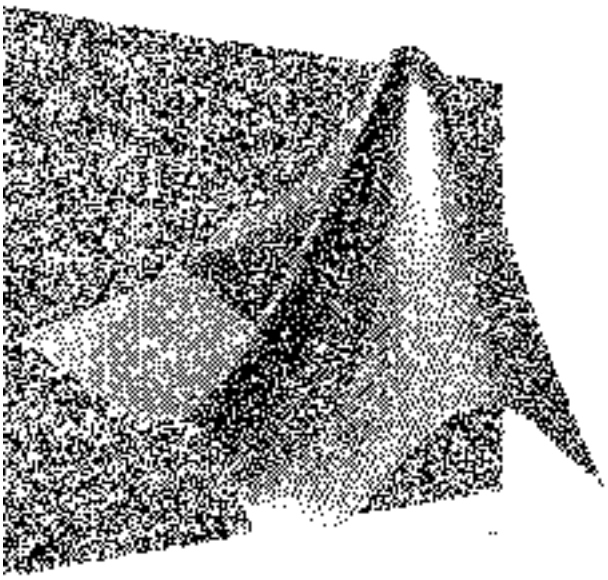
Introduzione all'uso di

MATLAB®

The Language of Technical Computing



Francesco Odetti
DIPTEM
Università di Genova
A.a. 2004/05 (ristampa dell'edizione 2003/04)

MatLab è un programma studiato apposta per operare su matrici. Il nome è infatti un'abbreviazione di **Matrix Laboratory**. **Le variabili sono infatti matrici** e le funzioni di base sono operazioni su matrici. Istruzioni più avanzate consentono tracciamento di grafici in 2D e in 3D, elaborazione di dati per il trattamento di segnali etc.

In queste pagine di introduzione elementare illustriamo alcune delle istruzioni più semplici.

INPUT DI MATRICI

Quando il programma attende un'istruzione, mostra un "prompt" del tipo

```
»
```

Per ottenere la matrice $\mathbf{a} = \begin{pmatrix} 5 & 0 & 2 \\ 1 & 4 & -1 \\ -2 & 2 & 3 \end{pmatrix}$ si scrive:

```
»a=[5,0,2 ; 1,4,-1 ; -2,2,3]
```

e si preme [RETURN]. Le virgole servono per separare gli elementi di una riga, il simbolo **;** per separare le righe della matrice.
Si ottiene:

```
a =
     5     0     2
     1     4    -1
    -2     2     3
```

In questo modo si è assegnato alla matrice il nome **a** e la matrice può essere riutilizzata in seguito.
In luogo delle virgole si possono usare, come normalmente conviene, gli spazi, quindi per ottenere la matrice **a** si può anche scrivere

```
»a=[5 0 2 ; 1 4 -1 ; -2 2 3]
```

ALCUNE OPERAZIONI ELEMENTARI SULLE MATRICI

Trasposta di una matrice:

```
»a'
```

Somma di matrici (errore se **a** e **b** non hanno lo stesso formato):

```
»a+b
```

Prodotto tra matrice e scalare: (**k** è lo scalare, **a** è la matrice)

```
»k*a
```

Prodotto di matrici: (errore se numero colonne (**a**) ≠ numero righe (**b**))

```
»a*b
```

Potenza **n**–ma di una matrice (errore se non è quadrata) :

```
»a^n
```

Inversa di una matrice (errore se non è quadrata): è ottenuta con l'algoritmo gaussiano (anche se non si vede).

```
»inv(a)
```

Determinante di un matrice (errore se non è quadrata): (sempre mediante l'algoritmo gaussiano)

```
»d=det(a)
```

Risultato:

```
d =
    90
```

In questo modo si assegna allo scalare **d** il valore **det(a)** per uso successivo.
Caratteristica di una matrice: (ancora con l'algoritmo gaussiano)

```
»rank(a)
```

Risultato:

```
ans =
    3
```

(dato che non si è assegnato un nome al numero **rank(a)**, il programma assegna d'ufficio il nome **ans** (answer), così il risultato non va perso fino al prossimo calcolo.)

EDITING DI MATRICI

Per cambiare l'elemento (1,2) di **a** da 0 a 7

```
»a(1,2)=7
```

Si ottiene:

```
a =
     5     7     2
     1     4    -1
    -2     2     3
```

INPUT DI MATRICI SPARSE

Per ottenere la matrice diagonale **4 x 4 s** che ha sulla diagonale 1,2,3,4 si può operare così:

```
»s1=[1 2 3 4];
»s=diag(s1)
```

Il simbolo **;** dopo la prima istruzione evita che il programma scriva su schermo la matrice riga **s1.**

Per ottenere la matrice a bocchi **b** si può operare così:

```
»b=[2 3 ; 7 8]
```

$$\mathbf{b} = \begin{pmatrix} 2 & 3 & 0 \\ 7 & 8 & 0 \\ 0 & 0 & 9 \end{pmatrix}$$

Si ottiene:

```
b =
     2     3
     7     8
```

Poi si pone:

```
»b(3,3)=9
```

Si ottiene:

```
b =
     2     3     0
     7     8     0
     0     0     9
```

Il formato di **b** è cambiato e gli elementi non definiti sono posti automaticamente uguali a zero.

La matrice identica **n x n** si ottiene con:

```
»eye(n)
```

Il curioso nome deriva dal fatto che in inglese la lettera I (che rappresenta la matrice identica) e la parola "eye" si pronunciano allo stesso modo, ma **eye(n)** è sicuramente più riconoscibile di **I(n).**

La matrice nulla quadrata **n x n** si ottiene con:

```
»zeros(n)
```

La matrice nulla rettangolare **m x n** si ottiene con:

```
»zeros(m,n)
```

Per ottenere la matrice a blocchi **c** si può quindi per esempio operare così:

```
»c1=[2 3 ; 7 8];
»c2=[9 4 5 ; 6 1 3 ; 3 3 3];
»c=[c1 zeros(2,3) ; zeros(3,2) c2]
```

$$\mathbf{c} = \begin{pmatrix} 2 & 3 & 0 & 0 & 0 \\ 7 & 8 & 0 & 0 & 0 \\ 0 & 0 & 9 & 4 & 5 \\ 0 & 0 & 6 & 1 & 3 \\ 0 & 0 & 3 & 3 & 3 \end{pmatrix}$$

ESTRAZIONE DI RIGHE, COLONNE E DIAGONALI

La matrice riga **r** costituita dalla seconda riga di **a** si ottiene con

```
»r=a(2,:)
```

Risultato:

```
r =
     1     4    -1
```

La matrice colonna **m** costituita dalla terza colonna di **a** si ottiene con

```
»m=a(:,3)
```

La funzione **diag** applicata a una matrice e non a un vettore fornisce la matrice colonna contenente la diagonale della matrice data

```
»diag(a)
ans =
     5
     4
     3
```

Quindi la funzione **diag(diag(a))** applicata a una matrice fornisce la matrice diagonale con diagonale identica a quella di **a** .

```
»diag(diag(a))
ans =
     5     0     0
     0     4     0
     0     0     3
```

Analogamente le funzioni **tril(a) triu(a)** forniscono una matrice rispettivamente triangolare inferiore (lower triangular) e triangolare superiore (upper triangular) partendo da **a** . Per esempio

```
»tril(a)
ans =
     5     0     0
     1     4     0
    -2     2     3
```

La funzione **tril(triu(a))** è perfettamente equivalente a **diag(diag(a))**.

OPERAZIONI ELEMENTARI SULLE RIGHE

L'operazione $R_3 \rightarrow 5\,R_3$ su **a** si può eseguire con il comando seguente:

```
»a(3,:)=5*a(3,:)
```

L'operazione $R_3 \rightarrow R_3 + 5\,R_2$ su **a** si può eseguire con il comando seguente:

```
»a(3,:)=a(3,:)+5*a(2,:)
```

L'operazione $R_1 \leftrightarrow R_3$ su **a** è un po' più complicata e si può ottenere in uno dei seguenti due modi:

```
»y=a(3,:) ; a(3,:)=a(1,:) ; a(1,:)=y
```

dove si usa la variabile temporanea **y** . In questo caso sono state date tre istruzioni in una sola riga, separate con il simbolo **;** che impedisce che vengano scritti su schermo i risultati delle prime due istruzioni.
Notare che, scrivendo semplicemente

```
»a(3,:)=a(1,:) ; a(1,:)=a(3,:)
```

senza usare la variabile ausiliaria **y** , andrebbe persa la prima riga di **a** .
L'altro modo è quello di scrivere:

```
»a([1 3],:)=a([3 1],:)
```

(notare gli spazi tra **1** e **3** e tra **3** e **1**) . La spiegazione dettagliata di questo tipo di comando viene data nel paragrafo seguente.

