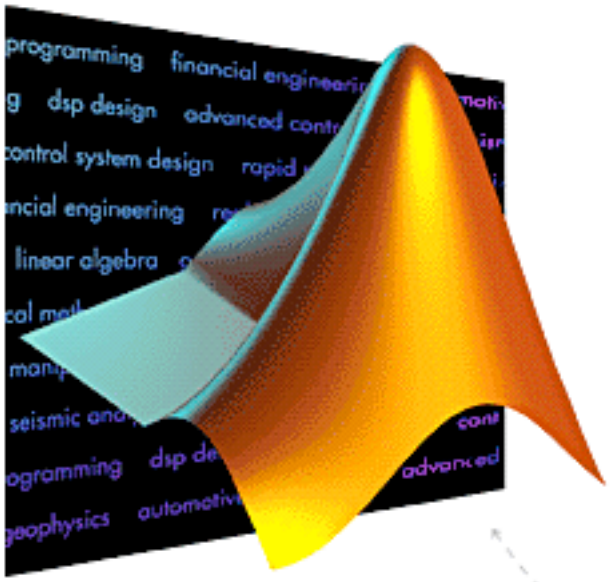
Introduzione all'uso di

MATLAB®

The Language of Technical Computing



The MATH WORKS Inc.

Francesco Odetti
DIPTEM
Università di Genova
A.a. 2009/10

MatLab è un programma studiato apposta per operare su matrici. Il nome è 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 $a = \begin{pmatrix} 5 & 0 & 2 \\ 1 & 4 & -1 \\ -2 & 2 & 3 \end{pmatrix}$ 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`) $\neq$ 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
```

Osservazione: 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 4×4 **s** che ha sulla diagonale 1 , 2 , 3 , 4 si può operare così:

```
» s1=[1 2 3 4];
» s=diag(s1)
```

Osservazione: dato che la matrice riga **s1** è solo un passaggio intermedio del conto, è possibile evitare che venga scritta su schermo apponendo il simbolo **;** dopo la prima istruzione.
Questa opzione verrà spesso usata anche in seguito.
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** $\times$ **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** $\times$ **n** si ottiene con:

```
» zeros(n)
```

La matrice nulla rettangolare **m** $\times$ **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]
```

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

Analogamente alla matrice nulla è possibile ottenere una matrice tutta composta di numeri 1 coi comandi:

```
» ones(n)
» ones(m,n)
```

Per esempio

```
» a=5*ones(2,3)
a =
     5     5     5
     5     5     5
```

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 $R_3 \to 5R_3$ su `a` si può eseguire con il comando seguente:

```
» a(3,:)=5*a(3,:)
```

L'operazione $R_3 \to R_3 + 5R_2$ 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 funzione **magic(n)** restituisce una matrice **n** $\times$ **n** che è un quadrato magico di ordine **n** (la somma di ogni riga, di ogni colonna e di ogni diagonale è sempre la stessa, in questo caso è 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** $\times$ **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** $\times$ **n** o **n** $\times$ **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.

Il comando **[ ]** fornisce la matrice vuota. Può essere usato per sopprimere parti di una matrice. Per esempio

```
» m(1:2,:)=[ ]
```

sopprime le prime due righe di **m** .

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 `a+k` non è equivalente a `a+k*I` .

• 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. Per i dettagli di questa operazione, vedi più avanti l'algoritmo di Gauss.

```
» 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 1×2 con il numero di righe e il numero di colonne di `a` .
Si può per esempio scrivere

```
» [r,c]=size(a)
```

per avere in `r` e in `c` rispettivamente 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)
```

Un'altra funzione spesso usata è la seguente

```
» length(a)
```

che restituisce la **massima** dimensione di `a` ed è quindi usata di solito per conoscere la lunghezza di un vettore.

HELP ON LINE

Per avere informazioni su un dato comando esiste il comando `help` che fornisce rapidamente (in inglese) uno *help-on-line*, cioè una sintetica descrizione del comando e del suo uso. Per esempio

```
» help length

 LENGTH Length of vector.
        LENGTH(X) returns the length of vector X. It is equivalent
    to MAX(SIZE(X)) for non-empty arrays and 0 for empty ones.
```

Attenzione: i comandi e le funzioni in **MatLab** vanno introdotti con le lettere minuscole, mentre lo help-on-line

le riporta in carattere maiuscolo per metterle in evidenza.

Comunque le versioni più recenti di **MatLab** hanno uno help sotto forma di ipertesto assai completo (anche se meno immediato).

