
Quinto	OR

Esercizio Dati i seguenti vettori:
a = [1 0 1 1 1 1]
b= [0 0 0 1 0 1]

Valutare le seguenti espressioni logiche:

1. NOT a
2. NOT b
3. a AND b
4. a OR b

5.3 Funzioni logiche

Le principali funzioni logiche implementate in Matlab sono:

any (x)	restituisce 1 se ALMENO un elemento di x è diverso da 0; altrimenti restituisce 0
any (A)	agisce per colonna
all (x)	restituisce 1 se TUTTI degli elementi di x sono diversi da 0; altrimenti restituisce 0
all (A)	agisce per colonna
find (x)	restituisce un vettore con gli indici degli elementi non nulli di x
[i, j, w]=find (A)	i contiene gli indici di riga degli elementi non nulli di A, j gli indici di colonna e w il valore degli elementi non nulli
xor (A, B)	restituisce un array delle stesse dimensioni di A e B, i cui elementi sono pari a 1 se SOLTANTO uno dei corrispondenti elementi di A e B è diverso da 0; pari a 0 in tutti gli altri casi

Esercizio Dati i vettori x=[5, -3, 18, 4] e y=[-9, 13, 7, 4] valutare:

1. z = ~y>x
2. z = ~(y>x)
3. z = x & y
4. z = x | y
5. z = xor (x, y)

5.4 Istruzioni condizionali

Le istruzioni condizionali permettono di scrivere programmi in grado di ***PRENDERE DECISIONI***. Specificamente, le istruzioni condizionali vengono utilizzate quando si vuole che determinate istruzioni vengano eseguite quando si verificano condizioni specifiche. Nel linguaggio quotidiano usiamo spesso strutture condizionali del tipo: se piove, prendo l'ombrello. Quando programmiamo dobbiamo trasformare i termini delle strutture condizionali in espressioni logico-matematiche. Una struttura condizionale per un programmatore può essere tradotta in questo modo: se una determinata variabile (o un insieme di variabili) assume certi valori, vengono eseguite certe istruzioni che modificano questa stessa variabile o altre variabili. La forma delle istruzioni condizionali è la seguente:

```
se condizione
        istruzioni
end
```

Matlab verifica l'espressione: se essa è **vera**, Matlab esegue le istruzioni successive fino all'operatore che indica la fine dell'istruzione condizionale (end); se è falsa salta alla prima riga successiva all'end.

5.4.1 Istruzione if

Sintassi:

```
if  espressione_logica
        istruzioni
end
```

L'espressione rappresenta una variabile di controllo, che permette di decidere se eseguire o meno le istruzioni che precedono l'end.
Per esempio, se vogliamo calcolare la radice quadrata di x soltanto per x maggiori o uguali a 9, possiamo digitare:

```
if x >= 9
        y = sqrt (x)
end
```

Per convenzione, le istruzioni interne ad un'istruzione condizionale vengono rientrate (**indentazione**) per semplificare la lettura dei programmi.
L'espressione_logica può essere una combinazione di istruzioni. È fondamentale comprendere il meccanismo di funzionamento della verifica dell'espressione. Il risultato è vero (e pertanto vengono eseguite le istruzioni) se **TUTTI** gli elementi risultanti dal calcolo one sono pari a 1. Si consideri il seguente esempio:

```
x = [3, -2, 0, 4, 6, 7]
if x<0
   disp ('Almeno un elemento del vettore è minore di zero')
end
```

In questo caso il comando disp non verrà MAI eseguito. Infatti:

```
x < 0
```

restituisce l'array: [0 1 0 0 0 0]. La verifica dell'espressione restituisce 0, poiché non tutti gli elementi dell'array sono pari a 1; pertanto, le istruzioni precedenti l'end non vengono eseguite.

5.4.2 Istruzioni condizionali annidate

È possibile annidare diverse istruzioni condizionali nel seguente modo:

```
if espressione_logica_1
        istruzioni_1
        if espressione_logica_2
                istruzioni_2
        end
end
```

Nel caso di istruzioni condizionali annidate l'indentazione risulta particolarmente utile.
Nel linguaggio quotidiano le istruzioni annidate corrispondono a forme tipo: se piove, prenderò l'ombrello e se ci sarà pure molto vento prenderò anche l'impermeabile.

5.4.3 Istruzioni else

Si utilizzano quando si intende eseguire un determinato insieme di istruzioni nel caso l'espressione di controllo risulti vera, un altro insieme di istruzioni nel caso risulti falsa:

```
if espressione_logica
        istruzioni_1
else
        istruzioni_2
end
```

Nel linguaggio comune le istruzioni else corrispondono a forme tipo: se piove, prenderò l'autobus, altrimenti farò due passi a piedi.

Esempio Vogliamo calcolare e visualizzare la radice quadrata di un numero immesso dall'esterno; se il numero è minore di 0 vogliamo visualizzare il messaggio 'Numero negativo immesso'.

```
x = input ('inserisci uno o più valori: ')
if x > = 0
        y = sqrt (x)
        disp ('La radice quadrata è: '), disp (y)
else
        disp ('Almeno un numero immesso è negativo')
end
```

Si noti che l'istruzione else comporta una decisione binaria su quali istruzioni eseguire. Per qualsiasi valore di x viene eseguito uno specifico insieme di istruzioni.

5.4.5 Istruzione elseif

Sintassi:

```
if espressione_logica_1
        istruzioni_1
elseif espressione_logica_2
        istruzioni_2
end
```

In questo caso, se l'espressione è verificata vengono eseguite le istruzioni_1, altrimenti se essa è falsa e contemporaneamente espressione_logica_2 è vera vengono eseguite le istruzioni_2.
Nel linguaggio comune le istruzioni elseif corrispondono a forme tipo: se piove, prenderò l'autobus, altrimenti se finirò di lavorare presto, tornerò a piedi passando per il centro.

Esempio Utilizzando opportunamente le istruzioni elseif ed else, scrivere il codice per calcolare il logaritmo naturale di x per x >10, la radice quadrata di x per $0 \leq x \leq 10$ e la funzione e^x-1 per x<0.

```
if x>10
        y=log(x)
elseif x>=0
```

```
        y=sqrt(x)
else
        y=exp(x)-1
end
```

Suggerimento: prima di scrivere il codice di una struttura decisionale è sempre meglio preparare un diagramma o uno schema del programma su carta.

Nell'esempio precedente il programma doveva prendere una decisione su quale istruzione eseguire in base al valore di x. Erano previsti tre casi mutuamente esclusivi ed esaustivi, poiché prevedevano tutti i possibili valori di x.

5.4.6 La struttura switch

È una struttura alternativa alle istruzioni if, else ed elseif. È utile quando il processo decisionale prevede numerose alternative.

Sintassi

```
swicth variabile_input
case valore_1
        istruzioni_1
case valore_2
        istruzioni_2
…
otherwise
        istruzioni_otherwise
end
```

Matlab confronta la variabile_input (scalare o stringa) con il valore che segue il case: se sono uguali vengono eseguite le istruzioni corrispondenti. Se nessun valore corrisponde alla variabile_input vengono eseguite le istruzioni_otherwise. La presenza dell'otherwise è facoltativa.

5.5 Cicli

Un ciclo (*loop*) è una struttura utilizzata dai programmatori per **RIPETERE** un insieme di istruzioni un certo numero di volte (*passaggi*). In generale, distinguiamo due tipi di cicli:

1. Cicli *for*: il numero di passaggi è noto in anticipo e viene impostato dal programmatore
2. Cicli *while*: il numero di passaggi **NON** è noto in anticipo, ma il ciclo termina quando viene soddisfatta una condizione specifica.

5.5.1 Il ciclo for

Sintassi:

```
for variabile_ciclo = espressione
    istruzioni
end
```

L'espressione rappresenta un insieme di valori. Al primo passaggio del ciclo, Matlab assegna il primo valore alla variabile_ciclo e così via fino all'ultimo valore. Un semplice esempio aiuterà a chiarire il concetto:

```
x=0;
for i = 1 :10
        x=x+1
end
```

L'espressione 1:10 genera un vettore linearmente intervallato, con passo unitario, da 1 a 10. Quando Matlab entra nel ciclo, assegna alla variabile_ciclo il valore 1, al secondo passaggio il valore 2 e così via. L'ultimo passaggio coincide con l'ultimo valore del range di variazione della variabile_ciclo. Terminato il ciclo vengono eseguite le istruzioni successive all'end.
È possibile annidare cicli for e istruzioni condizionali.