MANIPOLAZIONE DI MATRICI - IL COMANDO :

Come già visto, il simbolo **:** in **MatLab** è fondamentale per la manipolazione delle matrici.
Il manuale stesso recita: "once you master **:** , you master **MatLab**" (una volta appreso l'uso del simbolo **:** (colon in inglese) avrete appreso **MatLab**).
Diamo alcuni esempi dei principali usi del simbolo **:** .
— Generazione di vettore che sia una progressione aritmetica di passo **1**:

```
»x=1:5
x =
     1     2     3     4     5
```

— Generazione di vettore che sia una progressione aritmetica di passo **0.3**:

```
»x=1:0.3:2
x =
    1.0000    1.3000    1.6000    1.9000
```

Per avere una progressione aritmetica di passo negativo occorre precisare il passo negativo, altrimenti si ottiene la progressione vuota:

```
»x=3:0
x =
   Empty matrix: 1-by-0
»x=3:-1:0
x =
     3     2     1     0
```

Come già visto, mediante il simbolo **:** si possono estrarre righe e colonne delle matrici e anche riordinarle.
Illustriamo alcuni tipici esempi dell'uso di **:** adoperando la curiosa matrice **m** di ordine 4 che si ottiene con la funzione **magic** . La matrice è un quadrato magico (la somma di ogni riga, di ogni colonna e di ogni diagonale è sempre 34)

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

Estrazione della sottomatrice costituita da R_1, R_3, C_3, C_4 .

```
»m([1 3], [3 4])
ans =
     3    13
     6    12
```

Estrazione della sottomatrice costituita da R_1, R_3, C_3, C_4 , ma scambiando le due colonne:

```
»m([1 3], [4 3])
ans =
    13     3
    12     6
```

Estrazione della sottomatrice costituita da tutte le righe e da C_3, C_4 , scambiate:

```
»m(:,[4,3])
ans =
    13     3
     8    10
    12     6
     1    15
```

Estrazione della sottomatrice costituita dalla quarta riga e da tutte le colonne da C_1 a C_3 :

```
»m(4,1:3)
ans =
     4    14    15
```

— Vediamo ancora un modo di costruire le matrici a blocchi.
La matrice **5 x 5** a blocchi **c** definita sopra si può ottenere anche così:

```
»c=[2 3 ; 7 8];
»c(3:5,3:5)=[9 4 5 ; 6 1 3 ; 3 3 3];
```

— Flattening della matrice
Consiste nel trasformare una matrice in un vettore colonna costituito da tutti gli elementi della matrice (nel nostro esempio 16 elementi) letti per colonne.

```
»m(:)
ans =
     16
      5
      9
      4
      2
     11
......... etc ......
```

Aggiungiamo per inciso che il comando

```
»x(k)
```

applicato a un vettore, cioè a una matrice **1 x n** o **n x 1** , fornisce il **k**-mo elemento di **x** .
Lo stesso comando applicato a una matrice qualunque fornisce il **k**-mo elemento del flattening della matrice.
Per esempio

```
»m(6)
ans =
     11
```

fornisce il sesto elemento della matrice **m** letta per colonne.

ALGORITMO DI GAUSS

Se **a** è una matrice quadrata invertibile e **b** una matrice colonna con tante righe quante sono le colonne di **a**, il sistema lineare **ax = b** ha un'unica soluzione. **MatLab** è in grado di determinare la soluzione mediante l'algoritmo gaussiano facendo uso della pivotizzazione parziale, cioè scegliendo per ogni colonna il pivot con valore assoluto più alto. Il comando è il seguente

```
»x=a\b
```

Osservare che il simbolo dell'operazione è **** (backslash) e non **/** (slash).
Formalmente il comando equivale al seguente

```
»x=inv(a)*b
```

Ma, come è noto, il tempo di calcolo richiesto dall'algoritmo gaussiano è circa un terzo di quello richiesto dal calcolo dell'inversa. L'inversa di **a** quindi non viene calcolata. In effetti lo stesso manuale di **MatLab** fa osservare che la funzione **inv(a)** è in pratica di uso assai raro.

> Osserviamo per inciso che il comando **a\b** ha un significato assai più ampio e lo si può usare anche se **a** è una matrice qualunque (anche non quadrata e di qualunque caratteristica, purché **a** e **b** abbiano lo stesso numero di righe). In questo caso viene trovata la cosiddetta soluzione ai minimi quadrati.
>
> In poche parole, dato che non esiste o non è unica una **x** tale che **ax - b = 0** , viene trovata la **x** (di minimo modulo se ce n'è più di una) tale che la norma del vettore **ax - b** sia minima.

Se è necessario studiare ed eventualmente risolvere un sistema qualunque, occorre ridurre totalmente la matrice completa del sistema.

La funzione **rref** (reduced row echelon form) calcola una matrice **r** totalmente ridotta ed equivalente per righe alla matrice **a** .

```
»r=rref(a)
```

La matrice ridotta è determinata mediante la pivotizzazione parziale e l'algoritmo di Gauss-Jordan che è una variante di quello di Gauss.
Nell'algoritmo di Gauss-Jordan, in ogni colonna viene cercato il pivot col valore assoluto più alto, la riga viene portata al primo posto utile e viene divisa immediatamente per il pivot. Quindi tutta la colonna del pivot viene annullata con operazioni elementari (anche nelle righe sopra il pivot, diversamente dall'algoritmo di Gauss).
C'è anche, tra le demo, un'interessante funzione a scopo didattico

```
»r=rrefmovie(a)
```

che consente di visualizzare passo-passo la riduzione totale di **a** .

FATTORIZZAZIONE LU

Un altro modo di realizzare l'algoritmo di Gauss per un sistema lineare **ax = b** con **a** matrice invertibile è quello di passare attraverso la fattorizzazione *LU* .
La fattorizzazione *LU* di una matrice quadrata invertibile **a ,** ha come risultato due matrici. La sintassi per ottenere due output deve specificare il nome dei due risultati:

```
»[l,u]=lu(a)
```

Con questa funzione si ottengono due matrici: **l** invertibile ed "essenzialmente triangolare inferiore" e **u** triangolare superiore tali che **a=l*u** . In pratica **u** è la matrice triangolare superiore ottenuta da **a** mediante l'algoritmo di Gauss con pivotizzazione parziale, mentre **l** è "a meno di una permutazione delle righe" triangolare inferiore con tutti numeri 1 sulla diagonale.
Per esempio

```
»a =[1 -3 -1 ; 1 0 2 ; -4 2 1];
»[l,u]=lu(a)
l =
   -0.2500    1.0000         0
   -0.2500   -0.2000    1.0000
    1.0000         0         0
u =
   -4.0000    2.0000    1.0000
         0   -2.5000   -0.7500
         0         0    2.1000
```

La matrice **u** è triangolare superiore e ha sulla diagonale i pivot di **a** , mentre la matrice **l** risulta triangolare inferiore con diagonale di 1 purché vengano permutate le righe.
Per risolvere un sistema lineare **ax = b** si risolve il sistema ridotto equivalente **ux = c** dove la matrice **c** dei termini noti del sistema ridotto è la soluzione del sistema lineare "essenzialmente ridotto" **ly = b** . Ciò si ottiene eseguendo i due comandi

```
»c=l\b;
»x=u\c
```

Il tempo complessivo richiesto dai tre comandi è essenzialmente quello richiesto dal comando

```
»x=a\b
```

Il vantaggio della fattorizzazione *LU* è il seguente:
Se occorre risolvere due o più sistemi lineari aventi tutti la stessa **a** come matrice dei coefficienti, conviene calcolare una sola volta la sua fattorizzazione *LU* e quindi risolvere i sistemi lineari con i due comandi precedenti. In questo modo la riduzione gaussiana di **a** viene eseguita una sola volta con evidente risparmio di tempo, specialmente per matrici molto grandi.

