Introduzione a Matlab®

Matlab® è un software per il calcolo numerico che fornisce un ambiente per varie applicazioni scientifiche e ingegneristiche. Matlab® è un software commerciale a pagamento. Maggiori informazioni possono essere reperite all'indirizzo `http://www.mathworks.it/`.

Come alternativa, è possibile utilizzare GNU Octave , un linguaggio ad alto livello, principalmente pensato per il calcolo scientifico. Octave non possiede una interfaccia grafica di default come Matlab® , ma è possibile installarla separatamente. Senza interfaccia, l'inseriremento dei comandi avviene da linea di comando. Octave è essenzialmente compatibile con Matlab® , salvo per alcune piccole differenze. È un software gratuito che è possibile scaricare all'indirizzo `http://www.gnu.org/software/octave/`. Anche se tutti i laboratori possono indifferentemente essere svolti in Matlab® o Octave , per il corso è consigliato l'uso di Matlab® .

1 Matlab® : operazioni su scalari, vettori e matrici

Interfaccia Grafica di Matlab® . È costituita principalmente da quattro ambienti. Nel **workspace** sono rappresentate tutte le varibili memorizzate, il loro valore e tipo. La finestra **current directory** rappresenta una finestra sulla cartella in cui si sta lavorando e mostra tutti i file presenti nella cartella stessa. La **command history** contiene lo storico di tutti i comandi digitati. L'ambiente principale è la **command window** in cui vengono inseriti i comandi. Per quanto riguarda Octave , a meno che non si installi una interfaccia grafica, c'è solo la command window.

Matlab® è l'acronimo di **Mat**rix **Lab**oratory, per cui tutte le variabili in Matlab® sono considerate matrici. In particolare gli scalari sono considerati matrici 1×1, i vettori riga sono matrici $1 \times n$, i vettori colonna sono matrici $n \times 1$, dove n è la lunghezza del vettore. Octave si comporta nello stesso modo.

In questo laboratorio vengono forniti le nozioni necessarie per cominciare ad usare Matlab® (o Octave). Tutti i comandi qui presentati hanno la stessa sintassi sia in Matlab® che in Octave .

Assegnazione di scalari. Cominciamo con l'assegnare il valore 2.45 alla variabile `a`:

```
>> a = 2.45
 a  =
    2.45
```

assegniamo ora il valore 3.1 alla variabile `A`. Osserviamo che Matlab® fa distinzione tra le lettere maiuscole e le lettere minuscole.

```
>> A = 3.1
 A  =
    3.1
```

Le variabili sono sovrascrivibili, cioè se ora assegniamo ad `A` un nuovo valore:

```
>> A = 7.2
 A  =
    7.2
```

il precedente valore 3.1 viene definitivamente perso. Osserviamo che possiamo far seguire un comando da una virgola, senza rilevare nessuna differenza; tale virgola è però necessaria per separare più comandi scritti sulla stessa linea.

```
>> a = 1.2,
 a  =
    1.2
>> a = 1.7, a = 2.45
 a  =
    1.7
 a  =
    2.45
```

Se invece si fa seguire il comando da un punto e virgola, Matlab® non visualizzerà sulla finestra di comando il risultato dell'operazione; il punto e virgola può essere usato per separare due comandi sulla stessa riga.

```
>> a = 1.2;
>> a = 1.7; a = 2.45
 a  =
    2.45
```

Per sapere quali sono le variabili dell'utente attualmente in memoria si utilizza il comando:

```
>> who
 Your variables are:
  A   a
```

Per sapere quali sono le variabili in memoria definite dall'utente è anche possibile utilizzare il comando `whos`. Quest'ultimo, a differenza di `who`, mostra anche la dimensione, l'occupazione di memoria in numero di bytes e il tipo della variabile.

```
>> whos
  Name      Size            Bytes  Class     Attributes

  A         1x1                 8  double
  a         1x1                 8  double
```

Le variabili possono essere cancellate utilizzando il comando `clear`. Possiamo ad esempio cancellare la variabile `A` digitando:

```
>> clear A
```

o tutte le variabili con il comando clear:

```
>> clear
```

Per ripulire la finestra grafica dalle istruzioni precedenti è possibile usare il comando

```
>> clc
```

oppure

```
>> home
```

Per conscere la funzionalita di una istruzione sconosciuta si usa il comando

```
>> help istruzione
```

Digitando soltanto il comando `help`, viene mostrato a schermo un elenco di tutti i pacchetti disponibili in Matlab® .
La seguente tabella riassume le principali funzioni di gestione dell'ambiente e la loro azione:

comando	azione
`clear`	Cancella tutte le variabili
`clear var`	Cancella la variabile `var`
`clc`	Cancella tutte le istruzioni a schermo e blocca la barra di scorrimento
`home`	Cancella tutte le istruzioni a schermo
`help istruzione`	Fornisce le funzionalità e le modalitaà d'uso di `istruzione`
`who`	Elenca le variabili in memoria
`whos`	Elenca le variabili, il loro tipo e la dimensione di memoria occupata

Variabili predefinite. Alcune variabili, proprie di Matlab® , non necessitano di alcuna definizione. Tra queste ricordiamo:

- » `pi`: il numero $\pi = 3.141592653589793\ldots$;
- » `i`: l'unità immaginaria $\sqrt{-1}$;
- » `exp(1)`: il numero di Nepero $e = 2.718281828459046\ldots$

Attenzione! le variabili predefinite possono essere ridefinite, ovvero:

```
>> a = 5 + 2 * i
 a  =
    5.0000 + 2.0000i
>> i = 2
 i  =
    2
>> a = 5 + 2 * i
 a =
    9
```

Istruzione `format`. Permette di modificare il formato di visualizzazione dei risultati ma NON la precisione con cui i calcoli vengono condotti. Il comando ha la sintassi

```
 >> format tipo
 >> 1/7
```

e produce i risultati elencati nella seguente tabella, in base al `tipo` usato

tipo	
rat	1/7
short	0.1429
short e	$1.4286\text{e}-01$
short g	0.142856
long	0.142857142857143
long e	$1.428571428571428\text{e}-01$
long g	0.142857142857143

Il comando `format`, senza ulteriori specifiche, seleziona automaticamente il formato più conveniente per la classe delle variabili in uso. In Octave il comando `format` si usa esattamente nello stesso modo, è pero possibile che fornisca risultati leggermente diversi da quelli di Matlab® .

Assegnazione di vettori. Riportiamo alcuni metodi per l'inserimento di vettori.