Esempio Dato il seguente array x=[1.92 0.05 -2.43 -0.02 0.09 0.85 -0.06], creare un programma per sostituire tutti gli elementi di x tali che $-0.1 \leq x_i \leq 0.1$ con il valore 0.
Soluzione:

```
y = [ ]; z = [ ];
for i =1 : length (x)
        if abs(x(i))>=0.1
                y=[y, x(i)]
        else
                y=[y, 0]
        end
end
```

5.5.2 Il ciclo while

A differenza del ciclo *for*, il ciclo *while* si utilizza quando non si conosce in anticipo il numero di passaggi. Sintassi:

```
while condizione
        istruzioni
end
```

Prima di ogni passaggio, Matlab verifica la condizione, se è vera entra nel ciclo, esegue le istruzioni fino all'end e torna a verificare la condizione; se è ancora vera esegue nuovamente le istruzioni e così via. Quando Matlab verifica che la condizione è diventata falsa il ciclo termina e vengono eseguite le istruzioni successive all'end. La variabile contenuta nella condizione prende il nome di variabile di ciclo.

ATTENZIONE: affinché il ciclo *while* venga correttamente implementato è necessario accertarsi che:

1. la variabile di ciclo contenga un valore al momento in cui Matlab entra nel ciclo
2. la variabile di ciclo venga modificata durante l'esecuzione delle istruzioni la variabile indicata nella condizione. Altrimenti, se la condizione è vera Matlab non uscirà MAI dal ciclo, generando un ciclo infinito.

Consideriamo il seguente esempio:

```
x=0;
while x ~ = 10
  disp (x)
   x = x+1;
end
```

Quando Matlab entra nel ciclo verifica se x è diverso da 10; poiché x è uguale a 0, la condizione è vera e vengono eseguite le istruzioni. Il punto cruciale è l'istruzione che modifica il valore di della variabile contenuta nella condizione. Questo assicura che ad ogni passaggio la variabile venga aggiornata. Quando Matlab verifica che la condizione è diventata falsa, x=10 nel nostro esempio, esce automaticamente dal ciclo.

5.6 Le funzioni definite dall'utente

I programmatori Matlab possono implementare speciali M-file chiamati file di funzione. A differenza degli script tutte le variabili in un file funzione sono LOCALI: i loro valori sono disponibili solo all'interno della funzione.

5.6.1 Regole sintattiche dei file di funzione

Un file funzione DEVE iniziare con una riga di definizione della funzione, che contiene l'elenco delle variabili di input e output ed il nome della funzione:

```
function [varibili_output] = nome_funzione (variabili_input)
```

Fare molta attenzione alle seguenti regole:

1. la parola **function** deve avere tutte le **lettere minuscole**
2. le variabili di output devono essere racchiuse tra parentesi **quadre**
3. le variabili di input devono essere racchiuse da parentesi **tonde**
4. il nome del file deve essere uguale al nome della funzione

E' utile scrivere una riga di commento, contenente le parole chiave della funzione, immediatamente dopo la riga di definizione della variabile per facilitare le ricerche con il comando lookfor.

Esempio Scrivere una funzione per calcolare la velocità e la distanza percorsa da un corpo in caduta libera con velocità iniziale v_0. Le variabili di input sono la velocità iniziale ed il tempo trascorso. Si supponga che le variabili di input siano array.

```
function [velocita, distanza] = drop (v0, t)
% velocità e distanza percorsa corpo caduta libera
g = 9.8;
velocita = g * t + v0;
distanza = t ^ 2 * g + v0 * t;
```

5.6.2 Chiamare le funzioni

Per chiamare una funzione è sufficiente digitarne il nome seguito dagli argomenti racchiusi tra parentesi tonde. Sebbene non sia obbligatorio specificare le variabili di output, nella maggioranza dei casi chiameremo una funzione allo scopo di passare al file script chiamante un certo risultato. In questo caso bisognerà dire al programma in quale variabile registrare l'output della funzione. Il nome di tale variabile può coincidere con quello specificato nella definizione della funzione.
Nel nostro esempio, supponiamo di passare alla funzione una velocità iniziale di 30 m/s ed un intervallo di 10s:

```
[vel, dist]=drop(30, 10)
```

La velocità e la distanza saranno registrate nelle variabili **GLOBALI** vel e dist.

Esercizio Modificare lo script precedente in modo da passare alla funzione un vettore di velocità iniziali ed un vettore di intervalli temporali.

5.6.3 Variabili locali

Tutte le variabili di un file funzione sono LOCALI. Ciò significa che vengono cancellate dopo l'esecuzione della funzione. L'unica eccezione a questa regola si verifica quando i nomi delle variabili in cui vogliamo registrare l'output della funzione coincidono con i nomi delle variabili di output utilizzati nella definizione della funzione.

Problemi

5.1) Dati i seguenti array:
x=[-3 0 0 2 6 8]
y=[-5 -2 0 3 4 10]
Utilizzare Matlab per trovare gli elementi di x maggiori dei corrispondenti elementi di y.

5.2) Il seguente array contiene le quotazioni di un noto titolo azionario x=[220 330 221 198 208 206 188 196 200 204 219 205 177]. Usare Matlab per calcolare il numero di giorni in cui il titolo è stato superiore a 200. Calcolare media e deviazione standard.

5.3) I seguenti array contengono le quotazioni di due titoli azionari in un periodo di due settimane:
a=[220 330 221 198 208 206 188 196 209 200 198 204 219 205 177]
b=[167 189 200 199 202 208 200 193 199 210 216 222 210 202 201]
Usare Matlab per trovare il numero dei giorni in cui il titolo a è stato superiore a b. Calcolare media e deviazione standard di a e b.

5.4) Dati i seguenti array:
x=[-3 0 0 2 6 8]
y=[-5 -2 0 3 4 10]
Calcolare a mano le seguenti operazioni e valutarle successivamente con Matlab:

1. z = y < ~ x
2. z = x & b
3. z = ~ (x & b)
4. z = x | b
5. z = x | ~ b
6. z = xor (x, y)
7. z = xor (~ x+y, y)

5.5) Dato il seguente array x=[1.92 0.05 -2.43 -0.02 0.09 0.85 -0.06], creare un programma per eliminare tutti gli elementi di x tali che $-0.1 \leq x_i \leq 0.1$ e sostituirli con altrettanti 0 alla fine dell'array. Il nuovo array deve essere x2=[1.92 -2.43 0.85 0 0 0 0].

5.6) Il seguente array contiene le quotazioni di un titolo azionario:
x=[167 189 200 199 202 208 200 193 199 210 216 222 210 202 201]
Un investitore ha 1000 azioni il primo giorno e intende comprare 100 azioni ogni giorno che il prezzo scende sotto le 200 lire e venderne 100 quando il prezzo sale sopra le 210 lire. Usare Matlab per calcolare:

1. la spesa totale per acquistare le azioni
2. il guadagno totale dovuto alla vendita delle azioni
3. il numero totale di azioni dopo 10 e 15 giorni

[borsa.m]

5.7) Utilizzando opportune istruzioni condizionali annidate, scrivere il codice per calcolare: y = ln x per x > 0 ed y = 5 x per $x \leq 0$; z = 4 y per $y \geq 3$, z = 2 y per $1 \leq y < 3$, z = 0 per $y < 1$ e z = 7 x per $x \leq 0$.
Suggerimento: annidare i condizionali in modo da separare le decisioni basate sulla variabile x da quelle basate sulla variabile y.

[es_elseif.m]

5.8) Scrivere uno script per visualizzare la direzione dell'ago di una bussola in funzione dei seguenti angoli (rispetto all'Est): 0, 45, 90, 135, 180, 225, 270, 315. Usare la struttura switch.

[bussola.m]

5.9) Utilizzare Matlab per calcolare quanto tempo occorre per accumulare un miliardo di lire in un conto corrente con 10 milioni iniziali e tenendo conto che vengono depositati 10 milioni alla fine di ogni anno e che la banca offre un tasso di interesse annuo del 5%.

5.10) Date le seguenti matrici:

A=

$$\begin{pmatrix} 3 & -2 & 1 \\ 6 & 8 & 5 \\ 7 & 9 & 10 \end{pmatrix}$$

B=

$$\begin{pmatrix} 6 & 9 & -4 \\ 7 & 5 & 3 \\ -8 & 2 & 1 \end{pmatrix}$$

Calcolare A*B usando i cicli e verificare il risultato.

5.11) Data la matrice:

A=

$$\begin{pmatrix} 3 & 7 & -4 & 12 \\ -5 & 9 & 10 & 2 \\ 6 & 13 & 8 & 11 \\ 15 & 5 & 4 & 1 \end{pmatrix}$$

Calcolare la media di ogni colonna usando i cicli e verificare il risultato.

5.12) Scrivere una funzione sin_gradi che accetta un angolo x espresso in grado e restituisce sin x sempre in gradi. Suggerimento: chiamare la funzione sin.

5.13) Creare una matrice quadrata *nxn* che abbia il valore 1 in tutti gli elementi della prima colonna e della prima riga ed i restanti elementi siano pari alla somma dell'elemento che occupa la cella immediatamente superiore e dell'elemento immediatamente a sinistra; se la somma è maggiore o uguale a 20, gli elementi sono pari al valore massimo tra i due elementi superiore e a sinistra.