È anche possibile ottenere la tradizionale fattorizzazione *PA = LU* in cui la matrice **l** è proprio triangolare inferiore con diagonale di 1 e la matrice **p** è di permutazione con la seguente funzione a tre output

```
»[l,u,p]=lu(a)
```

OPERAZIONI ALGEBRICHE SULLE MATRICI

Ricapitoliamo ora le operazioni algebriche tra matrici nelle loro varie forme.
— Somma tra matrici: è la somma usuale tra matrici dello stesso formato.

```
»a+b
```

— Somma tra una matrice e uno scalare: addiziona ad ogni elemento di **a** lo scalare. Per esempio:

```
»a+2
```

Attenzione ! Per matrici quadrate non è equivalente a **a+kI** .
— Prodotto tra matrici: è il prodotto usuale tra matrici (occorre che numero colonne (**a**) = numero righe (**b**)).

```
»a*b
```

— Prodotto tra una matrice e uno scalare: è il prodotto usuale tra matrice e scalare. Per esempio:

```
»a*2
```

— Prodotto puntuale **.*** : è il prodotto elemento per elemento tra matrici dello stesso formato.

```
»a.*b
```

— Divisione a sinistra: è logicamente equivalente a **inv(a)*b** , ma è ottenuta con l'algoritmo gaussiano.

```
»a\b
```

— Divisione a destra: equivalente logicamente a **b*inv(a)**, quindi eseguibile se i formati lo consentono, ma ottenuta anch'essa con l'algoritmo gaussiano. In effetti è ottenuta calcolando **(a'\b')'**

```
»b/a
```

Prestare attenzione al fatto che i due simboli **** e **/** determinano la soluzione ai minimi quadrati nei casi non standard.

— Divisione puntuale a destra **./** : è la divisione elemento per elemento tra matrici dello stesso formato. Attenzione: **a** fornisce i numeratori, **b** i denominatori.

```
»a./b
```

— Divisione puntuale a sinistra **.** : è la divisione elemento per elemento di matrici dello stesso formato. È perfettamente equivalente alla precedente, ma attenzione: anche qui **a** fornisce i numeratori, **b** i denominatori.

```
»b.\a
```

— La potenza **n** -esima (**n** numero intero) di una matrice quadrata è il prodotto della matrice **a** per sé stessa **n** volte.

```
»a^n
```

Questa funzione può essere calcolata anche se **n** non è un numero intero, in tal caso il significato è assai più complesso e l'algoritmo più lungo (cfr. più avanti la funzione **expm(a)**).

— La potenza puntuale **.^** consiste nell'elevare alla **n** (numero reale) ogni elemento di **a** (matrice qualunque)

```
»a.^n
```

— Formato di una matrice: è spesso necessario conoscere il formato di una variabile definita (ricordiamo che tutte le variabili in **MatLab** sono matrici). La funzione è

```
»size(a)
```

che restituisce il formato di **a** , cioè una matrice 1x2 con il numero di righe e il numero di colonne di **a** .
Se si vuole solo il numero di righe di o il numero di colonne della matrice **a** , le funzioni rispettive sono

```
»size(a,1)
»size(a,2)
```

FUNZIONI ELEMENTARI SULLE MATRICI

Sono definite le principali funzioni elementari reali (e complesse) che consistono nell'eseguire la funzione elementare su ogni elemento della matrice.
Le principali funzioni applicabili a una matrice **x** di qualunque formato sono:
— Valore assoluto (modulo se l'elemento è complesso).

```
»abs(x)
```

— Ci sono quattro funzioni per la parte intera:

- Il numero intero più prossimo a **x** :

```
»round(x)
```

- Il numero intero più prossimo a **x** in direzione dello 0, in pratica togliendo i decimali:

```
»fix(x)
```

- La parte intera tradizionale (il più grande numero intero minore o uguale a **x**):

```
»floor(x)
```

- Il più piccolo numero intero maggiore o uguale a **x** :

```
»ceil(x)
```

— Radice quadrata (se l'elemento è negativo o complesso fornisce una delle radici quadrate complesse, come vedremo in seguito). L'espressione **sqrt** è l'abbrevazione di "square root" (radice quadrata)

```
»sqrt(x)
```

— Esponenziale (funziona anche se l'elemento è complesso mediante l'uso della formula di Eulero, come vedremo in seguito).

```
»exp(x)
```

— Logaritmo naturale in base **e** (funziona anche se l'elemento **x** è negativo o complesso come vedremo in seguito).

```
»log(x)
```

— Le tre funzioni trigonometriche (funzionano anche se l'elemento è complesso).

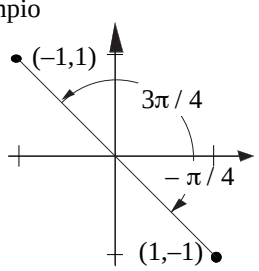
```
»sin(x)
»cos(x)
»tan(x)
```

— Ci sono anche varie altre funzioni trigonometriche tra cui arcoseno, arcocoseno e arcotangente, rispettivamente **asin** , **acos** , **atan** .
Per quanto riguarda quest'ultima, ricordiamo che la funzione arcotangente semplice **atan(x)** restituisce il numero θ con $-\pi/2 < \theta < \pi/2$ tale che tang(θ) (θ in radianti) sia **x** , per cui esiste anche la funzione **atan2** (arcotangente a due variabili) che tiene conto del quadrante nel piano cartesiano.

Precisamente, **atan2(y,x)** è l' unico numero θ con $-\pi < \theta \le \pi$. tale che $\begin{cases}\cos(\theta)=x/\sqrt{x^2+y^2}\\ \sin(\theta)=y/\sqrt{x^2+y^2}\end{cases}$

Geometricamente **atan2(y,x)** è l'angolo orientato θ in radianti tra l'asse positivo delle x e il segmento orientato (0,0)-(x,y) . Si ha **atan2(y,x)=atan(y/x)** nel primo e nel quarto quadrante del piano cartesiano; nel secondo quadrante e nel terzo quadrante la funzione **atan2(y,x)** aggiunge **π** . Per esempio

```
»atan(-1)
ans =
   -0.7854
»atan2(-1,1)
ans =
   -0.7854
»atan2(1,-1)
ans =
    2.3562
```



Ovvero: l'arcotangente di –1 è –0.7854 (circa $-\pi/4$);
anche l'arcotangente2 del punto (1,–1) (nel quarto quadrante) è $-\pi/4$.
ma l'arcotangente2 del punto (–1,1) (nel secondo quadrante) è $3\pi/4$.
— Accenniamo anche alla funzione

```
»expm(a)
```

che permette di calcolare l'esponenziale matriciale di una matrice quadrata **a** .
Grosso modo, se **a** è una matrice quadrata, l'esponenziale matriciale di **a** è il limite della sommatoria **i+a+a^2/2!+a^3/3!+...** , oppure, se **a** è diagonalizzabile e quindi **a = p d p^{-1}** , allora **expm (a)=p exp(d) p^{-1}** dove l'esponenziale della matrice diagonale è quello ovvio. In realtà l'esponenziale è ottenuto attraverso un altro algoritmo, più complesso a descriversi, ma più efficiente. È possibile, in modo analogo, calcolare altre funzioni matriciali, quali **logm(a), sqrtm(a)** , **a^n** (con **n** qualunque) e altre.

COSTANTI E VARIABILI

Le funzioni predefinite in **MatLab** per ottenere le costanti più comuni sono: **pi** (pi greco) **i,j** (entrambe unità immaginaria).
Inoltre possono essere ottenute le variabili speciali

Inf (infinito), quando si divide un numero per 0 o si calcola il logaritmo di 0 .

NaN (Not a Number) che compare quando si esegue l'operazione 0/0 o l'operazione **Inf/Inf** .