- Si possono costruire vettori riga elencando gli elementi separati da uno spazio o da una virgola, come nella tabella seguente

comando	risultato
`» b = [ 1 5 9 4 ]`	b =(1, 5, 9, 4)
`» c = [ 1, 5, 9, 4 ]`	c =(1, 5, 9, 4)

- Per costruire dei vettori riga con elementi equispaziati, il comando generico è

```
>> b = [ primo elemento : passo: ultimo elemento ]
```

  Alcuni esempi sono raccolti nella tabella sottostante:

comando	risultato
`» d = [ 1 :  1 :  4 ]`	d = (1, 2, 3, 4)
`» g = [ 1 :  4 ]`	g = (1, 2, 3, 4)
`» r = [ 1 :  0.1 :  2 ]`	r = (1, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2)
`» s = [ 0 :  -0.3 :  -1 ]`	s = (0, - 0.3, - 0.6, - 0.9)

  Si può utilizzare anche il comando `linspace` per definire un vettore riga di elementi equispaziati. Al comando devono essere forniti come parametri di ingresso i due estremi dell'intervallo e il numero di elementi del vettore.

```
>> p = linspace( primo elemento,  ultimo elemento, numero elementi)
```

  Nell'esempio il vettore `p` è costituito da 6 elementi equispaziati nell'intervallo $[0, 5]$:

```
>> p = linspace( 0, 5, 6)
 p  =
    0    1    2    3    4    5
```

- I vettori colonna vengono costruiti elencando gli elementi separati dal punto e virgola, come nell'esempio:

```
>> q = [ 1; 2; 3; 4 ]
  q  =
    1
    2
    3
    4
```

Assegnazione di matrici. È buone norma utilizzare per il nome delle variabili per le matrici le lettere maiuscole. Combinando i comandi visti per definire i vettori riga e colonna si ottengono immediatamente le istruzioni per definire le matrici, per esempio:

```
>> H = [ 1 2 3 ; 2 4 7 ; 1 4 3 ]
  H  =
    1    2    3
    2    4    7
    1    4    3
```

1.1 Operazioni su vettori e su matrici

- **Trasposizione.** L'operazione di trasposizione si realizza aggiungendo un apice all'istruzione di assegnazione:

  ```
  >> a = [ 1 : 4 ];
  >> b = a'
   b  =
      1
      2
      3
      4
  ```

  Naturalmente applicando l'operazione di trasposizione a tutte le modalità viste per definire dei vettori riga, si ottengono dei vettori colonna.

- **Operazioni algebriche.** Le principali operazioni algebriche tra matrici e vettori sono elencate nella tabella sottostante. Dati

  ```
  >> a = [ 1 3 5 ]; b = [ 4 6 4 ];
  >> H = [ 1 2 3 ; 2 4 7 ; 1 4 3 ]; G =  [ 4 6 3 ; 8 4 1 ; 3 2 9 ];
  ```

operazione	azione
`a + b`	Somma di due vettori della stessa dimensione
`H + G`	Somma di due matrici della stessa dimensione
`a - b`	Sottrazione di due vettori della stessa dimensione
`H - G`	Sottrazione di due matrici della stessa dimensione
`H * G`	Prodotto algebrico righe per colonne. Il risultato è una matrice 3×3
`G * a'`	Prodotto algebrico righe per colonne. Il risultato è un vettore 3×1
`a * G`	Prodotto algebrico righe per colonne. Il risultato è un vettore 1×3
`a' * b`	Prodotto algebrico righe per colonne. Il risultato è una matrice 3×3
`a * b'`	Prodotto algebrico righe per colonne. Il risultato è uno scalare
`3 * G`	Prodotto di uno scalare per una matrice
`3 * b`	Prodotto di uno scalare per un vettore
`cross(a,b)`	Prodotto vettore. `a` e `b` devono necessariamente avere solo tre elementi

Osserviamo che nel prodotto righe per colonne le dimensioni dei vettori o delle matrici utilizzate devono essere compatibili.

- **Operazioni puntate (o elemento per elemento).** In Matlab® sono definite anche delle operazioni particolari che agiscono sulle matrici e sui vettori elemento per elemento.

  – Somma o sottrazione di un valore scalare ad ogni elemento di una matrice (vettore). La matrice (vettore) risultante ha le stesse dimensioni della matrice (vettore) di partenza.

  ```
  >> f = [ 1 2 3 ; 2 4 6 ; 3 6 9 ];
  >> l = f + 3
    l =
      4    5    6
      5    7    9
      6    9    12
  ```

  – Prodotto elemento per elemento di due matrici (vettori). Date due matrici (vettori) il loro prodotto elemento per elemento è una matrice (vettore) delle stesse dimensioni e i cui elementi sono il prodotto degli elementi di posto corrispondente delle matrici (vettori) iniziali. (Attenzione alla compatibilità, in questo caso le due matrici (vettori) devono avere le stesse dimensioni!). Per esempio

  ```
  >> b = [2 5 3 2];
  >> c = [1 2 4 2];
  >> f = b .* c
   f  =
      2  10  12  4
  ```

  mentre se i due vettori non hanno le stesse dimensioni si ottiene un messaggio di errore, come nel seguente esempio

  ```
  >> b = [2 5 3 2];
  >> a = [ 3 5 6];
  >> f = a .* b
  ```

```
??? Error using ==> times
Matrix dimensions must agree
```

– Divisione elemento per elemento di due matrici (vettori). Date due matrici (vettori) la loro divisione elemento per elemento è una matrice (vettore) delle stesse dimensioni e i cui elementi sono il rapporto degli elementi di posto corrispondente delle matrici (vettori) iniziali. Per esempio

```
>> b = [2 5 3 2];
>> c = [1 2 4 2];
>> format rat
>> f = b ./ c
 f  =
    2  5/2  3/4  1
```

– Elevamento a potenza di una matrice (vettore) elemento per elemento. La matrice (vettore) risultante ha le stesse dimensioni della matrice (vettore) di partenza e i suoi elementi sono ottenuti elevando alla potenza richiesta gli elementi della matrice (vettore) di partenza. Per esempio

```
>> a = [1 2 3 4];
>> b = a.^2
  b =
    1    4    9    16
```

Si osservi che tutte le operazioni elemento per elemento sono contraddistite dal "." che precede il simbolo dell'operazione, con l'eccezione dell'operazione di somma e sottrazione

Nella tabella sottostante sono riassunte le operazioni "elemento per elemento". Dati

```
>> a = [ 1 3 5 6 ];
>> b = [ 4 6 4 8 ];
```

`operazione`	azione
`b + 3`	Somma lo scalare 3 a tutti gli elementi di `b`
`a - 3`	Sottrae lo scalare 3 a tutti gli elementi di `a`
`a .* b`	Moltiplica gli elementi di `a` e `b` di posto corrispondente
`a ./ b`	Divide ogni elemento di `a` per il corrispondente elemento di `b`
`a .^ n`	Eleva alla potenza `n` tutti gli elementi di `a`

Funzioni matematiche elementari. Le funzioni matematiche elementari restituiscono matrici o vettori della stessa dimensione della variabile cui è applicata la funzione. Data una matrice `A`:

`funzione`	azione
`abs(A)`	Valore assoluto degli elementi di `A`
`sqrt(A)`	Radice quadrata degli elementi di `A`
`exp(A)`	Funzione esponenziale di ogni elemento di `A`
`log(A)`	Logaritmo naturale di ogni elemento di `A`
`log10(A)`	Logaritmo in base 10 di ogni elemento di `A`
`log2(A)`	Logaritmo in base 2 di ogni elemento di `A`
`sin(A)`	Seno di ogni elemento di `A`
`cos(A)`	Coseno di ogni elemento di `A`
`tan(A)`	Tangente di ogni elemento di `A`
`asin(A)`	Arco seno di ogni elemento di `A` (in radianti)
`acos(A)`	Arco coseno di ogni elemento di `A` (in radianti)
`atan(A)`	Arco tangente di ogni elemento di `A` (in radianti)
`sinh(A)`	Seno iperbolico di ogni elemento di `A`
`cosh(A)`	Coseno iperbolico di ogni elemento di `A`
`tanh(A)`	Tangente iperbolica di ogni elemento di `A`

Osserviamo che per le funzioni trigonometriche i valori di `A` devono sono espressi in radianti.