[matrice_esercizio.m]

5.14) Scrivere una funzione che riceve come argomento una temperatura espressa in gradi Fahrenheit e restituisce il valore corrispondente in gradi Celsius. Suggerimento: C=5/9(F-32).

Grafica bidimensionale

Matlab prevede numerose funzioni che permettono di creare grafici, sia bidimensionali sia tridimensionali. In questo corso ci limiteremo a considerare i diagrammi bidimensionali. Per visualizzare le informazioni sulle funzioni grafiche digitare help graph2d.

6.1 Diagrammi xy

Sono utilizzati per rappresentare funzioni tipo y = f (x). La variabile indipendente (x) è rappresentata lungo l'asse orizzontale e la variabile dipendente (y) lungo l'asse verticale.
Un diagramma può essere ottenuto da dati sperimentali o da un'espressione matematica.
Per costruire correttamente un diagramma è necessario che:

1. ogni asse abbia un titolo o etichetta con il nome di ciò che rappresenta e l'unità di misura.
2. ogni asse deve avere segni di graduazione (tick mark) regolarmente distanziati.
3. se il diagramma visualizza più curve utilizzare una legenda per distinguerle
4. se il diagramma visualizza diversi insiemi di dati sperimentali utilizzare simboli diversi (cerchio, quadrato, croce, ecc.)
5. NON utilizzare i simboli per distinguere curve generate da equazioni.

6.2 Creare un diagramma

Per generare un diagramma si utilizza la funzione:

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

Se x e y sono vettori Matlab genera una curva con i valori di x sull'asse orizzontale e i valori di y sull'asse verticale.
La seguente tabella riporta una serie di comandi utili per completare il diagramma:

xlabel ('testo')	genera i titoli o etichette dell'asse x
ylabel ('testo')	genera i titoli o etichette dell'asse y
title ('testo')	genera il titolo del grafico
grid on	aggiunge una griglia al diagramma
grid off	elimina la griglia dal diagramma
axis ([xmin xmax ymin ymax])	imposta i valori dei limiti degli assi
axis square	seleziona automaticamente i limiti degli assi in modo da ottenere un diagramma quadrato
axis equal	imposta una spaziatura identica per gli assi
axis auto	imposta automaticamente limiti "ideali" per una corretta visualizzazione

L'ordine di questi comandi non è rilevante purché seguano il comando plot e siano su un'unica riga, separati dall'operatore virgola (,).

Esempio

```
x=[0:0.1:52];
y=0.4*sqrt(1.8*x);
plot (x,y), xlabel ('Distanza (km)'), ylabel ('Altezza (km)'),...
title ('Altezza di un razzo in funzione della distanza orizzontale'), grid on, axis ([0 52 0 5])
```

6.3 Il comando fplot

Sintassi:

```
fplot (‘stringa’, [xmax xmin])
```

dove ‘stringa’ è il testo che descrive la funzione da rappresentare e [xmax xmin] l’intervallo della varibile x.

Esempio

```
k='cos(tan(x))-tan(sin(x))';
fplot(k,  [1 2])
```

Il comando fplot è utile perché permette di stabilire automaticamente il numero di punti da utilizzare per rappresentare tutte le caratteristiche di una funzione.

Esercizio Generare il grafico dell’esempio precedente utilizzando il comando plot con 100 punti e confrontare il risultato con quello dell’esempio precedente

6.4 Diagrammi multipli

Per generare più digrammi nella stessa figura utilizzate il comando subplot.
Sintassi:

```
subplot (m,n,p)
```

Il comando suddivide la figura in *mxn* pannelli e posiziona il diagramma nel pannello *p*.

Esempio

```
x=[0:0.01:5];
y=exp(-1.2*x).*sin(10+x+5);
subplot (1,2,1)
plot(x,y), xlabel ('x'), ylabel ('y'), axis ([0 5 -1 1])
x=[-6:0.01:6];
y=abs(x.^3-100);
subplot (1,2,2)
plot(x,y), xlabel ('x'), ylabel ('y'), axis ([-6 6 0 350])
```

6.5 Diagrammi sovrapposti

Per generare diagrammi sovrapposti si utilizza il comando plot (x,A) dove x è un vettore ed A una matrice. Il comando visualizza ogni colonna di A in funzione di x, se la dimensione di x è pari alle righe di A; visualizza ogni riga di A in funzione di x, se la dimensione di x è pari alle colonne di A.

Esempio

```
x=[0:0.01:2];
y=sinh(x);
z=tanh(x);
A=[y' z']
plot (x,A)
```

Analisi dei segnali

Il Toolbox Analisi dei Segnali mette a disposizione strumenti per l'analisi dei segnali; in particolare, strumenti per la stima della correlazione e della covarianza e della densità spettrale di potenza. Le sezioni seguenti danno informazione su questi strumenti.

7.1 Correlazione e Covarianza.

Le funzioni **xcorr** e **xcov** stimano la mutua-correlazione (cross-correlation) e la mutua-covarianza (cross-covariance) di un segnale. Ovviamente, l'autocorrelazione e l'autocovarianza possono essere calcolate con le stesse funzioni. La mutua-correlazione è una quantità statistica definita come

$$R_{xy}(m) = E\{x_{n+m} y_n^*\} = E\{x_n y_{n-m}^*\}$$

Dove x_n e y_n sono segnali stazionari. La covarianza è la correlazioni dei due segnali a cui è sottratta la media.

$$C_{xy}(m) = E\{(x_{n+m} - \mu_x)(y_n - \mu_y)^*\} = R_{xy}(m) - \mu_x \mu_y^*$$

In pratica, si può solo stimare queste sequenze essendo possibile utilizzare solo segmenti finiti del segnale. Una stima di tale funzione basata su N campioni si x_n and y_n è data dalla:

$$\hat{R}_{xy}(m) = \begin{cases} \sum_{n=0}^{N-m-1} x_{n+m} y_n^* & \text{m} \geq 0 \\ \hat{R}_{yx}^*(-m) & \end{cases}$$

dove x_n and y_n vanno da 0 a N-1, e l'autocorrelazione da -(N-1) to N-1. La funzione **xcorr** valuta questa somma con un efficiente algoritmo basato sulla FFT. L'operazione è equivalente alla convoluzione.
Per esempio:

```
x = [1 1 1 1 1]';
y = x;
xyc = xcorr(x,y)
xyc =
   1.0000
   2.0000
   3.0000
   4.0000
   5.0000
   4.0000
   3.0000
   2.0000
   1.0000
```

Si noti che la sequenza è lunga il doppio meno uno, così che l'ennesimo elemento è la correlazione al ritardo 0.

7.2 Analisi di Fourier

L'analisi di Fourier è uno strumento estremamente utile per l'analisi dei dati che decompone un segnale nelle sue componenti sinusoidali. Per vettori di campioni l'analisi di Fourier è realizzata usando una trasformata discreta la DFT. La FFT (Fast Fourier Transform) è un metodo efficiente per calcolare la DFT. Per una sequenza di ingresso di lunghezza N si ha:

$$X(k)=\sum_{n=1}^{N} x(n)e^{-j2\pi(k-1)\left(\frac{n-1}{N}\right)} \qquad 1\le k\le N$$

$$x(n)=\frac{1}{N}\sum_{n=1}^{N} X(k)e^{j2\pi(k-1)\left(\frac{n-1}{N}\right)} \qquad 1\le n\le N$$

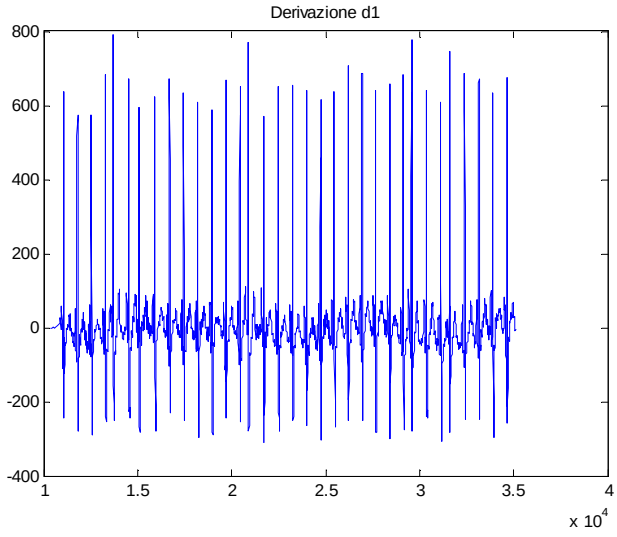
La sommatoria è estesa da 1 a N perché in MATLAB i vettori ammettono solo indici maggiori di zero. Se x(n) è reale la precedente equazione può anche essere scritta in termini di seni e coseni.

