Laboratorio di Informatica I
a.a. 2004/05
INTRODUZIONE A MATLAB®

prof. Agostino Dovier, dott. Dimitri Breda
Dipartimento di Matematica e Informatica
Università degli Studi di Udine
dovier,dbreda@dimi.uniud.it
http://www.dimi.uniud.it/dovier,dbreda

Indice

1 Premessa

Questa introduzione non vuole sostituire i manuali e nemmeno la guida *help* in linea ma vuole presentare brevemente il programma MATLAB ed evidenziarne alcune importanti caratteristiche.

Fin dagli anni '60 si era cercato di sviluppare un programma che fosse in grado di fornire un supporto semplice e veloce agli utenti che utilizzavano il calcolo matriciale. L'idea alla base di tale progetto era quella di fornire un interprete con il quale fosse possibile risolvere i problemi dell'algebra lineare, quali ad esempio il prodotto matriciale, il calcolo del determinante, il calcolo dell'inversa di una matrice, la ricerca degli autovalori, etc., con un'unica istruzione, venendo così incontro anche all'utente che non conosceva uno specifico linguaggio di programmazione.

Già esistevano dei programmi scritti in Fortran e presenti nelle librerie EISPACK e LINPACK e, proprio sfruttando tali prodotti, fu creata la prima versione di MATLAB (MATrix LABoratory), scritta interamente in Fortran. Con lo sviluppo degli elaboratori elettronici e dei linguaggi di programmazione è stato possibile realizzare nuove versioni di MATLAB, scritte in linguaggio C, migliori delle precedenti. Attualmente si è raggiunta la versione 7 (5.3 in laboratorio: per verificare da MATLAB si digiti il comando *version*) e con MATLAB ora si può disporre di un ambiente non solo dedicato all'algebra lineare ma anche capace di risolvere numericamente svariati altri problemi come la ricerca di zeri di funzioni non lineari, la ricerca di minimi e massimi, il calcolo di integrali definiti, l'integrazione di equazioni differenziali, etc..

Un'altra caratteristica importante di MATLAB è quella di gestire automaticamente l'allocazione di memoria delle matrici, sollevando così l'utente dal compito del dimensionamento delle stesse, cosa che invece è richiesta, ad esempio, in Fortran e C. Infine MATLAB non è solo un interprete di istruzioni, ma è anche un linguaggio di programmazione evoluto che dà la possibilità all'utente di scrivere propri programmi.

MATLAB è in genere accompagnato da toolbox: una *toolbox* è un insieme di file che estendono MATLAB per risolvere particolari classi di problemi, come ad esempio ottimizzazione, elaborazione di segnali digitali, design ed analisi di sistemi di controllo, statistica, etc. Per l'elenco completo e aggiornato delle toolbox (e anche per altre informazioni su MATLAB) si vedano le pagine web *www.mathworks.com* e *www.teoresi.it* (Mathworks è la ditta americana produttrice di MATLAB e Teoresi è il distributore italiano) oppure si dia, da MATLAB, il comando *ver*.

2 Prime istruzioni

2.1 Sessioni

Una volta aperta la sessione MATLAB compare il *prompt*

```
>>
```

A questo punto MATLAB è pronto a ricevere istruzioni e ad eseguirle. La sessione viene chiusa con il comando *quit* oppure con *exit*. Nel seguito si suppone che l'inizio di ogni nuova sezione di questa dispensa coincida con l'inizio di una nuova sessione MATLAB.

2.2 Istruzioni base

Supponiamo di voler dividere 17 per 26 e assegnare il risultato alla variabile a. Basterà scrivere

```
>>a=17/26
```

Il simbolo "=" è l'operatore di assegnamento in MATLAB. Premendo il tasto "Invio" l'istruzione viene eseguita e si ottiene la risposta

```
a=
    0.6538
```

MATLAB ha memorizzato nella variabile a il risultato della divisione. Non specificando alcuna variabile a cui assegnare il risultato come in

```
>>17/26
```

si ottiene

```
ans=
    0.6538
```

Il risultato è stato in questo caso assegnato di default alla variabile *ans* (che sta per *answer*). Se non si vuole visualizzare il risultato assegnato, l'istruzione va terminata con un punto e virgola ";". Ad esempio:

```
>>b=1/a;
```

Per vedere il "contenuto" di una variabile basta digitare al prompt la variabile stessa. Ad esempio:

```
>>a
a=
    0.6538
>>b
b=
    1.5294
```