Esercizio: calcolare le seguenti espressioni matematiche e confrortarne il risultato

- $\frac{e^5+\sin^3(\pi)}{\sqrt{\ln 30}-10} = -18.1973$
- $e^{\log_2 50} + e^{\log_{10} 40} + e^{\ln 30} = 317.5134$

Una volta definiti `x = 4` e `y = 5` calcolare

- $2x\ln(|y|+1) - y\ln(x+2) = 5.3753$
- $\arctan\left(\frac{x}{y}\right) - \sin^2(x\sqrt{|y|}) = 0.4611$

Funzioni per la gestione di vettori e matrici. Nella tabella successiva è riportato un elenco delle piu importanti funzioni Matlab® (e Octave) per la gestione di vettori e matrici, con alcuni esempi di applicazione. Alcuni comandi sono indifferentemente utilizzabili sia per i vettori che per le matrici. In caso contrario è esplicitamente dichiarato. Data una matrice `A` e un vettore `b`:

`funzione`	Azione
`size(A)`	Restituisce un vettore di due elementi, il primo è il numero di righe di `A`, il secondo il numero di colonne di `A`
`size(A,1)`	Restituisce il primo elemento di `size(A)`, cioè il numero di righe di `A`
`size(A,2)`	Restituisce il secondo elemento di `size(A)`, cioè il numero di colonne di `A`
`length(b)`	Restituisce la lunghezza del vettore `b` (in generale equivale a `max(size(b))`)
`max(b)`	Restituisce il piú grande elemento di `b`
`min(b)`	Restituisce il piú piccolo elemento di `b`
`max(A)`	Restituisce un vettore riga contenente il piú grande elemento di ogni colonna di `A`
`min(A)`	Restituisce un vettore riga contenente il piú piccolo elemento di ogni colonna di `A`
`sum(b)`	Restituisce uno scalare pari alla somma di tutti gli elementi di `b`
`sum(A)`	Restituisce un vettore i cui elementi sono la somma degli elementi di colonna di `A`
`diag(A)`	Estrae la diagonale principale di `A`
`diag(A,1)`	Estrae la sopradiagonale di ordine 1 di `A`
`diag(A,-1)`	Estrae la sottodiagonale di ordine 1 di `A`
`diag(A,k)`	Estrae la sopra/sottodiagonale di ordine `k` di `A`; se la dimensione di `A` è `n`, `k` può variare da `-n+1` a `n-1`
`diag(b)`	Costruisce una matrice quadrata diagonale con gli elementi di `b` sulla diagonale principale
`diag(b,k)`	Costruisce una matrice quadrata con gli elementi di `b` sulla sopra/sottodiagonale di ordine `k`.
`tril(A)`	Crea una matrice triangolare inferiore con elementi coincidenti con i corrispondenti elementi di `A`
`triu(A)`	Crea una matrice triangolare superiore con elementi coincidenti con i corrispondenti elementi di `A`
`A(1,1)`	Estrae l'elemento di posto (1,1) di `A`
`A(:,1)`	Estrae la prima colonna di `A`
`A(1,:)`	Estrae la prima riga di `A`
`A(1:3,:)`	Estrae le prime tre righe di `A`
`A([1,3],:)`	Estrae la prima e la terza riga di `A`
`A(1:3,1:3)`	Estrae la sottomatrice costituita dalle prime tre righe e tre colonne di `A`
`A(2:4,[1,3])`	Estrae la sottomatrice costituita dalle righe 2, 3, 4 e dalle colonne 1, 3 di `A`

Merita una descrizione a parte, la funzione `norm`. Essa ha due parametri in input: un vettore `v` e un numero intero `n` oppure `inf`. Se viene passato solo il vettore, calcola la norma 2 (*norma euclidea*) di `v`, definita come $||\mathtt{v}||_2 = \sqrt{\sum_{i=1}^{\mathtt{length(v)}} \mathtt{v}_i^2}$. Se viene passato un numero intero `n`,

calcola la norma n di `v` definita come $||\texttt{v}||_n = \left(\sum_{i=1}^{\texttt{length(v)}} |\texttt{v}_i|^n\right)^{\frac{1}{n}}$. Infine se viene passato come secondo argomento `inf`, calcola la norma infinito di `v` definita come $||\texttt{v}||_\infty = \max_{1\leq i\leq \texttt{length(v)}} |\texttt{v}_i|$.

`operazione`	azione
`norm(v)`	Calcola la norma euclidea (norma 2) di `v`
`norm(v,1)`	Calcola la norma 1 di `v`
`norm(v,2)`	Calcola la norma euclidea (norma 2) di `v`
`norm(v,inf)`	Calcola la norma infinito di `v`

Funzioni per definire vettori o matrici particolari.
Dati `Nrighe` e `Ncolonne` il numero di righe e il numero di colonne della variabile vettoriale che vogliamo costruire, la tabella seguente riassume i comandi per la costruzione di matrici particolari:

`funzione`	azione
`zeros(Nrighe, Ncolonne)`	Costruisce una matrice (vettore) di tutti 0
`zeros(Nrighe)`	Costruisce una matrice quadrata di tutti 0
`ones(Nrighe, Ncolonne)`	Costruisce una matrice (vettore) di tutti 1
`ones(Nrighe)`	Costruisce una matrice quadrata di tutti 1
`eye(Nrighe)`	Costruisce una matrice quadrata con elementi pari ad 1 sulla diagonale (matrice identità)
`rand(Nrighe, Ncolonne)`	Costruisce una matrice (vettore) di elementi casuali compresi tra 0 ed 1
`rand(Nrighe)`	Costruisce una matrice quadrata di elementi casuali compresi tra 0 ed 1

Esercizio: quadrato magico di ordine 4.

$$\texttt{A} = \begin{bmatrix} 16 & 2 & 3 & \boxed{13} \\ 5 & 11 & \boxed{10} & 8 \\ 9 & \boxed{7} & 6 & 12 \\ \boxed{4} & 14 & 15 & 1 \end{bmatrix}$$

a) Inserire la matrice.

- I modo)

```
>> A = [16 2 3 13; 5 11 10 8; 9 7 6 12; 4 14 15 1]
 A =
    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1
```

- II Modo)

```
>> A = [ 16 2 3 13
>>       5 11 10 8
```

```
>>        9 7 6 12
>>        4 14 15 1 ]
 A =
    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1
```

- III Modo) Usare il comando `magic(4)` che costruisce la matrice magica di ordine 4.

```
>> A = magic(4)
 A =
    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1
```

b) Sommare per colonne.

```
>> sum(A,1)
 ans =
   34    34    34    34
```

c) Sommare per righe.

- I Modo)

```
>> sum(A,2)
 ans =
    34
    34
    34
    34
```

- II Modo)

```
>> A = A'
 A =
    16     5     9     4
     2    11     7    14
     3    10     6    15
    13     8    12     1
>> sum(A,1)
 ans =
    34    34    34    34
```

Il comando `A'` costruisce la matrice trasposta della matrice `A`. Osserviamo che il comando `sum(A)` calcola la somma per colonne degli elementi della matrice `A`.

d) Sommare gli elementi della diagonale principale.