Esempi di calcolo
La FFT di un vettore colonna x
x = [4 3 7 -9 1 0 0 0]' ;
può essere calcolata con la
y = fft(x)
il cui risultato in y
y =
6.0000
11.4853 - 2.7574i
-2.0000 -12.0000i
-5.4853 +11.2426i
18.0000
-5.4853 -11.2426i
-2.0000 +12.0000i
11.4853 + 2.7574i
Si noti che nonostante la sequenza x è reale, y è complessa. Il primo elemento è la continua e il quinto corrisponde alla frequenza di Nyquist, gli ultimi tre valori di y corrispondono alle frequenze negative.

Supponiamo di volerci calcolare lo spettro del segnale ECG dato nel file ecg1201__asc:

Carichiamo il dato e visualizziamolo

```
load ecg_filtered
tempo = ecg_filtered (:,1);
d1 = ecg_filtered (:,2);
plot(tempo,d1)
title ('Derivazione d1')
```

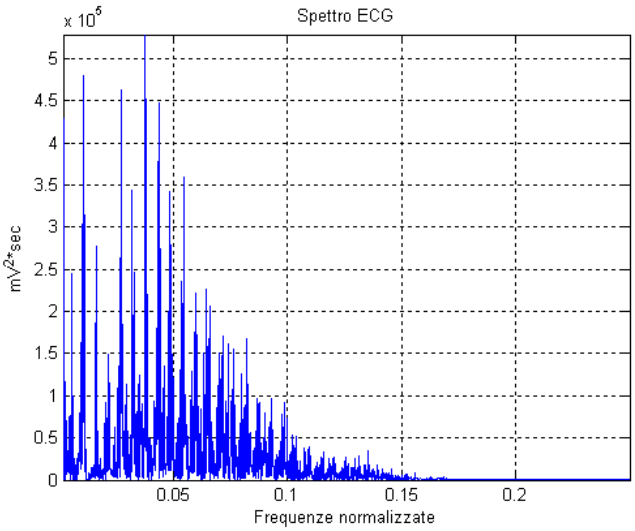


La trasformata del vettore d1 è data da

```
D1_F = fft(d1);
```

Il risultato è un vettore complesso D1_F, il suo quadrato è detto potenza del segnale e il plot della potenza in funzione della frequenza è detto periodogramma. Si suggerisce per la visualizzazione di rimuovere la continua che rappresenta solo il valore medio del segnale.

```
N = length(D1_F);
D1_F(1) = [];
power = (abs(D1_F(1:N/2)).^2)/N;
nyquist = 1/2;
freq = [1:N/2]*[nyquist/(N/2)];
plot(freq(1:N/4),power(1:N/4)), grid on
xlabel('Hz')
title('Periodogramma')
```

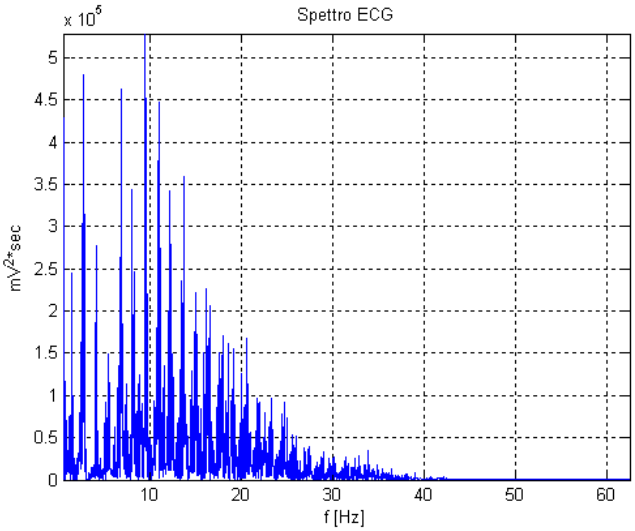


volendo ottenere un diagramma in frequenze non normalizzate le due righe

```
nyquist = 1/2;
freq = [1:N/2]*[nyquist/(N/2)];
```

ricordando che i dati sono stati campionati a 250 Hz, devono essere sostituite da

```
nyquist = 250/2;
freq = [1:N/2]*[nyquist/(N/2)];
```



Per stimare modulo e fase della trasformata si possono usare le funzioni **abs** e **angle.**

7.3 Filtri.

Il toolbox di analisi dei segnali ha a disposizione funzioni per il progetto dei filtri, l'analisi e la realizzazione. Qui di seguito presentiamo una funzione per l'analisi e una per la realizzazione del filtro.

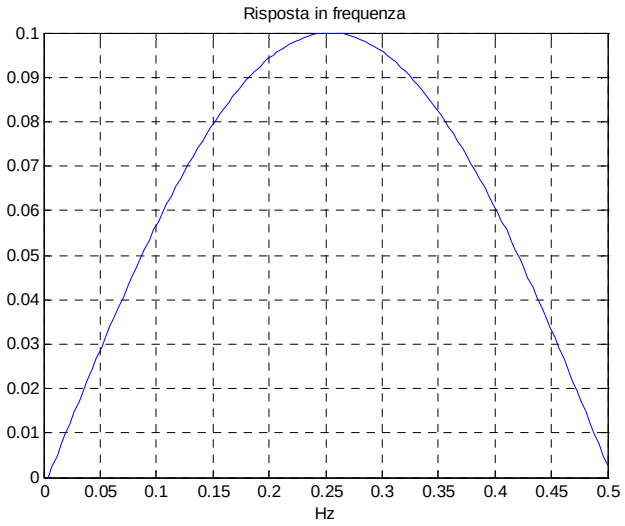
Dato un filtro digitale espresso dai vettori a,b con a coefficienti della parte ricorsiva e b di quella non ricorsiva la sua risposta in frequenza può essere calcolata con la funzione **freqz la cui sintassi è**

```
[h,w] = freqz(b,a,l)
```

Che restituisce la risposta in frequenza nel vettore h e la corrispondente risposta nel vettore w. I vettori h e w sono entrambi di lunghezza l. Il vettore w ha valori che variano tra 0 e π radianti per campione. Quando non si specifica l la risposta in frequenza è calcolata usando un valore di 512 campioni.

Esempio

```
b = [-.05 0 .05];
[h,w] = freqz (b,1,128);
N = length(h);
Freq_risp= abs(h);
nyquist = 1/2;
freq = (1:N)/ (N)*nyquist;
plot(freq,Freq_risp), grid on
xlabel('Hz')
title('Risposta in frequenza')
```

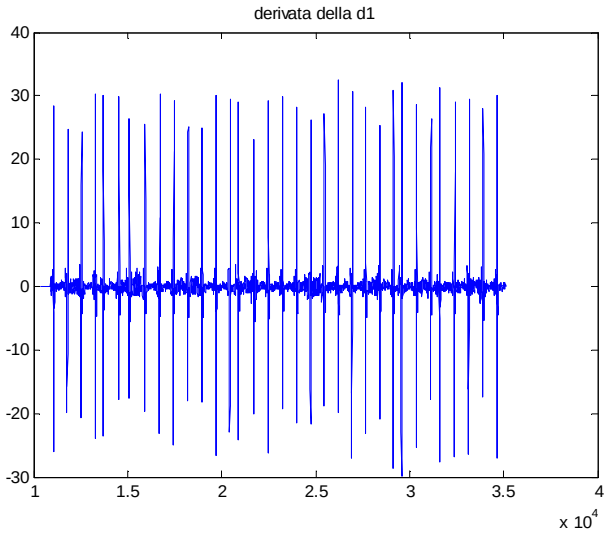


Per realizzare un filtro si usa la funzione **filter**

```
y = filter(b,a,x)
```

con x segnale di ingresso, a parte ricorsiva del filtro, b parte non recorsiva.
Utilizzando i dati degli esempi precedenti si ottiene:

```
y = filter(b,1,d1);
plot(tempo,y)
title ('derivata della d1')
```



Per la progettazione di filtri esistono differenti funzioni di progettazione la più semplice è la funzione **fir1**che ha la seguente sintassi:

b = fir1(n,Wn)
b = fir1(n,Wn,'ftype')
b = fir1(n,Wn,window)
b = fir1(n,Wn,'ftype',window)

dove *b* è il vettore dei coefficienti ordinato per potenze negative

$B(z) = b(1) + b(2)\, z^{-1} + b(3)\, z^{-2} + \ldots + b(n)\, z^{-n}$

Wn è un numero compreso tra 0 e 1 (0 rappresenta la frequenza nulla, 1 la frequenza di nyquist) che rappresenta la frequenza di taglio nel caso di filtri passa alto e passa basso, o un vettore a due elementi (frequenza di taglio inferiore e superiore) nel caso di filtri passa-banda e elimina–banda o nel caso di filtri multi banda un vettore a più elementi. In tal caso per il vettore $[w_1\ w_2\ w_3\ \ldots\ w_n]$ si ottiene un filtro con le bande $[0\ w_1]\ [w_2\ w_3]\ \ldots\ [w_n\ 1]$

ftype può assumere più valori ‘high’ per passa alto e ‘stop’ per ottenere un elimina-banda

window permette di modificare la funzione finestra utilizzata per la progettazione.