COSTANTI, VARIABILI E FORMATO

Le variabili in **MatLab** possono avere un nome lungo un massimo di 31 caratteri. La prima lettera di una variabile deve essere un carattere alfabetico (`a-z` , `A-Z`) mentre dalla seconda lettera in avanti possiamo utilizzare un qualsiasi carattere alfanumerico incluso il simbolo underscore _ .
È bene tener presente che anche le variabili in **MatLab** sono "case-sensitive" (cioè sensibili al maiuscolo minuscolo); per esempio le variabili `a` e `A` sono differenti e la funzione `Det` (con la d maiuscola) non esiste.

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

`Inf` (infinito), quando si divide un numero per 0 o si calcola il logaritmo di 0 o si va in overflow.
`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` , `eps` , `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` , `eps` , `j` rimangono comunque validi.

L'insieme delle variabili definite dall'utente viene detto **workspace** ed è possibile visualizzarne l'elenco mediante il comando

```
» who
```

Per avere maggiori informazioni per ogni variabile si può usare

```
» whos
```

che fornisce, per ogni variabile anche il formato, il numero di byte usati e la classe.
Esistono varie classi di variabili.
Le matrici di default sono formate da numeri in doppia precisione che occupano 8 byte per elemento (classe `double`). Oltre a queste ci sono altri tipi di variabili quali le stringhe alfanumeriche (classe `char`), le variabili logiche (classe `logical`) che vedremo più avanti, e altre.

Per sapere a che classe appartiene una data variabile (per esempio `a`) esiste il comando

```
» class(a)
```

Osserviamo comunque che le ultime versioni di **MatLab** possono visualizzare in un'apposita finestra il **Workspace** ed è disponibile un editore per le variabili (che sono matrici) simile a un foglio elettronico (tipo Excel).

Per eliminare una variabile (per esempio `a`) dal *workspace* esiste il comando.

```
» clear a
```

Lo stesso comando senza argomento elimina dal *workspace* tutte le variabili

```
» clear
```

Normalmente ogni variabile è memorizzata internamente in doppia precisione e quindi è esatta fino a circa la 15-ma cifra decimale, ma in genere vengono visualizzate solo le prime quattro cifre decimali. Per esempio

```
» x=pi/2
x =
    1.5708
```

Per far sì che **MatLab** visualizzi tutte le cifre decimali disponibili di ogni numero occorre il comando

```
» format long
x =
    1.57079632679490
```

Per ripristinare la visualizzazione breve si usa

```
» format short
```

Il comando `format` ha inoltre le due opzioni

```
» format compact
» format loose
```

che fanno passare rispettivamente alla visualizzazione compatta senza righe vuote (che permette di far comparire più righe sullo schermo) e a quella larga con una riga vuota tra un comando e la sua risposta.
Le schermate riportate in questo manuale sono state ottenute con l'opzione `format compact` per risparmiare spazio.

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 dirette e inverse (funzionano anche se l'elemento è complesso).

```
» sin(x)
» cos(x)
» tan(x)
```

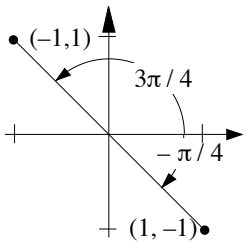
```
» asin(x)
» acos(x)
» atan(x)
```

È bene ricordare che la funzione arcotangente `atan(x)` restituisce l'unico numero θ con $-\pi/2 < \theta < \pi/2$ tale che tang(θ) sia `x` (θ in radianti).

Esiste quindi un'altra funzione denotata `atan2` (arcotangente a due variabili) che tiene conto del quadrante nel piano cartesiano.
Precisamente, `atan2(y,x)` è l'unico numero θ con $-\pi < \theta \leq \pi$ tale che $\begin{cases} \cos(\theta) = x \Big/ \sqrt{x^2+y^2} \\ \sin(\theta) = y \Big/ \sqrt{x^2+y^2} \end{cases}$

Geometricamente `atan2(y,x)` è l'angolo orientato θ in radianti tra l'asse positivo delle x e il segmento orientato $(0,0)-(x,y)$.
Nel primo e nel quarto quadrante del piano cartesiano si ha `atan2(y,x)=atan(y/x)` ; mentre nel secondo e nel terzo quadrante la funzione `atan2(y,x)` aggiunge π a `atan(y/x)` .
Osservare che negli argomenti di `atan2` l'ordinata `y` viene prima dell'ascissa `x` .
Esempio

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



che va interpretato così:
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 2.3562 (circa $3\pi/4$).