- I Modo) Scriviamo la somma degli elementi richiesti. Con il comando `A(n,m)` otteniamo il valore dell'elemento di `A` posizionato sull'`n`-esima riga e sull'`m`-esima colonna.

```
>> A(1,1) + A(2,2) + A(3,3) + A(4,4)
 ans  =
    34
```

oppure, utilizzando il comando `sum` visto per i vettori:

```
>> sum([A(1,1) A(2,2) A(3,3) A(4,4)])
 ans  =
    34
```

- II Modo) Cominciamo con l'estrarre la diagonale principale:

```
>> a = diag(A)
 a  =
    16
    11
     6
     1
```

oppure

```
>> a = diag(A,0)
 a  =
    16
    11
     6
     1
```

Una volta estratta la diagonale, possiamo usare il comando `sum` visto nel caso dei vettori.

```
>> sum(a)
 ans  =
    34
```

L'operazione si può eseguire in una sola riga di comando scrivendo:

```
>> sum(diag(A))
 ans  =
    34
```

f) Calcolare `A*A`, `A^2`, `A.*A` e `A.^2`.

```
>> A * A
 ans =
    345   257   281   273
```

```
    257   313   305   281
    281   305   313   257
    273   281   257   345
>> A^2
 ans =
    345   257   281   273
    257   313   305   281
    281   305   313   257
    273   281   257   345
>> A .* A
 ans =
    256     4     9   169
     25   121   100    64
     81    49    36   144
     16   196   225     1
>> A.^2
 ans =
    256     4     9   169
     25   121   100    64
     81    49    36   144
     16   196   225     1
```

i) Calcolare la radice quadrata degli elementi di `A`:

```
>> sqrt(A)
 ans =
    4.0000    1.4142    1.7321    3.6056
    2.2361    3.3166    3.1623    2.8284
    3.0000    2.6458    2.4495    3.4641
    2.0000    3.7417    3.8730    1.0000
```

oppure

```
>> A.^(1/2)
 ans =
    4.0000    1.4142    1.7321    3.6056
    2.2361    3.3166    3.1623    2.8284
    3.0000    2.6458    2.4495    3.4641
    2.0000    3.7417    3.8730    1.0000
```

2 Definizione di funzioni

Matlab® (e Octave) mettono a disposizione una serie di funzioni matematiche "standard" (`sin`, `cos`, `exp`, ...), ma consentono all'utente anche di definire le proprie funzioni.

In altre parole, è possibile scrivere un frammento di codice che prenda in ingresso una variabile x e restituisca il valore $f(x)$, ad esempio $f(x) = x^4 + 3\log(x)$, e che venga eseguito tramite il comando `nome_funzione(x)` [1].

Vengono forniti quattro diversi modi per definire una funzione:

1. tramite il comando `eval`;
2. tramite il comando `inline`;
3. tramite `anonymous functions`, con il comando `@`;
4. tramite `.m file`;

Per ora ci occupiamo solo dei primi tre casi. Il quarto, che prevede l'utilizzo di un file esterno, verrà illustrato più avanti.

2.1 Il comando `eval`

Supponiamo di voler implementare la funzione che restituisca il valore $f(x) = x^3 + 1$. Un modo di procedere è scrivere le operazioni che devono essere eseguite all'interno di una stringa di caratteri che chiamiamo `cubica` (notare l'uso di `.^`):

```
>> cubica='x.^3-1'
```

poi dichiarare il vettore dei valori di `x` su cui vogliamo valutare $f(x)$:

```
>> x=[0:1:3];
```

ed infine il comando `eval` ci permette di valutare la funzione memorizzata nella stringa `cubica` in corrispondenza dei valori contenuti nel vettore `x`:

```
>> eval(cubica)
ans =
    -1     0     7    26
```

È importante osservare che in questo caso la variabile utilizzata per definire la stringa `cubica` deve necessariamente avere lo stesso nome del vettore di punti che vogliamo valutare, cioè `x`. In caso contrario si otterrà un errore, come nel seguente esempio:

```
>> clear x
>> t=[0:1:3]
t =
     0     1     2     3
>> eval(cubica)
??? Error using ==> eval
Undefined function or variable 'x'.
```

[1] è buona norma di programmazione dare nomi significativi alle funzioni!

2.2 Il comando `inline`

Il secondo metodo per definire una funzione utilizza il comando `inline`. La sintassi è:

```
>> x=[0:1:3];
>> cubica = inline('x.^3-1','x');
>> cubica(x)
ans =
    -1     0     7    26
```

Il comando `inline` quindi prende in ingresso una stringa contenente la definizione della funzione e una stringa contenente il nome della variabile in ingresso a tale funzione. Va notato che `cubica` adesso non è una stringa, come nel caso di `eval`, ma un oggetto di tipo `function`:

```
>> cubica = inline('x.^3-1','x')
cubica =
     Inline function:
     cubica(x) = x.^3-1
```

e quindi sono consentite valutazioni della funzione anche su variabili che non si chiamino `x`:

```
>> t=[0:1:4];
>> cubica(t)
ans =
    -1     0     7    26    63
```

Tramite il comando `inline` è anche facile definire funzioni di più variabili:

```
>> cubicaR2 = inline('x.^3+y.^3','x','y');
>> cubicaR2(2,4)
ans =
    72
```

2.3 `Anonymous functions ( @)`

Questo terzo modo può essere pensato come una "contrazione" del comando `inline` [2] . La sintassi è la seguente:

```
>> cubica = @(x) x.^3-1
cubica =
    @(x) x.^3-1
>> cubica(x)
ans =
    -1     0     7    26
```

[2] In realtà i due meccanismi sono diversi: le `anonymous function` usano il concetto di `function handle @`, che non approfondiremo in questo corso.

Dopo il carattere speciale `@` si indica fra parentesi la variabile in ingresso, e poi si scrive l'operazione che deve essere eseguita. Anche in questo caso è possibile valutare `cubica` su variabili che non si chiamano `x`, ed è facile dichiarare funzioni di più variabili:

```
>> t = [4 5];
>> cubica(t)
ans =
    63   124

>> cubicaR2 = @(x,y) x.^3+y.^3
cubicaR2 =
    @(x,y) x.^3+y.^3
>> cubicaR2(2,4)
ans =
    72
```

3 Disegnare grafici di funzione

3.1 Il comando `plot`

Per disegnare il grafico di una funzione reale di variabile reale si utilizza il comando `plot`.

La sintassi di `plot` prevede che si diano in ingresso al comando:

- due vettori `x` e `y` (delle stesse dimensioni) contenenti le ascisse e le ordinate dei punti del grafico (il grafico viene disegnato unendo tali punti con dei segmenti).
- i comandi opzionali con cui specificare le caratteristiche del grafico (tipo di linea, colore, spessore, colore dello sfondo...)

Nel nostro caso, il comando:

```
>> x=[0:0.01:3];
>> plot(x,cubica(x))
```

produce il grafico in Figura 1.