Analisi delle immagini

Il Toolbox Analisi delle Immagini mette a disposizione oltre a strumenti per la visualizzazione anche strumenti per l'analisi delle immagini; in particolare strumenti per il filtraggio e l'Image Enhancement.
Il Toolbox lavora su differenti tipi di immagine e su differenti classi di memorizzazione. Qui di seguito si riporta una tabella con le definizioni dei tipi di immagini e delle classi.

Tipo di immagine	Definizione
Immagine Binaria	Un immagine binaria contiene solo pixels bianchi o neri. In MATLAB un immagine binaria è rappresentata come un array logico di zeri ed uni che rappresentano rispettivamente i pixels bianchi e neri.
Immagini Indicizzate	E' un immagine in cui i valori dei pixel indirizzano una mappa di colori RGB. In MATLAB le immagini indicizzate sono rappresentate da array di classe uint8 (intero a 8 bits), uint16 (intero a 16 bits) o double (a virgola mobile). La mappa è sempre un array di double mx3 .
Immagini Intensità	E' un immagine in cui valori dei pixel consistono in valori di intensità di una scala di grigi. In MATLAB le immagini intensità sono rappresentate da array di classe uint8 (intero a 8 bits), uint16 (intero a 16 bits) o double (a virgola mobile). Anche se le immagini intensità non si riferiscono ad un mappa di colori MATLAB usa per visualizzarle la mappa di sistema.
Immagini Multiframe	Contengono più di un immagine o frame. Sono memorizzate in array 4-D dove la 4° dimensione specifica il numero di frame.
Immagini RGB	E' un immagine i cui pixel contengono tre valori rappresentanti le tre componenti del colore del pixel: il rosso (Red), il verde (Green) ed il blu (Blue). Sono memorizzati in array mxnx3.

Alcune funzioni presenti nel toolbox non sono in grado di utilizzare tutte le differenti classi di memorizzazione per le differenti immagini. Nella guida del MATLAB, quando necessario, è prevista una specifica sezione intitolata "Class Support".

8.1 Filtraggio di immagini.

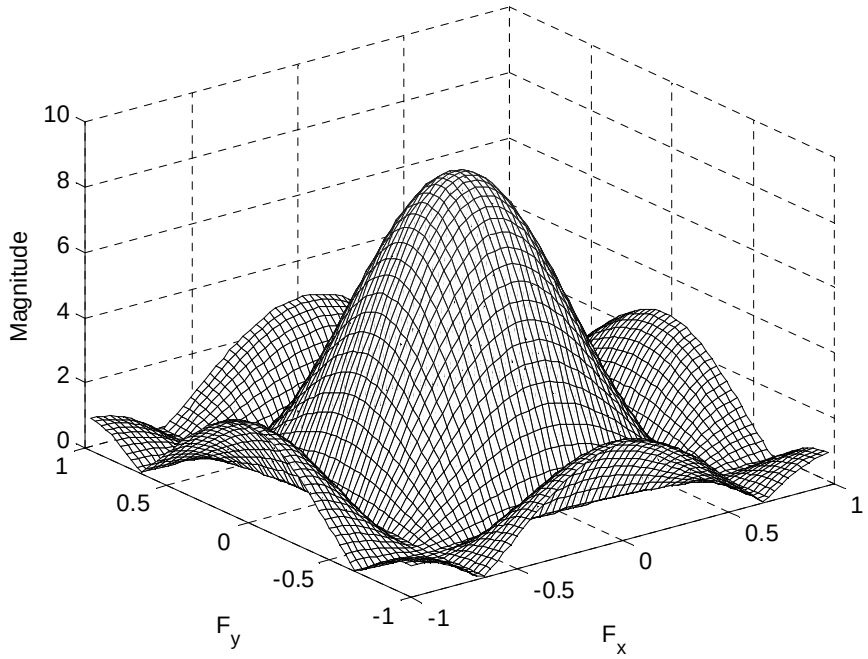
Il toolbox di analisi delle immagini ha a disposizione funzioni per l'analisi e la realizzazione di filtri digitali 2D. Qui di seguito presentiamo una funzione per l'analisi e una per la realizzazione del filtro.

Dato un filtro digitale espresso dalla matrice C la sua risposta in frequenza può essere calcolata con la funzione freqz2 la cui sintassi è

```
[h,f1,f2] = freqz2(C)
```

Che restituisce la risposta in frequenza nel vettore h e la corrispondenti frequenze normalizzate nei vettori f1,f2. Il valore 1.0 di tali frequenze corrisponde a metà della frequenza di campionamento o a л radianti.
Esempio

```
C = [1 1 1; 1 1 1; 1 1 1];  % maschera SM1
freqz2(C);
```



Per realizzare un filtro si usa la funzione **filter2**

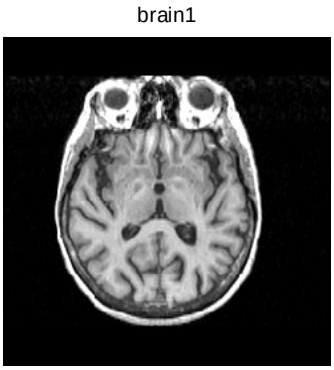
Y = filter2(h,X,shape)

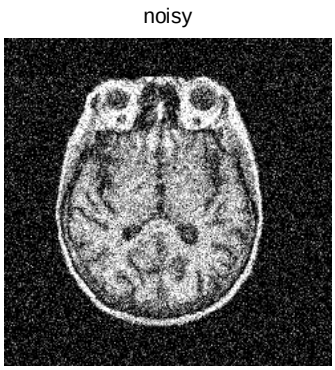
con segnale di ingresso X, il filtro h e shape che può assumere i seguenti valori:

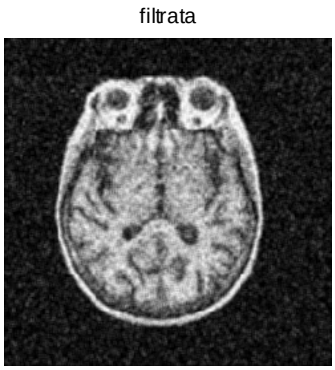
- **'full'** Restituisce la correlazione bidimensionale completa. In questo caso, Y ha dimensioni maggiori di X.
- **'same'**(default) Restituisce la parte centrale della correlazione. In questo caso, Y ha la stessa dimensione di X.
- **'valid'** Restituisce solo la parte di correlazione calcolata senza l'aggiunta di zeri ai bordi. In quest caso, Y è più piccolo di X.

Esempio:

```
load Brain1
imshow(brain1)
% per aggiungere rumore gaussiano a media nulla e varianza = 0.05
noisy = imnoise(brain1, 'gaussian', 0.0, 0.05);
imshow(noisy)
filtrata = filter2(C, noisy,'same');
filtrata = uint8(filtrata); % necessaria per ottenere una matrice di interi a 8 bit
imshow(filtrata)
```







8.2 Istogramma e sue manipolazioni.

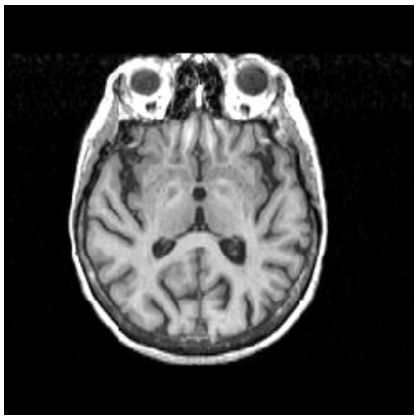
Per ottenere l'istogramma di un'immagine utilizzare la funzione:

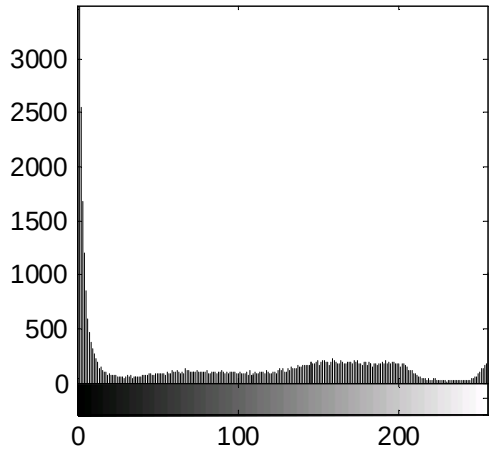
imhist(I,n)

che visualizza l'istogramma dell'immagine I in n barre; se non specificato n = 256 se I è un immagine Intensità, n=2 se I è un immagine binaria.