Prestare inoltre attenzione al fatto che **pi** , **i ,j** , **Inf , NaN** possono essere definite come variabili e perdere in alcuni comandi il loro significato di funzioni o di variabili predefinite. Ma i comandi **pi** , **i ,j** rimangono comunque validi.
Inoltre è bene tener presente che le funzioni e le variabili in **MatLab** sono "case-sensitive" (cioè sensibili al maiuscolo minuscolo); per esempio le variabili **a** e **A** sono differenti e la funzione **Cos** (con la c maiuscola) non esiste.

MATRICI CASUALI

Le funzioni per ottenere una matrice casuale sono:

```
»rand(m,n)
»rand(n)
```

con **m,n** numeri interi.
Si ottengono rispettivamente una matrice casuale **m x n** e una matrice casuale quadrata di ordine **n** (possono essere utili per collaudare comandi o funzioni). Ognuno degli elementi è scelto casualmente con distribuzione uniforme ed è un numero compreso tra 0 e 1.
La curiosa funzione già vista sopra

```
»magic(n)
```

con **n** numero intero fornisce una matrice **n x n** che è un quadrato magico (stessa somma in tutte le righe, colonne e diagonali) (anche questa può essere utile per collaudare dei comandi).

NUMERI COMPLESSI

MatLab è in grado di lavorare con i numeri complessi e quindi anche con matrici a elementi complessi.
Occorre distinguere tra il comando **i** e la variabile **i** . Se alla variabile **i** non è stato assegnato alcun valore, allora, mediante il comando

```
»i
```

si ottiene:

```
i =
        0 + 1.0000i
```

In questo caso è possibile assegnare alle variabili di **MatLab** valori complessi con un comando del tipo:

```
»z=1+2*i
```

che assegna alla variabile **z** il numero 1+2*i* . Si ottiene:

```
z =
   1.0000 + 2.0000i
```

Se invece la variabile **i** è stata definita ed è stato assegnato un valore diverso da *i* , per esempio se si è posto

```
»i=3
```

si può avere il risultato imprevisto

```
»z=1+i
z =
   4
```

Si tenga comunque presente che per assegnare alla variabile **z** il numero 1+2*i* si possono usare le sintassi seguenti, senza il simbolo ***** , che prescindono dalla eventuale definizione della variabile **i** .

```
»z=1+2i
z =
   1.0000 + 2.0000i
»z=1+1i
z =
   1.0000 + 1.0000i
```

Comunque per riassegnare alla variabile **i** (o se si preferisce a **j**) il valore di unità immaginaria si può scrivere:

```
»i=sqrt(-1)
i =
        0 + 1.0000i
```

Il perché **i** debba essere $\sqrt{-1}$ (simbologia che, di norma, in matematica è bene evitare), verrà chiarito in seguito.
Oppure si può dare il comando

```
»clear i
```

che cancella qulunque valore assegnato alla variabile **i** . In questo caso il comando **i** (e anche, con le stesse cautele, il comando **j**) fornisce comunque l'unità immaginaria.

MODULO, ARGOMENTO E CONIUGATO

— Parte reale del numero complesso **z** .

```
»real(z)
```

— Parte immaginaria del numero complesso **z** .

```
»imag(z)
```

— Modulo del numero complesso **z** .

```
»abs(z)
```

— Argomento. Per calcolare l'argomento del numero complesso **z** si può usare la funzione **atan2.** La funzione **atan2(y,x)** fornisce *l'unico argomento* θ del numero complesso $x + i\,y$ *con* $-\pi < \theta \le \pi$. Quindi si può scrivere:

```
»atan2(imag(z),real(z))
```

Comunque c'è anche la funzione predefinita **angle(z)** che svolge la stessa identica funzione.

```
»angle(z)
```

MatLab conosce la formula di Eulero. Per esempio con

```
»exp(pi*i)
```

che corrisponde all'usuale $e^{\pi i}$ si ottiene:

```
ans =
     -1.0000 + 0.0000i
```

È anche possibile calcolare il logaritmo **log(x)** di un numero complesso **x** . In questo caso la funzione fornisce una soluzione **z** dell'equazione **exp(z)=x** . Per questo calcolo il numero **x** viene scritto come **x=r**

exp(iθ) con $-\pi < \theta \leq \pi$, quindi l'equazione diventa **exp(z)=r exp(iθ)** , ovvero **exp(z)=exp(log(r)+iθ)** . Tra le infinite soluzioni dell'equazione la funzione fornisce **z=log(r)+iθ.**
Per esempio è possibile ottenere il numero **π** in questo modo:

```
»log(-1)
ans =
        0 + 3.1416i
```

— Il coniugato del numero complesso **z** è

```
»conj(z)
```

— Lavorando con matrici a elementi complessi occorre prestare attenzione al fatto che la funzione di trasposizione **a'** applicata a matrici complesse fornisce la matrice trasposta coniugata complessa, mentre la semplice trasposta si ottiene con la funzione **a.'** . Per esemplificare:

```
»a=[i 1+i; 2+i 3-i]
a =
        0 + 1.0000i   1.0000 + 1.0000i
   2.0000 + 1.0000i   3.0000 - 1.0000i
»a'
ans =
        0 - 1.0000i   2.0000 - 1.0000i
   1.0000 - 1.0000i   3.0000 + 1.0000i
»a.'
ans =
        0 + 1.0000i   2.0000 + 1.0000i
   1.0000 + 1.0000i   3.0000 - 1.0000i
```

RADICI DI UN NUMERO COMPLESSO

Come appena detto, **MatLab** assegna ad ogni numero complesso, tra i tanti argomenti, un argomento privilegiato θ con $-\pi < \theta \leq \pi$, quindi per calcolare la radice *n*-esima di un numero complesso divide per *n* questo argomento e trova la radice aritmetica *n*-esima del modulo.
Per esempio, per calcolare la radice terza di $2 + 2i$ e assegnarla alla variabile **z** si scrive

```
»z=(2+2*i)^(1/3)
```

e si ottiene:

```
z =
     1.3660 + 0.3660i
```

che tra le radici terze di $2 + 2i$ è quella ottenuta dividendo per 3 l'argomento privilegiato $\pi/4$ di $2 + 2i$.
Ciò spiega anche perchè con **sqrt(-1)** si ottenga i , dato che -1 ha argomento π (e i ha argomento $\pi/2$).
Per ottenere le altre due radici occorre aumentare l'argomento di $2\pi/3$, cosa che si può fare moltiplicando questo numero successivamente per $e^{2\pi i/3}$ e per $e^{4\pi i/3}$. Per esempio col comando

```
»z*exp(2*pi*i/3)
```

si ottiene la successiva radice terza di $2 + 2i$.

```
ans =
     -1.0000 + 1.0000i
```

POLINOMI E LORO RADICI

MatLab è in grado di calcolare (in modo approssimato !) radici di polinomi mediante un complesso algoritmo.
Prima però occorre trasformare un polinomio in una matrice. Per esempio il polinomio $p = x^4 - 3x^3 - 5x + 2$ che, essendo di quarto grado, ha cinque coefficienti (uno è zero), va introdotto come quintupla di numeri, con i suoi coefficienti dalla potenza più alta alla più bassa:

```
»p=[1 -3 0 -5 2]
```

Dato che i polinomi possono essere considerati come funzioni, esiste un comando per calcolare la funzione polinomiale $p(x) = x^4 - 3x^3 - 5x + 2$ in qualsiasi punto x . Per esempio per calcolare $p(1)$ si scrive

```
»polyval(p,1)
ans =
    -5
```

E in effetti e $p(1) = -5$.
Per moltiplicare due polinomi occorre usare la funzione **conv** (convoluzione).
Per esempio, se $d = 3x^2 - 5x + 2$, il prodotto $p \cdot d$ si trova così:

```
»d=[3 -5 2];
»conv(p,d)
ans =
     3   -14    17   -21    31   -20     4
```

e in effetti $p \cdot d = 3x^6 - 14x^5 + 17x^4 - 21x^3 + 31x^2 - 20x + 4$.