Un ulteriore comando `plot` disegna il nuovo grafico sulla stessa finestra, cancellando quello precedente. Per modificare questo comportamento, così che i nuovi grafici siano sulla stessa figura, si usa il comando `hold on`. Si torna allo stato precedente con `hold off` mentre `hold` da solo cambia tra i due stati. Ad esempio, i comandi per disegnare sullo stesso grafico la funzione $f(x) = 3 - x^2$ sono:

```
>> parabola = @(x) 3-x.^2;
>> hold on;
>> plot(x,parabola(x))
```

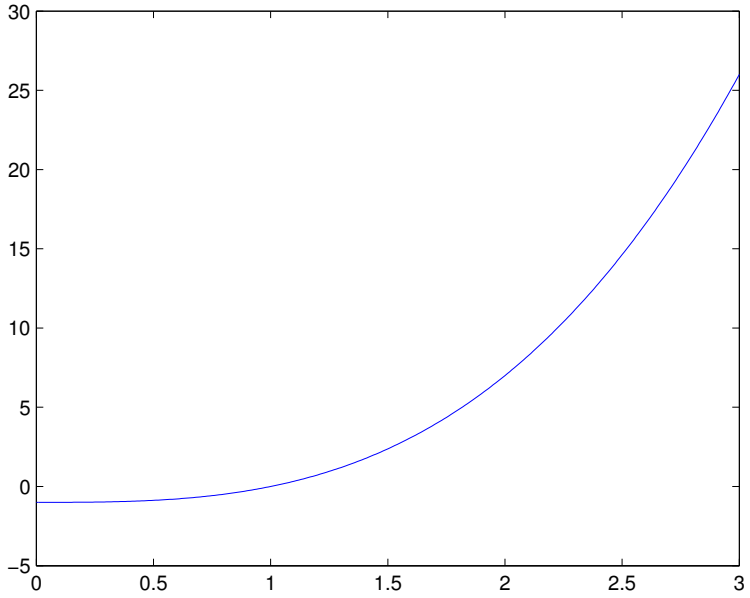

Figura 1: $x^3 - 1$ per $0 \leq x \leq 3$

Per migliorare la leggibilità dei grafici sono utili i comandi di stile grafico: ad esempio, si può far disegnare il secondo grafico in rosso (Figura 2). Per aprire una nuova finestra grafica si usa il comando `figure`.

```
>> figure;
>> plot(x,cubica(x));
>> hold on;
>> plot(x,parabola(x),'r')
```

Digitando `help plot` si ottiene una panoramica sulla formattazione che si può assegnare ad un grafico. Ad esempio questi sono i comandi per definire colore, stile della linea e stile dei punti:

b	blue	.	point	-	solid
g	green	o	circle	:	dotted
r	red	x	x-mark	-.	dashdot
c	cyan	+	plus	--	dashed
m	magenta	*	star	(none)	no line
y	yellow	s	square		
k	black	d	diamond		
		v	triangle (down)		
		^	triangle (up)		
		<	triangle (left)		
		>	triangle (right)		
		p	pentagram		
		h	hexagram		

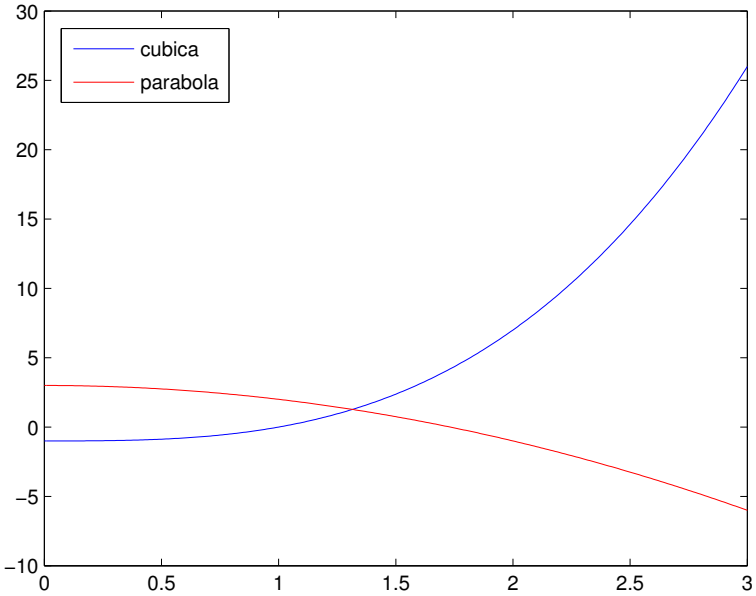

Figura 2: grafico di $x^3 - 1$ e $3 - x^2$ per $0 \leq x \leq 3$

Nota: in Matlab® se si vuole cancellare un grafico dalla finestra mentre `hold on` è attivato, si può entrare in modalità `edit plot` (menù `tools/edit plot`, quindi selezionare il grafico desiderato col mouse e cancellarlo premendo il tasto `canc`).

3.2 Grafici in scala logaritmica

In molte aree scientifiche vengono usati grafici in scala logaritmica/semilogaritmica. Matlab® (e Octave) forniscono a tale proposito i comandi `semilogy`, `semilogx` e `loglog`, che sono l'equivalente di `plot` ma tracciano un grafico rispettivamente con l'asse delle ordinate in scala logaritmica, con l'asse delle ascisse logaritmico ed entrambi in scala logaritmica.

Supponiamo di voler disegnare in scala y-logaritmica sull'intervallo $0 \leq x \leq 10$ il grafico delle funzioni $y = e^x$ e $y = e^{2\,x}$. I comandi necessari sono (stavolta disegnamo il secondo grafico in verde, con linea continua, indicando i punti con degli asterischi, vedi figura 3):

```
>> x=[0:0.1:10];
>> semilogy(x,exp(x))
>> hold on;
>> semilogy(x,exp(2*x),'-*g')
```

I grafici ottenuti sono delle rette, dal momento che stiamo tracciando $\log(y) = \log(e^x) = x$, cioè una retta. La funzione $y = e^{2x}$ risulta essere una retta con pendenza doppia, poiché $\log(e^{2\,x}) = 2\,x$.

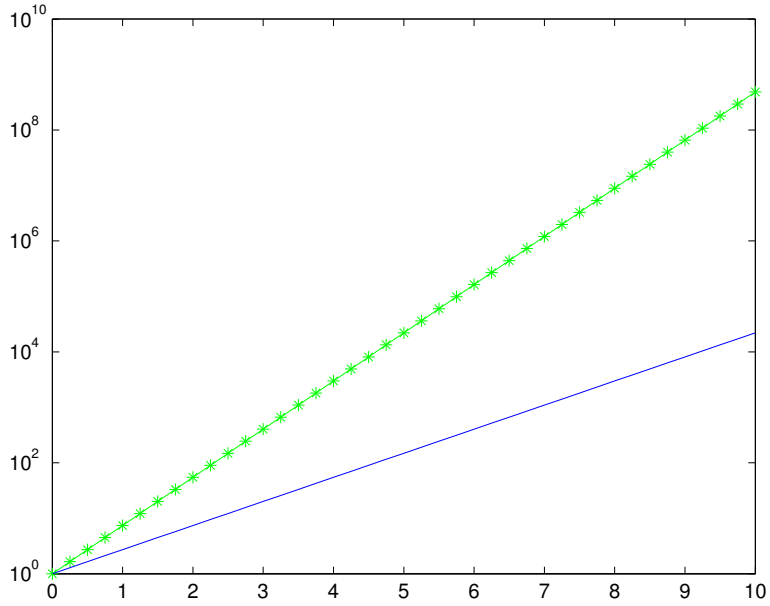

Figura 3: grafico in scala y-logaritmica di $y = e^x$ e $y = e^{2x}$

Possiamo aggiungere molti dettagli al disegno (vedi figura 4):

- la griglia:

  ```
  >> grid on
  ```

  Il comando `grid on` traccia la griglia, `grid off` la rimuove mentre `grid` da solo cambia tra i due stati.