Esempio:

```
load Brain1;
imshow(brain1);
imhist(brain1);
```



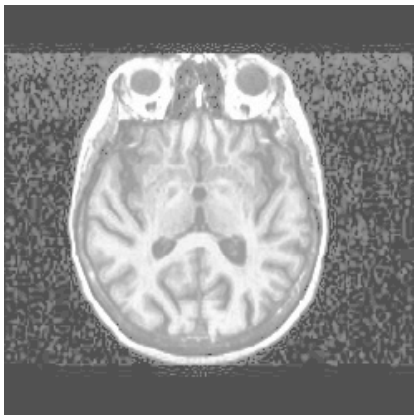


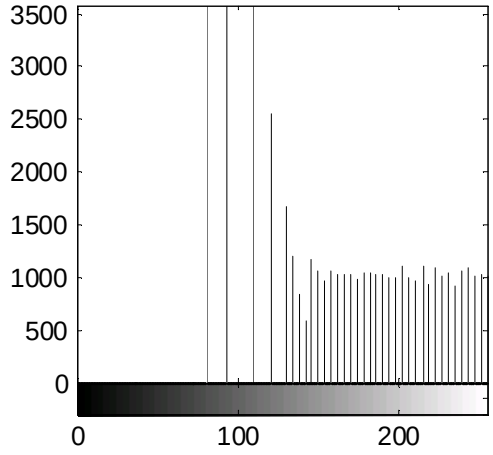
Per ottenere un'immagine equalizzata utilizzare la funzione

J = histeq(I)

Esempio:

```
brain2 = histeq(brain1);
imshow(brain2);
imhist(brain2);
```





Graphic user interface

Le interfacce grafiche sono ormai pervasive in gran parte della applicazioni. Esse rendono l'utilizzo delle applicazioni più semplice, più intuitivo, più produttivo. Le interfacce grafiche rendono più agevole l'apprendimento dell'uso delle applicazioni, riducono le possibilità di errori ed avvicinano ai computer un utenza meno specializzata.

Tuttavia la creazione di interfacce grafiche richiede un impegno di implementazione notevole. Studi recenti riportano che in media il 45% del codice di un'applicazione è dedicato all'interfaccia grafica. Per l'interfaccia grafica è speso il 45% del tempo nella fase di progetto, il 50% del tempo nella fase di implementazione ed il 37% del tempo nelle manutenzioni. Inoltre aggiungere a posteriori un'interfaccia grafica ad un'applicazione non è compito agevole e spesso dà luogo a risultati deludenti; pertanto l'interfaccia grafica va progettata insieme con l'applicazione stessa.

In questo sezione della guida al MATLAB studieremo le tecniche per la progettazione e l'implementazione di interfacce grafiche. La programmazione di interfacce grafiche utilizza tecniche specifiche, che, pur applicandosi anche in altri campi, hanno qui un ruolo insostituibile, in particolare la **programmazione ad eventi** e la **programmazione ad oggetti**.

Queste tecniche sono necessaria per superare il modello di programmazione tradizionale, che mal si adatta alla flessibilità richiesta da un'interfaccia grafica.

9.1 Ambienti visuali

Nel ***modello di programmazione tradizionale*** tutta la attività del sistema è guidata dal programma in esecuzione che emette ordini alle varie componenti del sistema, ivi compreso l'utente. Gli strumenti tradizionali di programmazione sono un Editor di testi e un Compilatore. I programmi creati con questi strumenti solitamente hanno lo scopo di elaborare dei dati in genere letti da file su disco o immessi da terminale e stampare i risultati.

Si consideri il più elementare programma per il calcolo della somma di *n* numeri, che interagisce con l'utente da tastiera. È il programma che chiede all'utente di fornire i dati, e l'utente non ha scelta sulle operazioni che può fare in quel momento. Dopo aver inserito alcuni dei numeri da sommare, non può ad esempio cambiare i valori che ha fornito nelle righe precedenti, se non interrompendo il programma e rilanciandolo dall'inizio.

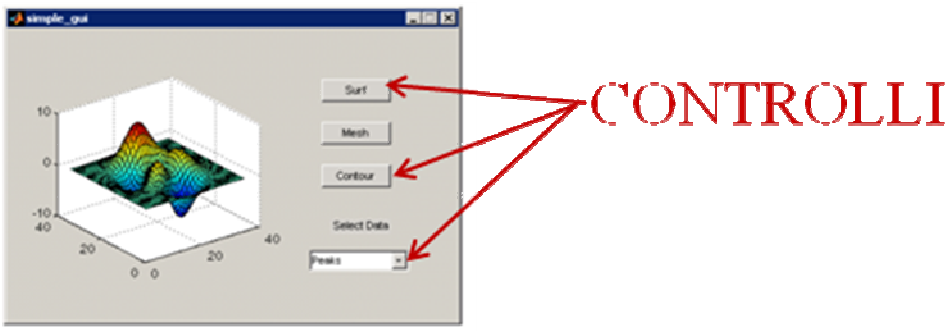
Con la diffusione dei sistemi operativi a interfaccia grafica **GUI (Graphical User Interface)** è cambiata la concezione dei programmi. Con le GUI, i programmi presentano all'utente una serie di finestre (maschere o form) contenenti degli oggetti predefiniti che consentono di dare dei comandi, visualizzare, modificare dei dati.

Per realizzare le interfacce grafiche che lasciano ampia libertà di interazione all'utente, occorre superare il modello tradizionale di programmazione per passare ad un ***modello reattivo***, in cui il sistema di elaborazione presenta all'utente una serie di elementi a sua disposizione, con cui l'utente interagisce a sua scelta e nell'ordine che più gli conviene.

Nei linguaggi tradizionali esiste il **programma principale** dal quale si richiamano i **sottoprogrammi** (procedure sequenziali).

Negli ambienti a interfacce grafiche questo modo di procedere non è più concepibile. Nelle GUI l'utente comunica con il programma tramite delle **maschere** nelle quali si sposta con il mouse anche in modo non sequenziale usando indifferentemente sia il mouse che la tastiera

Negli ambienti visuali le applicazioni si costruiscono creando una serie di maschere nelle quali si posizionano degli elementi detti **controlli** (caselle di testo, pulsanti, ecc) che consentono all'utente di vedere o modificare i dati e di dare comandi. Le azioni dell'utente devono quindi essere gestite **associando un codice ai controlli** in modo che questi rispondano opportunamente alle suddette azioni.



Negli ambienti di sviluppo delle GUI i controlli vengono creati prelevandoli da un pannello e disegnandoli sulla maschera.
Il controllo può essere poi specializzato modificando le sue caratteristiche.
Il controllo è reso attivo dotandolo delle funzionalità richieste e cioè provvedendo a scrivere il codice delle azioni corrispondenti.

9.2 Programmazione ad eventi

Per realizzare le interfacce grafiche sofisticate a cui siamo oggi abituati e che lasciano ampia libertà di interazione all'utente, occorre superare il modello tradizionale per passare ad un modello **reattivo**, in cui il sistema di elaborazione presenta all'utente una serie di elementi a sua disposizione, con cui l'utente interagisce a sua scelta e nell'ordine che più gli conviene.
Un evento è un messaggio che il sistema genera in risposta ad una azione effettuata dall'utente quando questi interagisce con l'applicazione.

- ✓ Gli eventi sono gestiti da procedure-evento che specificano il comportamento dell'applicazione in risposta all'evento.
- ✓ Gli eventi più comuni sono quelli generati dal mouse o dalla tastiera.
- ✓ Quasi sempre una azione dell'utente genera più di un evento.
- ✓ Per gestire un evento esistono due possibili soluzioni:
- ✓ La tecnica del **Read-evaluation loop** in cui l'applicazione gestisce tutti i possibili eventi anche quelli ai quali non è interessata.
- ✓ La tecnica del **Notification-based** che prevede la presenza di un controllore degli eventi esterno all'applicazione.

Notification-based

- ✓ Il ciclo di controllo degli eventi risiede all'esterno dell'applicazione in un *notifier*.
- ✓ L'applicazione comunica al *notifier gli eventi ai quali è* interessata e gli associa una funzione ***call-back***.
- ✓ Il *notifier filtra gli eventi. Quando trova un evento* registrato passa il controllo alla relativa funzione *callback.*

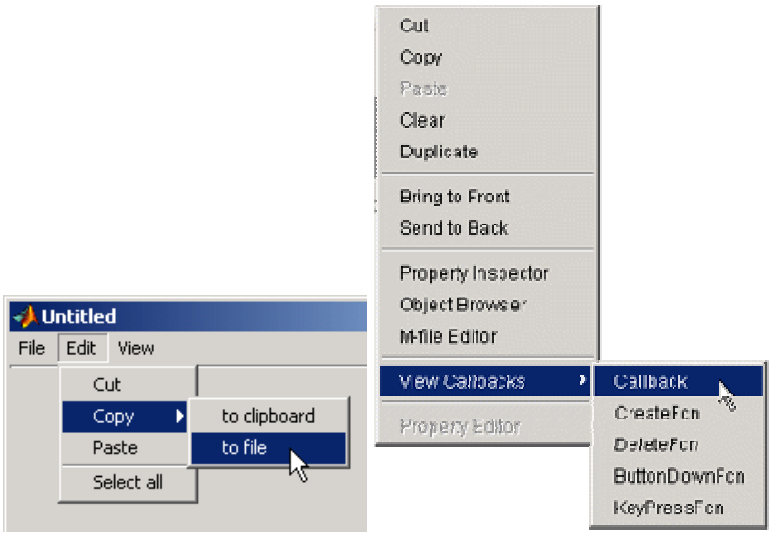
9.3 Interfacce WIMP

Nell'interazione uomo-macchina l'acronimo WIMP denota uno stile di interazione che si basa sugli elementi di un interfaccia grafica.
Gli elementi di una interfaccia **WIMP sono**:

- ✓ ***Windows***
- ✓ ***Icons***
- ✓ ***Menus***
- ✓ ***Pointers***.