La somma di polinomi è un po' più complicata: si sommano i vettori che li rappresentano, ma occorre che abbiano lo stesso grado. Per esempio, se $d = 3x^2 - 5x + 2$, per sommare p e d è necessario usare il polinomio ausiliario d_1 con gli stessi coefficienti di d, ma grado uguale a a quello di p, cosa che si ottiene aggiungendo degli zeri.

```
»d1=[0 0 3 -5 2];
»p+d1
ans =
     1    -3     3   -10     4
```

Si può anche eseguire la divisione con resto di due polinomi. La funzione relativa è **deconv** (deconvoluzione). Dato che però il risultato è dato da due oggetti, il quoto **q** e il resto **r**, occorre una sintassi leggermente diversa:

```
»[q,r]=deconv(p,d)
q =
    0.3333   -0.4444   -0.9630
r =
    0.0000   -0.0000   -0.0000   -8.9259    3.9259
```

Si noti che **r**, nonostante abbia grado minore di quello di **d**, ha virtualmente grado uguale a quello di **p** in modo che si possa facilmente verificare che $q \cdot d + r = p$. Per eseguire questo calcolo occorre il comando

```
»conv(q,d)+r
```

con cui si ottiene p (forse non esattamente, perché possono esserci arrotondamenti).
Le radici di un polinomio si trovano con la funzione **roots**. Per calcolare le radici di **p** si scrive

```
»roots(p)
```

e si ottiene una matrice colonna i cui elementi sono le quattro radici di p.

```
ans =
   3.3848
  -0.3788 + 1.2006i
  -0.3788 - 1.2006i
   0.3728
```

Per esempio, per trovare le radici terze di $2 + 2i$ si può anche usare la funzione:

```
»roots([1 0 0 -2-2*i])
```

cioè trovare le radici del polinomio $x^3 - 2 - 2i$. Ma non è detto che l'ordine delle radici sia quello per argomento.

NORME E PRODOTTI TRA VETTORI

La funzione **norm** calcola la norma euclidea di un vettore.

```
»norm([1 2])
ans =
    2.2361
```

Notiamo per inciso che la funzione **norm** applicata a un vettore ne calcola la norma euclidea; applicata a una matrice quadrata **a** ne calcola la "norma-2", il più grande valore singolare della matrice, che è il maggior autovalore della matrice simmetrica **a'*a** (che ha tutti autovalori non negativi). La norma-2 è utile nello studio della stabilità, rispetto alle variazioni dei suoi elementi, delle soluzioni di un sistema associato alla matrice.

— Prodotto scalare: Il calcolo del prodotto scalare di due vettori colonna **x** e **y** si può ricondurre a un prodotto di matrici mediante il comando

```
»x'*y
```

Comunque esiste la funzione

```
»dot(x,y)
```

— Per il prodotto vettoriale **x** ∧ **y** tra vettori a tre componenti esiste la funzione

```
»cross(x,y)
```

AUTOVALORI E AUTOVETTORI

Mediante complessi algoritmi **MatLab** è in grado di determinare gli autovalori e gli autovettori di una matrice.
Se **a** è una matrice quadrata **n** x **n**, la funzione

```
»eig(a)
```

produce una matrice colonna con tutti gli **n** autovalori di **a** (**eig** sta per eigenvalues: autovalori in inglese). Naturalmente è assai facile che, anche in una matrice reale, alcuni degli autovalori non siano reali. In questo caso alcuni degli elementi della matrice degli autovalori saranno complessi. Per esempio

```
»a =[1 2 3 ; 4 0 1 ; 1 2 -1];
»eig(a)
 ans =
   4.5810
  -2.2905 + 1.3187i
  -2.2905 - 1.3187i
```

Gli autovalori sono grosso modo in ordine decrescente di modulo, ma non sempre, dato che l'ordine in cui sono calcolati dipende dal complesso algoritmo effettuato da **MatLab** .
Se si desiderano anche gli autovettori, oltre gli autovalori, la sintassi della funzione va modificata, in modo da consentire l'output di due risultati, nel modo seguente:

```
»[p,d]=eig(a)
p =
   0.6645    -0.3953 - 0.2599i    -0.3953 + 0.2599i
   0.6577     0.1037 + 0.6559i     0.1037 - 0.6559i
   0.3548     0.4787 - 0.3259i     0.4787 + 0.3259i
d =
   4.5810           0                   0
        0   -2.2905 + 1.3187i           0
        0           0            -2.2905 - 1.3187i
```

In questo modo si ottengono perciò due matrici **p** e **d** tali che **a p = p d** .
La matrice **d** è diagonale e ha sulla diagonale gli autovalori.
Ogni colonna della matrice **p** contiene le coordinate di un autovettore relativo all'autovalore della colonna corrispondente di **d** .
Se la matrice **a** è diagonalizzabile, allora **p** è invertibile.
Se invece, come può capitare, la matrice **a** non è diagonalizzabile, allora la matrice **p** non è invertibile e le colonne corrispondenti agli autovalori con molteplicità maggiore di 1 possono essere uguali o proporzionali, se gli autospazi sono deficitari.
È possibile anche ottenere il polinomio caratteristico di una matrice quadrata **a** di ordine **n** mediante la funzione

```
»poly(a)
```

che produce un vettore di lunghezza **n+1** avente come elementi i coefficienti del polinomio caratteristico della matrice. I coefficienti sono elencati dalla potenza più alta alla più bassa. Il primo coefficiente è sempre 1.

E' utile notare che **MatLab** sceglie sempre autovettori (le colonne della matrice **p**) di modulo 1. Nel nostro esempio la cosa può essere verificata per esempio coi comandi

```
»v=p(:,2); norm(v)
 ans =
   1.0000
```

ll primo comando estrae la seconda colonna dalla matrice **p** (nel nostro esempio un autovettore dell'autovalore -2.2905 + 1.3187i), il secondo ne calcola la norma euclidea.
Nel caso di autovalori di molteplicità superiore a 1, il programma non calcola basi ortonormali per gli autospazi relativi, anche perché difficilmente viene rilevato che un autovalore abbia molteplicità maggiore di 1, essendoci errori di arrotondamento.
Le basi però sono ortonormali nel caso di matrice simmetrica. In questo caso la matrice **p** ottenuta mediante ila funzione **eig** a due output è una matrice ortogonale, cioè una matrice in cui tutte le colonne hanno modulo 1 e sono a due a due ortogonali.

FATTORIZZAZIONE QR

La fattorizzazione *QR* di una matrice quadrata invertibile **a** è ottenibile con la seguente funzione

```
»[q,r]=qr(a)
```

Con questa funzione si ottengono due matrici **q** e **r** tali che **q r = a** : la prima matrice **q** è ortogonale, in pratica l'ortogonalizzazione delle colonne di **a** ottenuta però, non con l'algoritmo di Gram-Schmidt, ma con il più efficiente algoritmo di Householder. La seconda matrice **r** è triangolare superiore e rappresenta la matrice di passaggio tra le due.
Con la stessa funzione è possibile calcolare la fattorizzazione *QR* di una matrice **a** di qualunque formato **m x n** e di qualunque caratteristica (anche se di solito si ha **m>n** e la matrice ha caratteristica pari al numero di colonne)
La matrice **q** ottenuta con questa funzione è quadrata e ortogonale, la matrice **r** ha lo stesso formato di **a** ed è triangolare superiore.

La fattorizzazione "economica" *QR* di una matrice qualunque si ottiene in questo modo:

```
»[q1,r1]=qr(a,0)
```

La matrice **q1** ottenuta in questo caso ha lo stesso formato di **a** e ha le colonne a due a due ortogonali e di modulo 1, mentre **r1** è quadrata **n x n** ed è triangolare superiore.
La **q1** ottenuta nel modo "economico" ha nelle colonne l'ortonormalizzazione delle colonne di **a** . In pratica è costituita dalle prime **n** colonne della **q** ottenuta col primo comando, mentre **r1** è ottenuta eliminando le ultime righe di **r** (che, se **m>n** , sono sempre nulle).
Nel caso delle colonne di **q** però, a causa degli errori di arrotondamento, le colonne sono di solito non perfettamente ortogonali, ma hanno comunque un prodotto scalare molto piccolo.