- il titolo del grafico:

  ```
  >> title('Grafico di exp(x) e di exp(2x)')
  ```

- i titoli degli assi:

  ```
  >> xlabel('Scala lineare')
  >> ylabel('Scala logaritmica')
  ```

- la legenda:

  ```
  legend('exp(x)','exp(2*x)','Location','NorthWest')
  ```

  Il comando `legend` attribuisce le stringhe di testo che gli sono passate ai grafici disegnati da plot, nello stesso ordine (prima stringa con il primo grafico, seconda stringa con il secondo grafico etc.). Alcune particolari stringhe di testo, come `'Location'` che abbiamo appena utilizzato, sono interpretate da `legend` come comandi che permettono di modificare aspetto e posizione della legenda. Ad esempio `'Location','NorthWest'` indica di porre la legenda in alto a sinistra (*NordOvest* in una carta geografica con il nord in alto). È possibile anche modificare a mano la legenda utilizzando il mouse nella finestra del grafico (menù `insert`).

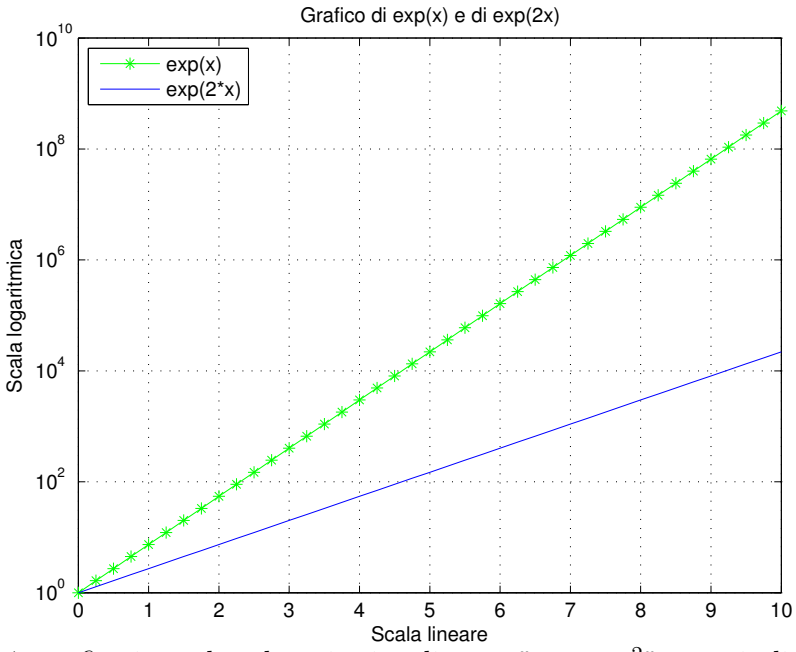

Figura 4: grafico in scala y-logaritmica di $y = e^x$ e $y = e^{2x}$, con titoli e legenda

4 Script

La scrittura di programmi da linea di comando è pratica perché immediata ed interattiva, ma è comoda solo quando il numero di istruzioni è limitato. Nel caso si voglia scrivere codice lungo e soprattutto si voglia avere la possibiltà di riutilizzare in futuro quanto già scritto, è utile salvare i comandi in uno file di *script*. Matlab® (ed Octave) utilizzano dei semplici file di testo non formattato con estensione `.m` per memorizzare le istruzioni; file di questo tipo sono detti *M–files*. A questo scopo apriamo l'*editor* di Matlab® o Octave utilizzando il comando `edit`. Matlab dispone di un proprio editor, mentre Octave può aprire un editor presente sistema oppure anch'esso un editor dedicato, se si utilizza un'interfaccia grafica come *QtOctave* (si rimanda al sito web del corso per i dettagli). L'editor di Matlab o QtOctave può essere aperto anche cliccando sulla apposita icona nella barra in alto. In realtà può essere utilizzato qualsiasi editor che possa salvare testo non formattato: ne esistono tantissimi e molti sanno riconoscere la sintassi di Matlab® e Octave. Dopo aver aperto l'editor, si scrivono i comandi come se fossero nel prompt dei comandi:

Listing 1: Esempio di script

```
x = [0:0.01:1];
f = @(x) sin (x) .* cos (x);
plot (x, f (x), 'b.-');
grid on;
f (0.2)
```

Salviamo il file con il nome `test_plot.m`. Il file verrà salvato nella directory di lavoro corrente, il cui percorso è mostrato nella barra degli strumenti, oppure tramite il comando `pwd`. Tale directory può essere cambiata utilizzando il comando `cd` (si digiti `help cd` per maggiori chiarimenti) oppure tramite l'apposita casella disponibile nell'interfaccia grafica (sia per Ma-

tlab che per QtOctave).
Si possono eseguire i comandi contenuti nel file scrivendo semplicemente il nome del file dalla riga di comando [3] oppure cliccando sul pulsante `Run` situato nella barra degli strumenti dell'editor (identificato dall'icona a forma di triangolo verde).

```
>> test_plot
ans =
    0.1947
```

All'interno dello script è possibile utilizzare variabili già presenti in memoria. Inoltre, le variabili create (o modificate) durante l'esecuzione dell'M–file vengono assegnate nello spazio di lavoro, come se le istruzioni fossero state battute nel prompt dei comandi.

Notiamo infine che è possibile inserire *commenti* all'interno degli script Matlab® utilizzando il simbolo `%`:

Listing 2: Esempio di commenti in Matlab®

```
% un esempio di commento
a = 2;

b = a+1 % un altro commento. I caratteri dopo il '%' vengono ignorati
```

I commenti possono occupare sia un'intera riga sia la parte finale di una riga. In qualunque caso, i caratteri che seguono il simbolo `%` saranno ignorati da Matlab® quando lo script sarà eseguito.

5 Strutture di controllo: cicli ed espressioni logiche

5.1 Ciclo `for`

Nel primo laboratorio è stato introdotto il quadrato magico come esempio di matrice:

$$\mathtt{A} = \begin{bmatrix} 16 & 2 & 3 & \boxed{13} \\ 5 & 11 & \boxed{10} & 8 \\ 9 & \boxed{7} & 6 & 12 \\ \boxed{4} & 14 & 15 & 1 \end{bmatrix}.$$

Tra le varie domande dell'esercizio proposto, era stato chiesto di sommare gli elementi dell'antidiagonale (elementi nei riquadri). Ad esempio è possibile scrivere:

```
>> A( 1, 4 ) + A( 2, 3 ) + A( 3, 2 ) + A( 4, 1 )
 ans  =
    34
```

oppure

```
>> sum( [ A( 1, 4 ), A( 2, 3 ), A( 3, 2 ), A( 4, 1 ) ] )
 ans  =
    34.
```

[3] Il file deve trovarsi nella cartella in cui stiamo lavorando

Le soluzioni proposte sopra non sono ovviamente adatte a calcolare la somma degli elementi sull'antidiagonale nel caso di matrici di grandi dimensioni (si pensi ad esmpio ad una matrice di dimensioni 1000×1000!). Per ovviare a questa limitazione, sarebbe utile disporre di un modo automatico per scandire tutti gli elementi dell'antidiagonale uno per uno. Le operazioni da compiere sono semplici e ripetitive: partendo dal primo elemento dell'antidiagonale:

1. memorizzare l'elemento estratto dalla matrice come elemento di un vettore che contenga l'antidiagonale
2. se l'elemento considerato non è l'ultimo, passare al successivo e tornare al punto (1) *altrimenti* terminare l'esecuzione.