• Accenniamo anche alla funzione

```
» expm(a)
```

che permette di calcolare l'**esponenziale matriciale** di una matrice quadrata `a` .
Se `a` è una matrice quadrata, l'esponenziale matriciale di `a` si può definire grosso modo come il limite della

sommatoria `I+a+a^2/2!+a^3/3!+...` (`I` matrice identica)
Se `a` è diagonalizzabile e quindi `a = p d p^(-1)` , allora si può definire `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 diverse altre.

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` $\times$ `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.
Da notare che in realtà i numeri sono *pseudo*-casuali e l'algoritmo di generazione dei numeri pseudocasuali viene resettato ad ogni sessione di **MatLab** per cui il primo numero ottenuto con `rand` è sempre 0.9501 e la successione di numeri casuali è sempre la stessa.
Per avere un numero veramente imprevedibile, viene consigliato di resettare la successione di numeri casuali con un comando del tipo

```
» rand('state',sum(100*clock))
```

dopodiché il primo numero pseudocasuale è generato usando la data e l'ora corrente.

La curiosa funzione già vista sopra

```
» magic(n)
```

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

ALGORITMO DI GAUSS

Se `a` è una matrice quadrata invertibile e `b` una matrice colonna con con lo stesso numero di righe 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` e `b` sono matrici qualunque (di qualunque formato e di qualunque rango, 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 di `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` (row reduced 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).
Tra le demo c'è un'interessante funzione a scopo didattico, purtroppo mancante nelle ultime versioni di **MatLab**

```
» 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 opportunamente 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)
```

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 $1 + 2i$. 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 $1+2i$ 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 qualunque valore assegnato alla variabile `i`. In ogni 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, tra tutti gli argomenti di `z`, l'**unico argomento** θ con $-\pi < \theta \leq \pi$ del numero complesso $x+iy$. 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(i`θ`)` con $-\pi < \theta \leq \pi$, quindi l'equazione diventa `exp(z)=r exp(i`θ`)`, ovvero `exp(z)=exp(log(r)+i`θ`)`.
Tra le infinite soluzioni dell'equazione la funzione fornisce `z=log(r)+i`θ.
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 perché con **sqrt(-1)** si ottenga i , dato che -1 ha argomento π (e i ha argomento $\pi/2$).
Per ottenere le altre due radici di $2+2i$ 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 $d1$ 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
```

Vale la pena di osservare che la funzione `norm` applicata, invece che a un vettore, a una matrice quadrata `a` ne calcola la **norma-2**, il più grande **valore singolare** della matrice.
I valori singolari di una matrice sono le radici quadrate degli autovalori 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 lineare 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 $\mathtt{x} \wedge \mathtt{y}$ 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 $\mathtt{n} \times \mathtt{n}$, 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.

È interessante sapere che il comando `roots` per determinare le radici di un polinomio `p` equivale in realtà al comando `eig(compan(p))`.
La matrice **compagna** di un polinomio `p`, che si ottiene col comando `compan` è una semplice matrice quadrata che ha come polinomio caratteristico proprio `p`.
Quindi, per determinare le radici di un polinomio, **MatLab** calcola gli autovalori di una matrice e non viceversa, come poteva essere naturale pensare.

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 la 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 (anche non perfettamente, a causa di errori da approssimazione).

FATTORIZZAZIONE QR E ORTONORMALIZZAZIONE

La fattorizzazione QR di una matrice quadrata invertibile `a` è ottenibile con la seguente funzione a due output

```
» [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 e le sue colonne sono un'**ortonormalizzazione** dei vettori colonna di `a` ottenuta però, non con l'algoritmo di Gram-Schmidt, ma con il più efficiente algoritmo di Householder e quindi leggermente differente da quella ottenibile con Gram-Schmidt.
La seconda matrice `r` è triangolare superiore e rappresenta la matrice di passaggio tra le due.

È possibile calcolare con la stessa funzione la fattorizzazione QR di una matrice `a` di qualunque formato `m` $\times$ `n` e di qualunque rango (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, e quindi è una **ortonormalizzazione** delle colonne di `a`, mentre `r1` è quadrata `n` $\times$ `n` ed è triangolare superiore.
La `q1` ottenuta nel modo economico 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).

Osserviamo comunque che le colonne delle matrici `q` ottenute colla funzione `qr`, a causa degli errori di approssimazione, sono di solito non perfettamente ortonormali, ma hanno comunque un prodotto scalare molto piccolo (dell'ordine di 10^{-15}) e un modulo molto prossimo a 1 (sempre a meno di 10^{-15} circa).

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 classe **logical**.
Le matrici di classe 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 vengono usati soprattutto nei test condizionali che seguono di solito il comando `if`.
Comunque le matrici di classe 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))
```

Osserviamo che 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(:)`.