GLI OPERATORI RELAZIONALI

Gli operatori relazionali che consentono di confrontare gli elementi di due matrici sono i seguenti:

- **==** uguale (da non confondersi con il segno **=** che serve per assegnare valori alle variabili)
- **~=** diverso (il contrario del precedente)
- **<** minore (strettamente)
- **>=** maggiore o uguale (il contrario del precedente)
- **>** maggiore (strettamente)
- **<=** minore o uguale (il contrario del precedente)

Dei sei operatori, quindi tre sono esattamente contrari degli altri tre.
Si tenga presente che gli operatori relazionali sono operazioni binarie esattamente come la somma e il prodotto, e danno quindi luogo a un risultato.
La differenza è che il risultato può assumere solo due valori: **0** (se la relazione è falsa) e **1** (se la relazione è vera) e che il risultato è una matrice di tipo particolare, la matrice di tipo **logical**.
Le matrici di tipo logical sono utili in alcuni comandi di selezione di elementi di una matrice e in questo si distinguono da altre matrici formate solo di zeri e di uni, ma non ottenute da operatori relazionali.
L'operatore relazionale viene applicato ad ogni elemento delle matrici che si confrontano. Le matrici devono perciò avere lo stesso formato, oppure una di esse deve essere uno scalare.
Per esempio:

```
»x=[0 3 8 5 7]; y=[5 0 -1 6 7];
»x<y
ans =
     1     0     0     1     0
»x<=y
ans =
     1     0     0     1     1
»x==y
ans =
     0     0     0     0     1
»x<=5
ans =
     1     1     0     1     0
```

Gli operatori relazionali verranno usati soprattutto nei test condizionali degli M-files che seguono comando **if**, però le matrici di tipo logical ottenute in questo caso possono anche servire per scegliere gli elementi che interessano di una matrice. Per esempio mediante il comando

```
»x(x<=5)
ans =
     0     3     5
```

si ottiene la lista degli elementi di **x** che sono minori o uguali a 5.
Il comando precedente può essere applicato anche a una matrice **a** che non sia un vettore, ma in questo caso si ottiene una sottomatrice del flattening di **a**.

RICERCA DI ELEMENTI IN UNA MATRICE

I comandi che fanno uso delle matrici logical permettono di ottenere la lista degli elementi di una matrice che soddisfano particolari criteri, ma non la loro posizione nella matrice. A questo scopo c'è però la funzione **find** che elenca gli indici degli elementi non nulli di una matrice. Per esempio.

```
»find(x)
ans =
     2     3     4     5
```

elenca gli indici degli elementi non nulli di **x**.
Combinando la funzione **find** con gli operatori relazionali si può ottenere la lista degli indici degli elementi di una matrice che soddisfano particolari criteri. Per esempio

```
»find(x<=5)
ans =
     1     2     4
```

elenca gli indici degli elementi di **x** che sono minori o uguali a 5.
Sono pertanto equivalenti le due funzioni seguenti (la prima è quella vista sopra).

```
»x(x<=5)
»x(find(x<=5))
```

Se si lavora con una matrice **a** che non sia un vettore, la funzione **find** elenca gli indici degli elementi non nulli del flattening della matrice **a(:)**. Se si vogliono i veri indici degli elementi non nulli di una matrice **a**, la sintassi è la seguente:

```
»[i,j]=find(a)
```

La matrice colonna **i** fornisce gli indici di riga e la matrice **j** gli indici di colonna degli elementi cercati. Per esempio

```
»a=[0 5 2; 6 1 7; 0 1 -1];
»[i,j]=find(a>5)
i =
     2
     2
j =
     1
     3
```

Gli elementi maggiori di 5 della matrice **a** sono quelli di indici (2,1) e (2,3) .

— Il massimo e il minimo elemento di una vettore **x** si ottengono con le funzioni

```
»max(x)
»min(x)
```

Se si vogliono conoscere anche le posizioni del massimo e del minimo, la sintassi è la seguente

```
»[m,i]=max(x)
```

e si ottengono **m** , il valore del massimo e **i** l'indice del massimo (il primo indice se più di uno degli elementi di **x** è massimo). Analogamente per il minimo.

— Per riordinare gli elementi di un vettore dal minimo al massimo si usa la funzione **sort** . Per esempio

```
»x=[0 8 3 7 5];
»sort(x)
ans =
     0     3     5     7     8
```

Se si vogliono conoscere anche le posizioni degli elementi in ordine di grandezza, la sintassi è

```
»[x1,i]=sort(x)
x1 =
     0     3     5     7     8
i =
     1     3     5     4     2
```

La matrice **i** fornisce le posizioni in cui si trovavano in **x** gli elementi della matrice riordinata.
Questo è utile per esempio, se si ha anche una matrice **y** i cui elementi sono in corrispondenza con quelli di **x** . È possibile riordinare **y** mantenendo la corrispondenza con **x** , mediante la matrice **i** .

```
»x=[0 8 3 7 5]; y=[0 18 33 27 15];
»[x1,i]=sort(x);
»y(i)
ans =
     0    33    15    27    18
```

— Le funzioni **any** e **all** : la prima ritorna 1 se almeno un elemento della matrice è non nullo e 0 in caso contrario, la seconda ritorna 1 se tutti gli elementi della matrice sono non nulli e 0 in caso contrario . Le due funzioni sono solitamente usate in congiunzione con gli operatori relazionali.
Per comprenderne l'uso conviene osservare attentamente il seguente esempio:

```
»x=[1 2 3] ; y=[1 5 7] ; z=[10 11 12];
»any(x==y)
ans =
     1
»all(x==y)
ans =
     0
»any(x==z)
ans =
     0
»any(x~=y)
ans =
     1
»all(x~=y)
ans =
     0
»all(x~=z)
ans =
     1
»all(x<y)
ans =
     0
»all(x<z)
ans =
     1
```

INTRODUZIONE ALLA GRAFICA

MatLab ha grosse capacità grafiche. Ci limitiamo a descrivere il comando elementare **`plot(x,y)`** che è alla base di comandi più avanzati.
Il comando **`plot`** ha in pratica la funzione di disegnare una spezzata nel piano. Per esempio per disegnare la spezzata che congiunge i punti di coordinate (1,1) (2,2) (–1,1) (3,–1) (2,0) occorre formare due matrici riga **`x`** e **`y`** , la prima con le ascisse dei cinque punti, la seconda con le ordinate.

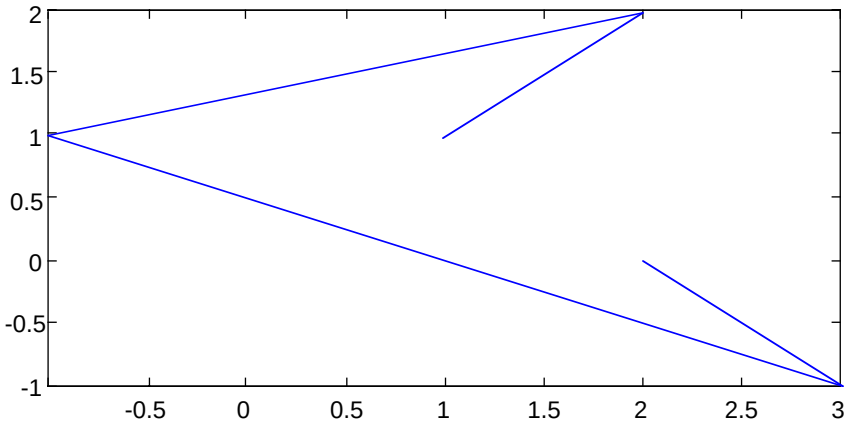
```
»x=[1 2 -1 3 2];
»y=[1 2 1 -1 0];
```

Dopodiché il comando

```
»plot(x,y)
```

disegna la spezzata che appare nella finestra grafica. Se la finestra grafica non appare, la si può richiamare col comando **`shg`** (showgraph)

```
»shg
```



Il grafico appare non monometrico, senza assi e solo circondato da una cornice graduata avente come limiti i minimi e i massimi delle ascisse e delle ordinate dei punti. Comando grafici più avanzati consentono di impostare altre opzioni.
Nell'esempio le scale del grafico hanno passo 0.5, ma dipendono solo dalla grandezza della finestra grafica.
Per esempio il comando

```
»axis=equal
```

consente di avere un grafico con assi monometrici.