Si tratta di un esempio di *ciclo*, cioè di una serie di istruzioni da ripetere un numero determinato di volte.

```
>> n = size( A, 1 )
 n  =
    4
>> for i = 1 : n
      ad( i ) = A( i, n-i+1 );
>> end
>> ad
 ad  =
    13   10   7   4
>> sum( ad )
 ans  =
    34.
```

L'istruzione `for i = 1 : n` indica l'inizio di un ciclo, da eseguire un numero di volte pari alla lunghezza del vettore `1:n`. Ogni volta la variabile i assumerà un valore differente, prima 1 poi $2, 3, \ldots$ fino a `n`. Le istruzioni da ripetere sono racchiuse tra la riga contente `for` e quella contenente `end`. Si noti che la variabile `i` è disponibile all'interno del ciclo, con un valore che è incrementato ad ogni iterazione. Al termine del ciclo alla variabile `i` resta assegnato il valore pari a `n`, cioè 4. (*Nota:* si ricorda che `i` è una variabile predefinita in Matlab® ed Octave e corrisponde all'unità immaginaria. Con le ultime istruzioni ha però assunto un nuovo valore, dato dal ruolo di iteratore del ciclo. Per riassegnare l'unità immaginaria ad `i` si può utilizzare l'istruzione `clear i` oppure `i = sqrt(-1)`).

Si sarebbe potuta effettuare direttamente la somma all'interno del ciclo, tramite somme parziali:

```
>> s = 0;
>> for i = 1 : n
      s = s + A( i, n-i+1 );
>> end
>> s
 s  =
    34
```

5.2 Ciclo `while`

Non sempre è noto in anticipo il numero di iterazione da compiere nel ciclo: a volte la *condizione di uscita* si determina nel ciclo stesso e non è possibile prevedere a priori quando sia verificata. In questi casi il ciclo `for` non è adatto, ma si può utilizzare il ciclo `while`. Abbiamo visto che la funzione `eps(n)` si comporta diversamente in Octave e Matlab: in particolare in Matlab permette di calcolare l'epsilon macchina rispetto ad un numero qualsiasi `n`, non necessariamente 1. Vogliamo scrivere un programma che riproduca il comportamento di `eps(n)` di Matlab, così che sia possibile scrivere una funzione analoga in Octave (la vedremo dopo aver introdotto le funzioni nel prossimo laboratorio). Nel file `epsilon.m` scriviamo uno script che calcoli l'epsilon macchina di 1000:

Listing 3: Script `eps_script.m`

```
n=1000;
epsilon = 2^floor(log2(n));

while ( (n+epsilon) > n )
    epsilon = epsilon/2;
end

epsilon = epsilon*2
```

Proviamo ad eseguire lo script ed a verificare il risultato con `eps` di Matlab:

```
>> eps_script

epsilon =

   1.1369e-13

>> eps(1000)

ans =

   1.1369e-13
```

Ad ogni iterazione l'espressione `(n+epsilon) > n` è valutata ed il ciclo prosegue solo nel caso sia vera. Si noti che è valutata anche in concomitanza del primo ingresso nel ciclo: nel caso sia già falsa, *non* sarà effettuata alcuna iterazione e l'esecuzione proseguirà con la prima istruzione dopo `end`.

Al posto dell'utilizzo di `while`, a volte si scrive un ciclo `for` con un numero di iterazioni sufficientemente alto e si permette un'uscita anticipata tramite l'istruzione `break`. Si tratta di una pessima prassi di programmazione, infatti in un ciclo deve essere chiaramente visibile quale sia la condizione di uscita, indicata all'inizio o al termine del ciclo stesso (dipende dalle istruzioni utilizzate e dai lunguaggi). Inserire condizioni di uscita alternative tramite `break` sparsi nel codice, specialmente nel caso di codice complesso, significa offuscare il comportamento dell'algoritmo e rendere difficile l'individuazione di eventuali errori.

5.3 Espressioni booleane

Dopo l'esecuzione dello script, l'espressione `(n+epsilon) > n` è sicuramente verificata. Proviamo ad inserirla nel prompt dei comandi:

```
>> (n+epsilon) > n

ans =

     1

>> (n+epsilon) < n

ans =

     0
```

come si vede Matlab ed Octave attribuiscono un risultato numerico 1 ad un'espressione valutata come *vera* e 0 ad una espressione valutata come *falsa*. Questo comportamento, che sembra molto naturale a chi conosca un minimo di logica Booleana e di programamzione, non è per nulla banale nel caso siano coinvolte matrici come vedremo tra poco. Le espressioni booleane sono costruite utilizzando gli operatori `==` (uguaglianza), `~=` (diversità), `>`, `>=`, `<`, `<=` e possono inoltre essere concatenate tramite gli operatori logici `&&` (congiunzione logica), `||` (disgiunzione logica) e `~` (negazione logica).

```
>> 5 == 3

ans =

     0

>> 5 ~= 3

ans =

     1

>> 5 > 3

ans =

     1

>> 5 < 3

ans =

     0
```

```
>> 5 >= 3

ans =

     1

>> 5 <= 3

ans =

     0

>> 5 == 3 && 5 >= 5

ans =

     0

>> 5 == 3 || 5 >= 5

ans =

     1

>> ~(5 == 3)

ans =

     1
```

5.4 Costrutti `if-else`

`while` non è l'unica istruzione di Matlab che permetta una valutazione di una espressione, infatti è possibile eseguire delle istruzioni al verificarsi di determinate condizioni tramite il comando `if`:

```
>> if (5>3)
disp('5 e'' maggiore di 3')
end
5 e' maggiore di 3
>> if (5<3)
disp('5 e'' minore di 3')
end
>> if (5==3)
disp('5 e'' uguale a 3')
end
```

```
>> if (5~=3)
disp('5 e'' diverso da 3')
end
5 e' diverso da 3
```

È possibile anche indicare un'alternativa tramite `else`:

```
if (5<3)
disp('5 e''  minore di 3')
else
disp('5 e'' maggiore  o uguale a 3')
end
5 e' maggiore  o uguale a 3
```

È possibile concatenare più `if` con condizioni alternative tramite `elseif`:

```
>> if (5<3)
disp('5 e'' minore di 3')
elseif (5==3)
disp('5 e'' uguale a 3')
else
disp('5 e'' maggiore di 3')
end
5 e' maggiore di 3
```

5.5 Confronto di matrici

Proviamo a valutare alcune espressioni, il cui risultato è *vero* o *falso* nel caso di matrici:

```
>> A=ones(3)

A =

     1     1     1
     1     1     1
     1     1     1

>> B=2*A

B =

     2     2     2
     2     2     2
     2     2     2

>> B>A

ans =
```

```
     1     1     1
     1     1     1
     1     1     1

>> B<A

ans =

     0     0     0
     0     0     0
     0     0     0

>> B==A

ans =

     0     0     0
     0     0     0
     0     0     0

>> B~=A

ans =

     1     1     1
     1     1     1
     1     1     1

>> B(1,1)=1

B =

     1     2     2
     2     2     2
     2     2     2

>> B>A

ans =

     0     1     1
     1     1     1
     1     1     1
```