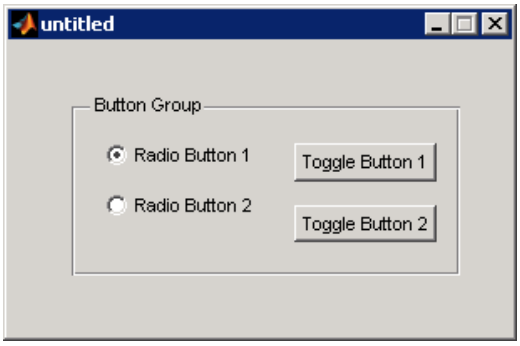
WIMP è lo standard per la maggior parte dei sistemi interattivi, specialmente PC e desktop.

Le Windows sono aree dello schermo che si comportano come se fossero indipendenti, possono contenere testo o grafica, possono essere mosse o ridimensionate possono sovrapporsi (parzialmente o totalmente) o essere tutte contemporaneamente visibili.
Sono elementi costitutivi delle windows le scrollbars che permettono di presentare una quantità di informazione maggiore di quella che entrerebbe nell'area della singola finestra e permettono all'utente di scorrere il contenuto (muovendolo verso l'alto, il basso, o uno dei lati) e il title bars che contiene il nome della finestra.
Le icons sono piccole figure o immagini che rappresentano oggetti nell'interfaccia, e.g., finestre, azioni, stati, opzioni. E' utile ricordare che anche le finestre possono essere ridotte a icone ("iconificate"), permettendo così di mettere in stand-by un task e liberare spazio sullo schermo; ciò e' necessario per poter riprendere il lavoro di un altro task dallo stato nel quale era stato interrotto. In genere le icone possono assumere un aspetto grafico altamente stilizzato o realistico.
I pointers, puntatori, sono componenti importanti del paradigma WIMP. Esso infatti poggia fortemente sull'idea di puntare e di selezionare, attraverso strumenti fisici quali: mouse, joystick, trackball, keyboard shortcuts. Il puntatore può assumere varie forme che spesso servono anche indicare lo stato dell'elaborazione.
I menù sono altri elementi del paradigma che sono utilizzati per effettuare la scelta delle operazioni e dei servizi e le cui opzioni possono anche essere selezionate utilizzando un puntatore. Esistono vari tipi di menù: i Menu Bar che sono in genere disponibili in cima alla finestra e possono essere divisi in "pull-down menu - mouse hold and drag down menu", "drop-down menu - mouse click reveals menu", "fall-down menus - mouse just moves over bar". I contextual menu appaiono invece nel luogo puntato e possono essere pop-up menus che visualizzano azioni che riguardano l'oggetto selezionato (windows pulsante destro) o pie menus, che presentano elementi posizionati in modo circolare.



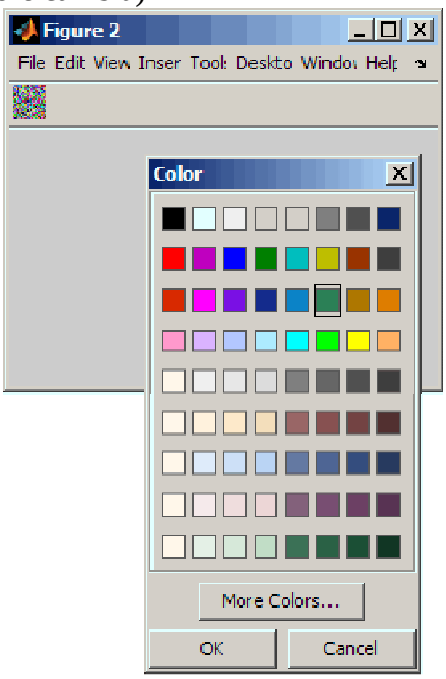
Esempi di menu: a sinistra un menu bar e a destra un contextual memnu

All'interno delle finestre sono poi disponibili i buttons, singole regioni sullo schermo che possono essere selezionate per attivare una specifica azione del software. Esistono alcuni tipi speciali di bottoni quali i radio buttons che permettono scelte mutuamente esclusive e i check boxes che permettono scelte non mutuamente esclusive. Spesso sono poi attive le toolbar, insieme di icone raggruppate che permettono un accesso veloce ad azioni che si effettuano più spesso, quali ad esempio i pulsanti di Microsoft Word e che in genere sono personalizzabili dall'utente.



Esempi di radio bottons

Altri oggetti disponibili in questo tipo di interfacce sono le Palettes: insieme di azioni e/o "modi" raggruppati in piccole finestre (ad esempio nei programmi di grafica) spesso definiscono uno stato nel quale si sta operando (il cursore cambia)



Esempio di palets

I dialog boxes sono finestre informative che appaiono in cima alle altre finestre per segnalare eventi importanti o richiedere informazioni. Ad esempio: errori/warnings, passi di una installazione, salvataggio di un file, etc.

9.4 Windows e controlli come oggetti

Una finestra (WINDOW) è rappresentata da un oggetto con le sue proprietà. (per esempio le sue dimensioni o la sua posizione sullo schermo. Per ottenere il valore corrente della proprietà si può usare la funzione get, ad esempio per la posizione:

get(gcf,'Position')

per posizionare uno o più attributi si usa la funzione set, ad esempio:

set(gcf,'Position',[1 1 400 300])

per indicare la funzione di 'Callback' che deve essere richiamata nel caso che venga selezionato l'oggetto

set(gcf, 'Callback', 'funzione_risposta')

Una finestra o un controllo fa in realtà parte di un albero con un genitore lo schermo e tanti figli. E' sempre possibile conoscere il genitore di una finestra o di un controllo o di un altro oggetto grafico utilizzando la funzione get:

genitore = get(gcf,'Parent')

o conoscere i figli:

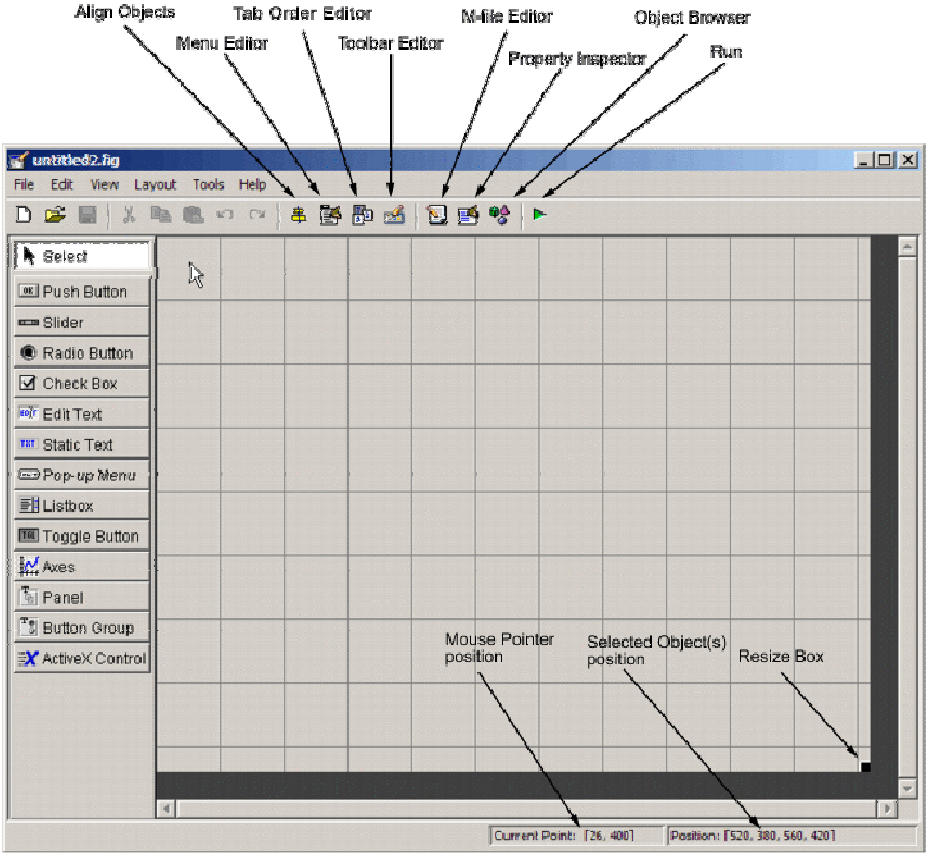
figli = get(gcf,'Children')

Ovviamente in questo modo è possibile esplorare tutto l'albero degli oggetti grafici.
Comandi utili per ritrovare oggetti grafici attivi sono:

gcf = figura grafica corrente
gca = assi correnti

9.5 il programma GUIDE

Il Matlab offre un utile strumento per progettare le GUI il programma ***GUIDE***. Il programma ***GUIDE*** si presenta con una finestra contenente una zona centrale dove progettare la finestra grafica dell' applicazione, un menu e una toolbar, da cui è possibile selezionare i tools disponibili e da una serie di buttons utili per l'inserimento dei controlli nel progetto.