Osserviamo ora che per disegnare il grafico di una funzione **`y = f(x)`** in un intervallo **`[a,b]`** è sufficiente disegnare la spezzata che ha come vertici vari punti del grafico.
Si può cominciare col suddividere l'intervallo in parti uguali scegliendo un passo più o meno ampio a seconda della precisione che si vuole ottenere. Per esempio, se l'intervallo fosse [–3,2] e il passo scelto fosse 0.1 allora si dovrebbe porre

```
»x=[-3  -2.9  -2.8 ..... 1.8  1.9  2]
```

ottenendo una matrice riga con 51 elementi. Il comando per ottenere automaticamente la matrice **`x`** è il seguente:

```
»x=-3:0.1:2
 Columns 1 through 7
  -3.0000  -2.9000  -2.8000 -2.7000 -2.6000 -2.5000  -2.4000
.......
```

Vengono elencati tutti i 51 elementi della matrice, quindi normalmente conviene far seguire il comando dal segno **`;`** che impedisce che la matrice venga scritta sul display.
Supponiamo di voler disegnare il grafico della funzione **`y = log (x^3-3x+2)`** nell'intervallo suddetto. Occorre tabulare la funzione, cioè calcolarla in tutti i punti scelti dell'intervallo in questione. Costruiamo quindi una matrice 1 x 51 **`y`** con tutti i valori corrispondenti ai valori di **`x`** .

```
»y=log(x.^3-3*x+2);
```

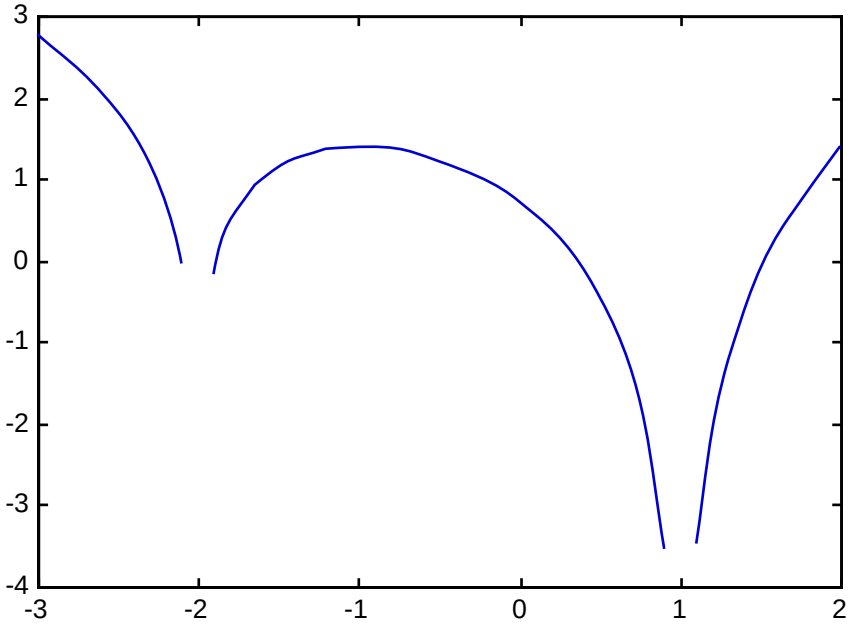
Notiamo il segno **`.^`** (potenza puntuale) per la potenza. Invece per moltiplicare **`x`** per lo scalare **`3`** non occorre il segno **`.*`** e anche la somma per uno scalare si ottiene semplicemente col segno **`+`** .
Avendo posto il segno **`;`** la matrice **`y`** non è riportata sul display, comunque compare la scritta

```
Warning: Log of zero.
```

che significa: Attenzione: log di zero. Questo perché tra i punti in cui è calcolata la funzione ci sono –2 e 1 dove la funzione non è definita.
In realtà la funzione non sarebbe definita anche tra –3 e –2 dato che si tratta del logaritmo di un numero negativo, ma in questo caso l'effetto è solo quello di ottenere valori non reali.
Ora possiamo procedere a disegnare il grafico con

```
»plot(x,y)
```

e si ottiene nella finestra grafica il grafico seguente:



Ovviamente la funzione non viene disegnata tra –2.9 e –1.9 e tra 0.9 e 1.1, dato che la matrice **y** non è definita in –2 e in 1 . Per quanto riguarda il pezzo di grafico tra –3 e –2 dove la funzione non è definita, esso è in realtà il grafico della funzione y=Re(log(x^3 –3x +2)) (parte reale). Se si lavora con funzioni di variabile reale il pezzo va trascurato.

GLI M-FILE

Dovendo eseguire più volte uno stesso comando complesso, non occorre ricopiarlo ogni volta. Si può invece scrivere il comando in uno "script" che viene registrato in un apposito file detto M-file. Lo script può poi essere richiamato quando necessario.
Per esempio se volessimo moltiplicare tutti gli elementi della diagonale di una matrice quadrata **a** per **5** potremmo eseguire i seguenti comandi

```
»d=diag(a)
```

che crea la matrice colonna contenente gli elementi della diagonale di **a** . Poi

```
»d1=diag(d)
```

che crea una matrice diagonale che ha la diagonale identica a quella di **a**. Poi

```
»e=a-d1
```

che crea una matrice **e** con diagonale nulla e per il resto identica ad **a**. Poi

```
»d1=d1*5
```

che moltiplica la diagonale di **d1** per **5** . Infine

```
»a1=d1+e
```

costruisce la matrice voluta. Il risultato finale è la matrice **a1** . (Questa procedura per moltiplicare la diagonale per **5** è puramente a titolo di esempio. In realtà esistono comandi più semplici).

Se vogliamo eseguire molte volte questa procedura e su diverse matrici, possiamo creare un M-File (c'è il comando apposito nel Menu "File") e su esso scrivere

```
1    d=diag(a);
2    d1=diag(d);
3    e=a-d1;
4    d1=d1*5;
5    a1=d1+e
```

(i numeri compaiono automaticamente e servono per il "debugging").
Conviene porre dei segni **;** alla fine di ogni riga del file tranne l'ultima, affinché eseguendo il comando non compaiano sul display i risultati intermedi delle matrici **d, d1, e** , ma compaia solo **a1.**
Indi si salva il file per esempio come "**diagocinque.m**" (suffisso **.m** obbligatorio) in una directory (folder) di quelle che il programma **MatLab** riconosce come valide nel suo elenco di "Path".

Ogni volta che si scriverà

```
»diagocinque
```

MatLab eseguirà la serie di comandi registrati sulla matrice **a** proprio come se li avessimo scritti ogni volta.
Le matrici **d, d1, e, a1** rimangono allocate nello spazio di lavoro corrente, ovvero vengono definite o ridefinite se già esistevano e possono essere richiamate.

DEFINIZIONE DI NUOVE FUNZIONI IN MATLAB

Naturalmente si vorrebbe poter eseguire il comando su una qualsiasi matrice senza che questa abbia necessariamente nome **a** . Per questo occorre definire un nuova funzione **MatLab** .
Questo si può fare mediante un M-File di tipo diverso che definisce una nuova funzione ed è congegnato così:

```
1    function y = diagocinque(a)
2    % moltiplica la diagonale per cinque
3    d=diag(a);
4    d1=diag(d);
5    e=a-d1;
6    d1=d1*5;
7    y=d1+e;
```

Si noti il "cappello" **function y = diagocinque(a).** Bisogna scegliere un nome (in questo caso **y**) per la variabile di uscita e quindi si noti che al risultato finale viene dato il nome **y** anzichè **a1** perché **y** è il nome che abbiamo scelto come variabile di output nella definizione iniziale alla riga 1.

Il nuovo M-File va salvato con il nome "**diagocinque.m**" che deve essere identico al nome dato alla funzione nella prima riga del listato.
Ogni volta che si calcola la funzione

```
»diagocinque(b)
```

su una qualunque matrice quadrata **b** si ottiene come **ans** la matrice **b** con la diagonale quintuplicata.
Osservazioni:

- Le righe del listato precedute dal simbolo **%** sono commenti ad uso di chi scrive e non vengono eseguite. Ma i commenti opzionali posti subito dopo la parola **function** appaiono nella finestra di comando quando si richiede aiuto per la funzione nel seguente modo:

```
»help diagocinque

  moltiplica la diagonale per cinque
```

- Gli M-File di tipo funzione hanno le loro variabili "locali". Per esempio, nella funzione ora definita, le variabili temporanee **a, d, d1, e, y** sono usate dall' M-File per eseguire la funzione. Esse vengono cancellate appena terminata l'esecuzione dello script e non interferiscono con le variabili definite dall'utente nello spazio di lavoro che possono anche avere gli stessi nomi.
- Dato che la variabile **y** è locale, conviene porre il simbolo **;** anche alla fine dell'ultima istruzione per evitare che il risultato compaia due volte sul display.