Come si vede il risultato è quello di una valutazione elemento per elemento. Come ci si aspetta, una matrice di tutti zeri è interpretata come *falso* ed una composta da soli numeri uno come *vero*. Ma cosa succede con una matrice come l'ultima? Per Matlab ed Octave sono

vere solo le epressioni che non hanno zeri nella matrice di valutazione; ciò sembra ragionevole, ma può portare a risultati inaspettati:

```
>> if(A==B)
disp('A e'' uguale a B')
elseif(A~=B)
disp('A e'' diversa da B')
else
disp('A non e'' ne'' uguale ne'' diversa da B')
end
A non e' ne' uguale ne' diversa da B
```

Per Matlab ed Octave le matrici A e B non sono né uguali né diverse!! Per questo, per confrontare matrici e vettori, è disponibile in Matlab il comando `isequal`, che restituisce un singolo valore di verità invece che un'intera tabella. Ad esempio:

```
>> isequal (A, B)

ans =

     0

>> ~isequal (A, B)

ans =

     1
```

Si noti inoltre che gli operatori booleani `&&` e `||` definiti sopra possono essere utilizzati solo con operandi scalari:

```
>> if ((A<B)||(A==ones(3)))
disp('la matrice A e'' minore di B oppure e'' pari a ones(3)')
end
??? Operands to the || and && operators must be convertible to logical scalar values.

>> a = 1;
>> b = 2;
>> if ((a<b)||(b==0))
disp('a e'' minore di b oppure b e'' nullo')
end
a e' minore di b oppure b e' nullo
```

Esistono anche i corrispondenti operatori elemento per elemento: `&` e `|`

```
>> (A<B)|(A==ones(3))

ans =
```

```
     1     1     1
     1     1     1
     1     1     1
```

6 Definizione di funzioni tramite `.m` files

Come abbiamo visto in precedenza, in Matlab® è possibile eseguire degli script chiamando nel prompt dei comandi il nome di un `.m` file presente nella cartella di lavoro. Ad esempio, se ho raccolto una serie di comandi nel file `miofile.m`, digitando nel prompt l'istruzione

```
>> miofile
```

Matlab® eseguirà fedelmente tutti i comandi presenti nel file. Le caratteristiche degli script sono quelle di non avere parametri in ingresso modificabili e di poter lavorare solo su variabili globali presenti in memoria.

Una funzione Matlab® risponde esattamente a queste limitazioni, ossia è uno script al quale è possibile passare parametri in ingresso ed ottenerne in uscita. Inoltre le variabili utilizzate durante l'esecuzione della funzione vengono automaticamente cancellate dalla memoria al termine della stessa. A differenza di uno script standard la sintassi di una funzione richiede che la prima riga del file abbia la struttura

```
function [Output1,Output2,...] = nomefunzione(Input1,Input2,...)
```

dove le parentesi quadre possono essere omesse se l'output è uno solo. È buona norma inserire alcune righe di commento (ovvero precedute dal carattere `%`) dopo la definizione, che possono essere visualizzate tramite il comando

```
>> help nomefunzione
```

Successivamente saranno inserite tutte le istruzioni necessarie all'esecuzione della funzione. Il file dovrà essere salvato con l'estensione `.m` e dovrà essere visibile a Matlab® nell'area di lavoro (ovvero deve essere posizionato nella cartella corrente).

Nel seguente file, che chiameremo `det2.m`, si implementa una funzione che calcola il determinante di una matrice quadrata di dimensione 2:

Listing 4: Script `det2.m`

```
function [det]=det2(A)
%DET2 calcola il determinante di una matrice quadrata di ordine 2
[n,m]=size(A);
if n==m
    if n==2
        det=A(1,1)*A(2,2)-A(2,1)*A(1,2);
    else
        error('Solo matrici 2x2');
    end
else
    error('Solo matrici quadrate');
```

```
end
return
```

dove si utilizza la funzione `error('...')` per interrompere l'esecuzione del programma e visualizzare una stringa di errore nel prompt dei comandi.

Funzioni più flessibili. Una funzione Matlab® può modificare il suo comportamento a seconda del numero di variabili in input o output che riceve: per questo scopo, le funzioni `nargin` e `nargout` contano rispettivamente il numero di parametri di input (output) che sono stati passati alla funzione. Ad esempio, la semplice funzione:

```
function c = test_nargin(a, b)
if (nargin == 1)
    c = a .^ 2;
elseif (nargin == 2)
    c = a + b;
end
```

modifica il suo comportamento a seconda che le venga passato un solo parametro in input (ne calcola il quadrato) o due (ne calcola la somma). Per differenziare il comportamento della funzione è standard l'uso della struttura `if() ... elseif()... end`. Ad esempio, la semplice funzione:

```
function [out1, out2] = test_nargout(a, b)
if (nargout == 1)
    out1 = a * b;
elseif (nargout == 2)
    out1 = a + b;
    out2 = a - b;
end
```

differenzia il suo comportamento a seconda del numero di parametri di output che richiediamo. Infatti:

```
>> a = test_nargout(3,4)
a =
    12
>> [b,c] = test_nargout(3,4)
b =
     7
c =
    -1
```

7 Output su monitor

Come già accennato in precedenza, la funzione `disp` permette di visualizzare una stringa di caratteri (o, più in generale, una matrice di stringhe), racchiusa tra apici. Ad esempio potremmo scrivere:

```
>> disp('Ciao a tutti')
Ciao a tutti
>> disp('Oggi e'' martedi'' ')   % doppio apice per visualizzarne uno
Oggi e' martedi'
```

In molte circostanze si ha la necessità di rappresentare come stringa valori di varibili numeriche. La prima possibilità è quella di convertire una variabile numerica in stringa tramite la funzione `num2string`:

```
>> x = 10;
>> stringa = ['La variabile x vale : ', num2str(x)];
>> disp(stringa)
La variabile x vale : 10
```

mentre un modo molto più versatile (e consigliato) consiste nell'utilizzare le funzioni `fprintf` e `sprintf`. La sintassi della prima è:

```
fprintf('Formato' , Variabili)
```

e scrive su video il valore delle `Variabili` indicate, utilizzando il `Formato` definito, che altro non è che una stringa che contiene i caratteri che si vogliono visualizzare e, nelle posizioni in cui si vuole venga inserito il valore delle `Variabili`, deve essere indicato uno dei formati di visualizzazione preceduti dal carattere `%` . I formati di visualizzazione sono riassunti nella seguente tabella:

Codice di formato	Azione
`%s`	stringa
`%d`	numero intero
`%f`	numero in virgola fissa (es. 1234.5678)
`%e`	numero notazione scientifica (es. $1.2345678e+003$)
`%g`	numero in notazione compatta (sceglie tra `%f` e `%e`)
`\n`	inserisce carattere ritorno a capo
`\t`	inserisce carattere di tabulazione

Ad esempio:

```
>> x = 10; y = 5.5;
>> fprintf('x vale %d mentre y vale %f \n' , x, y);
x vale 10 mentre y vale 5.500000
```