Interfaccia del programma GUIDE

Qui di seguito si riporta in tabella alcuni tools disponibili e una breve

Use This Tool...	To...
Layout Editor	Select components from the component palette, at the left side of the Layo arrange them in the layout area.
Figure Resize Tab	Set the size at which the GUI is initially displayed when you run it.
Menu Editor	Create menus and context, i.e., pop-up, menus.
Align Objects	Align and distribute groups of components. Grids and rulers also enable y components on a grid with an optional snap-to-grid capability.

Tab Order Editor	Set the tab and stacking order of the components in your layout.
Toolbar Editor	Create Toolbars containing predefined and custom push buttons and toggl
Icon Editor	Create and modify icons for tools in a toolbar.
Property Inspector	Set the properties of the components in your layout. It provides a list of al can set and displays their current
Object Browser	Display a hierarchical list of the objects in the GUI.
Run	Save and run the current GUI.
M-File Editor	Display, in your default editor, the M-file associated with the GUI.
Position Readouts	Continuously display the mouse cursor position and the positions of selec

9.6 Struttura del M-FILE creato dal programma GUIDE

Il programma GUIDE crea un M-file associato alla finestra progettata con una specifica struttura contenente alcune funzioni generali attivate all'avvio dell'applicazione o al suo termine e alcune callback collegate ai singoli controlli. Le funzioni generali sono la funzione principale che ha lo steso nome dell'applicazione, la openenig function e la output function:

- ✓ function varargout = Applicazione (varargin)
- ✓ function Applicazione_OpeningFcn(hObject, eventdata, handles, varargin)
- ✓ function varargout = Applicazione_OutputFcn(hObject, eventdata, handles)

Il programma GUIDE crea poi per ogni controllo alcune funzioni specifiche. Il tipo di funzioni disponibili dipende dal controllo. Non tutte le funzioni disponibili sono inserite automaticamente nell'M-file dal programma GUIDE

- ✓ Le due seguenti funzioni sono sempre disponibili:
- ✓ *CreateFcn, DeleteFcn;*
- ✓ la prima è chiamata alla creazione dell'oggetto sullo schermo la seconda al momento che l'oggetto viene cancellato.
- ✓ Altre funzioni disponibili sono:
- ✓ *Callback , ButtonDownFcn, ResizeFcn, KeyPressFcn, KeyReleaseFcn, …*
- ✓ che vengono chiamate in caso di particolari eventi collegati al controllo

I controlli come le finestre sono in realtà degli oggetti che hanno delle proprietà che possono essere lette modificate usando le istruzioni get o set. E' necessario in questo caso disporre di un puntatore all'oggetto che viene fornito utilizzando una struttura denominata "handles" direttamente nella chiamata delle funzioni del programma generato. E' possibile per esempio modificare la visibilità di un controllo, se infatti il suo puntatore è handles.bottone si può usare l'istruzione set nel modo seguente:

set (handles.bottone, 'Visibility', 'on') per rendere visibile il controllo
set (handles.bottone, 'Visibility', 'off') per rendere invisibile il controllo

E' possibile anche aggiungere dinamicamente dei nuovi controlli, usando l'istruzione uicontrol. Qui di seguito si riporta l'elenco dei controlli disponibili con il corrispondente valore per l'attivazione e la sua descrizione. Lo stile è una delle proprietà utilizzate nella funzione **uicontrl** per la creazione del controllo.

UI Control	*Style value*	*Description*
Check Box	'checkbox'	indicates the state of an option or attribute
Editable Text	'edit'	user editable text box
Frame	'frame'	used to visually group controls
Pop-up Menu	'popup'	provides a list of mutually exclusive options
List Box	'listbox'	shows a scrollable list of selections
Push Button	'pushbutton'	pushbutton' invokes an event immediately
Radio Button	'radio'	indicates an option that can be selected
Toggle Button	'toggle'	only two states, "on" or "off"
Slider	'slider'	used to represent a range of values
Static Text	'text'	displays a string of text in a box

Property	Read Only	ValueType/Options	Format
ListBoxTop	No	number	1 element
Max	No	number	1 element
Min	No	number	1 element
Position	No	[left bottom width height]	4-element row
String	No	string	string matrix
Style	No	[{pushbutton} | radiobutton | togglebutton | checkbox | edit | text | slider | frame | popupmenu | list box]	row
SliderStep	No	Number	1 element
TooltipString	No	String	row
Units	No	[inches | centimeters | normalized | points | {pixels}]	row
UIContextMenu	No	handle	1 element
Value	No	number	1 element
Tag	No	string	row
UserData	No	string(s) or number(s)	matrix
Visible	No	[{on} | off]	row

Nella precedente tabella si riporta l'elenco delle principali proprietà che possono essere utilizzate nella funzione **uicontrol** per creare lo specifico controllo. La proprietà style è quella che permette di scegliere il tipo di controllo da visualizzare. Non tutte le proprietà riportate in tabella sono definite

per tutti i tipi di controllo. Ovviamente, il programmatore deve selezionare solo le proprietà utili e definite per lo specifico controllo. E' comunque possibile indicare solo alcune delle proprietà necessarie a definire un controllo usando i valori di default previsti da MATLAB.
Con l'istruzione seguente è possibile creare un bottone con alcune particolari caratteristiche:

uicontrol('style','pushbutton','string','Esempio','Tag','Bottone_di_Esempio','Position',[10 80 50 20], 'Callback',@funzione_callback)

il controllo creato è un bottone, la stringa visualizzata sul controllo è 'Esempio', il nome del controllo è 'Bottone_di_Esempio', e la posizione del vertice superiore del controllo è posta nella posizione 10, 80 a partire dal margine inferiore della finestra. Quando l'operatore preme il bottone attiva la funzione funzione_callback. La funzione_callback deve essere una funzione interna al programma e deve avere due variabili di ingresso:
% hObject handle to figure
% eventdata reserved - to be defined in a future version of MATLAB
hObject contiene un puntatore all'oggetto grafico, mentre eventdata non è utilizzato.

Se si vuole passare un altro parametro alla callback è necessario modificare l'uicontrol nel modo seguente:
uicontrol('style','pushbutton','string','Esempio','Tag','Bottone_di_Esempio','Position',[10 80 50 20], 'Callback', {@funzione_callback,variabile_interna})
Ovviamente, è necessario modificare anche la funzione callback chiamata inserendo il terzo parametro.

È possibile anche creare una interfaccia completa senza usare il programm a GUIDE. Le poche linee di programma qui di seguito riportate ne danno un esempio:

H = figure;
set(H,'Position',[700, 100, 500, 100])
uicontrol('style','pushbutton','string','Esempio','Tag','Bottone_di_Esempio','Position',[10 80 50 20], 'Callback', @funzione_callback)

le tre linee qui riportate permettono di creare una interfaccia creando una figura, modificandone le dimensioni con l'istruzione set e di creare un controllo tipo bottone. La figura seguente mostra il risultato:

Ovviamente, si dovrà creare anche la funzione_callback contenente il codice da eseguire quando l'operatore preme il bottone "Esempio".
Se si vuole passare anche un altro parametro alla funzione possibile usare una costruzione con un cellarray ad esempio {@interna) ,H}. In questo caso il MATLAB crea una copia della variabile H, al momento della dichiarazione della chiamata, che sarà poi passato ad alla callback. Trattandosi di

una copia il valore del parametro non verrà aggiornato se nel programma la variabile H verrà modificata.
Se si vuole che la callback possa accedere dinamicamente a una o più variabili è necessario modificare il programma utilizzando la proprietà 'Userdata' di uno degli oggetti, per esempio la figura principale, dove memorizzare la struttura e allo stesso tempo passare nella chiamata l'handle della finestra a cui si fa riferimento.

```
function Prova
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here
H = figure;
set(H,'Position',[700, 100, 500, 100])
uicontrol('style','pushbutton','string','Esempio','Tag','Bottone_di_Esempio',...
    'Position',[10 80 50 20],'Callback',{@interna,H});
id2 = uicontrol('style','text','string','','Tag','Testo da scrivere',...
    'Position',[10 40 200 20]);
handles.text=id2;
set(H,'Userdata',handles)
end

function interna(primo,secondo,terzo)
handles = get(terzo,'Userdata');
id =handles.text;
set(id,'string','stringa da visualizzare');
end
```