Le funzioni possono avere anche molti argomenti. Per esempio se si esegue molte volte l'operazione $P^{-1} \cdot A \cdot P - I$ su due matrici quadrate dello stesso ordine A e P , potrebbe convenire creare la funzione in due variabili

```
1    function y = pinvap(a,p)
2    % Esegue l'operazione inv(p)*a*p-I
3    y= inv(p)*a*p-eye(a)
```

e sarebbe cura dell'utente che la matrici fossero quadrate e dello stesso ordine.

IL LOOP for...end NEGLI M-FILE

Esistono moltissimi comandi per creare M-File assai complessi e versatili e strutture logiche simili a quelle usate nei comuni linguaggi di programmazione, tipo **for … end , if … then … else , while … wend** etc. con ampia gamma di opzioni.
La più semplice è il loop **for … end**.
Dovendo per esempio eseguire il comando $R_j \rightarrow R_j - (a(j,1))/a(1,1))R_1$ (primo passo dell'algoritmo gaussiano) sulle righe di una matrice **a** di 10 righe si dovrebbero eseguire le nove istruzioni

```
»a(2,:)=a(2,:)-(a(2,1)/a(1,1))*a(1,:);
»a(3,:)=a(3,:)-(a(3,1)/a(1,1))*a(1,:);
.......
»a(10,:)=a(10,:)-(a(10,1)/a(1,1))*a(1,:);
```

Conviene quindi creare un M-File con loop del tipo **for :**

```
1    for j=2:10
2        a(j,:)=a(j,:)-(a(j,1)/a(1,1))*a(1,:);
3    end
```

Il comando **for** fa eseguire tutte le righe che successive fino al comando **end** facendo assumere a **j** successivamente i valori 2,3,…,10.
L'editore degli M-Files provvede ad "indentare" automaticamente tutti i comandi che appaiono tra **for** e **end** e che fanno parte del loop, cioè a spostarli a destra in modo che il file sia più leggibile e i loop rimangano evidenziati.
Se si vuole che **j** assuma i valori 2,4,6,…,20 la sintassi sarà

```
1     for j=2:2:20
```

Se si vuole che **j** assuma i valori 1 , 1.1 , 1.2, … , 3 la sintassi sarà

```
1     for j=1:0.1:3
```

Se si vuole che **j** assuma successivamente i valori 10,9,8,…,0 la sintassi sarà

```
1     for j=10:-1:0
```

I loop **for...end** possono essere "nidificati", cioè inseriti uno dentro l'altro. Per esempio se si vuole una funzione che crei una matrice quadrata **n** x **n** in cui l'elemento di posto **(i,j)** sia **i*j+2**, si potrebbe creare una funzione del tipo seguente che dipende solo da **n** .

```
1     function a = pippo(n)
2
3     for j=1:n
4          for i=1:n
5               a(i,j)=i*j+2;
6          end
7     end
```

Notare i due loop dipendenti da due variabili diverse **i** e **j** , con il loop in **i** "nidificato" dentro al loop in **j** e la doppia indentazione.
Alla funzione va dato un nome di fantasia, per esempio **pippo** . Occorre badare che il nome scelto non sia già il nome di un comando **MatLab** o di un M-File già esistente.
Nel primo caso la funzione **pippo** non sarebbe riconosciuta a favore dell'esistente comando **MatLab** .
Nel secondo caso l' M-File già esistente andrebbe perso, se salvato nella stessa Folder, oppure potrebbe non essere riconosciuto a favore del nuovo comando, se salvato in altra Folder.

Osserviamo che una funzione cancella il valore precedente della variabile di output.
Quindi, se per esempio fosse già definita nello spazio di lavoro una matrice **m** di formato, diciamo 10 x 10 e si eseguisse la funzione **m=pippo(5)** , dato che lo script è una funzione, la matrice **m** sarebbe completamente ridefinita e avrebbe formato 5 x 5.
Se invece lo script fosse un M-File semplice senza l'istruzione alla riga 1, allora lo script modificherebbe solo 25 elementi della matrice **a** , lasciando invariati gli altri e il formato della matrice resterebbe invariato (aumenterebbe solo se **a** avesse meno di 5 righe o meno di 5 colonne).

IL COMANDO if...end

Il test condizionale con **if … end** viene usato quando si vuole che un comando venga eseguito solo se è verificata una certa condizione. Il test con **if** viene usato quasi esclusivalmente negli M-files, anche se sarebbe possibile usarlo nella riga di comando.
La sintassi è

```
if [operazione]
 [comandi da eseguire se il risultato non è 0]
else
 [comandi da eseguire se il risultato è 0]
end
```

Di solito l'operazione che segue il comando **if** coinvolge un operatore relazionale. Quindi la prima serie di comandi è eseguita solo se la relazione è vera.
I comandi che seguono **else** sono eseguiti se la relazione è falsa. Il comando **else** è opzionale e può essere omesso. In tal caso, se la relazione è falsa, non viene fatto niente.
Esempio 1: Se si vuole eseguire direttamente l'algoritmo di Gauss su una matrice, si potrebbe voler evitare di eseguire l'operazione $R_3 \rightarrow cR_3$ su una matrice **a** quando il numero **c** è zero.
Il comando per l'operazione elementare è:

```
»a(3,:)=c*a(3,:)
```

Quindi si potrebbero dare i seguenti comandi:

```
if c~=0
  a(3,:)=c*a(3,:)
end
```

Esempio 2: Si vogliono cambiare in 0 tutti gli elementi non reali di una matrice riga **x** . È possibile creare mediante un M-File una funzione che esegua questa operazione per ogni matrice riga. La funzione seguente agisce su una matrice riga e ne crea un'altra seguendo appunto questa regola.

```
1    function y=zeroc(x)
2
3    s=size(x,2);
4    for index=1:s
5         if imag(x(index))~=0
6              y(index)=0;
7         else
8              y(index)=x(index);
9         end
10   end
```

Osservazioni sulle righe del listato:

1 La funzione ha un nome di fantasia **zeroc** . Il nome della variabile di input è **x** , quello della variabile di output è **y** . Da notare che, visto che abbiamo definito una funzione, **x** e **y** sono locali alla funzione stessa, cioè non interferiscono con le variabili definite nello spazio di lavoro.

3. La funzione **size(x,2)** calcola il numero di colonne della matrice **x** per saper quante volte va eseguito il loop.

4-10 Questo è il loop **for...end** che esamina tutti gli elementi della matrice riga **x** . La variabile di controllo del loop è stata chiamata **index** .

5-9 Questo è il comando **if...else...end** . L'istruzione 6 viene eseguita se la parte immaginaria dell'elemento di **x** non è zero (cioè se l'elemento non è reale). In caso contrario viene eseguita l'istruzione 8 che ricopia l'elemento di **x** nel corrispondente di **y** .

Per terminare osserviamo che occorre qualche cautela nell'usare il comando **if...end** con gli operatori relazionali, quando ci si riferisce a matrici e non a scalari, dato che gli operatori relazionali confrontano tutti gli elementi delle matrici.

Per esempio se si vuole eseguire un certo comando solo sulle matrici simmetriche, la sintassi corretta è

```
if a==a.'
 [comandi da eseguire se a è simmetrica]
end
```

Se invece si vuole esguire un certo comando solo su matrici non simmetriche, la sintassi seguente non funziona

```
if a~=a.'
 [comandi da eseguire se a non è simmetrica]
end
```

Il comando non verrà mai esguito, perché l'operatore relazionale **a~=a.'** restituisce sempre una matrice non completamente fatta di 1.

Conviene quindi una delle seguenti due sintassi:

```
if a==a.'
else
 [comandi da eseguire se a non è simmetrica]
end
```

Oppure, sfruttando la funzione **any** :

```
if any(a~=a.')
 [comandi da eseguire se a non è simmetrica]
end
```