Usando tutti i noti operatori $+$, $-$, $*$, $/$, $\hat{}$ con le usuali precedenze, eventualmente alterate con parentesi tonde, si possono costruire espressioni aritmetiche, ad esempio:

```
>>7+6/5*4^3-2
ans=
    81.8000
>>7+6/(5*4)^3-2
ans=
    5.0008
```

Si possono scrivere più istruzioni sulla stessa riga separandole con virgole ","(o con ";" se non si vuole visualizzare i risultati delle istruzioni precendenti). Esempio

```
>>a=2,a=1+1/a;a=1+1/a;a=1+1/a
a=
    2
a=
    1.6000
```

Viceversa è anche possibile continuare a scrivere un'istruzione sulla riga successiva inserendo i tre punti "..." e premendo il tasto "Invio", Ad esempio:

```
>>1+1/2+1/3+1/4+...
1/5+1/6+1/7+1/8+...
1/9+1/10+1/11+1/12
ans=
    3.1032
```

In MATLAB sono presenti le principali funzioni matematiche:

sqrt	radice quadrata
abs	modulo
sin, cos, tan	funzioni trigonometriche (NB: argomento in radianti)
asin, acos, atan	funzioni trigonometriche inverse
exp	funzione esponenziale
log, log10, log2	logaritmo naturale, logaritmo base 10, logaritmo base 2.

Ad esempio con l'istruzione

```
>>sqrt(2)+exp(1)*sin(pi/4)
ans=
    3.3363
```

si è valutata l'espressione $\sqrt{2} + e \sin(\pi/4)$. Si noti la costante *pi* il cui valore è π.

MATLAB manipola anche numeri complessi. Tramite le costanti i e j che stanno entrambe per l'unità immaginaria si può scrivere un numero complesso in forma algebrica o trigonometrica. Esempio

```
>>z=sqrt(2)+i*sqrt(2)
z=
```

```
    1.4142+1.4142i
>>z=2*exp(i*pi/4)
z=
    1.4142+1.4142i
```

L'unità immaginaria, se moltiplica a destra costanti numeriche, può anche essere semplicemente giustapposta. Esempio:

```
>>z=1+3.5i
z=
    1.0000+3.5000i
>>z=2*exp(6i)
z=
    1.9203-0.5588i
```

Sono disponibili le seguenti funzioni di numeri complessi:

real	parte reale
imag	parte immaginaria
abs	modulo
angle	argomento in $(-\pi, \pi]$
conj	coniugato.

2.3 Notazione scientifica

I numeri possono essere scritti direttamente in notazione scientifica (senza usare l'esponenziazione a 10): ad esempio $2.25 \cdot 10^{-4}$ e $1.04 \cdot 10^{22}$ si possono scrivere

```
>>a=2.25e-4,b=1.04e22
a=
    2.2500e-004
b=
    1.0400e+022
```

2.4 Numeri macchina

La rappresentazione binaria interna dei numeri reali in MATLAB segue lo *standard IEEE 754* in *doppia precisione* $\mathcal{F}(\beta, t, p_{min}, p_{max}) = \mathcal{F}(2, 53, 1021, 1024)$, cioè i numeri sono memorizzati in una parola di 64 bit divisi come segue: un primo bit per il *segno*, i successivi 11 bit per l'*esponente* e infine 52 bit per la *mantissa*. I possibili numeri di macchina (non zero) normalizzati sono della forma

$$\pm (d_1 d_2 \dots d_{53})_2 \cdot 2^p$$

dove $d_1 = 1$, $d_i \in \{0, 1\}$ e $p \in \{-p_{min}, \dots, p_{max}\} = \{-1021, \dots, 1024\}$. Nel primo bit compare 1 per $-$ o 0 per $+$, nei successivi 11 bit compare la rappresentazione binaria (con eventuali zeri di testa) della caratteristica $c = p + p_{min} + 1 \in \{1, \dots, 2046\}$ e, infine, negli ultimi 52 bit si trovano le

ultime cifre binarie $d_2, d_2, ..., d_{53}$ in quanto nella rappresentazione normalizzata binaria la prima cifra è sempre $d_1 = 1$ e quindi non serve memorizzarla.
Il numero normalizzato più grande (in modulo) rappresentabile è quindi

$$(\overbrace{11...1}^{53})_2 \cdot 2^{1024} \simeq 2^{1024} \simeq 10^{308} \quad (\beta^{p_{max}}(1-\beta^{-t}))$$

e il numero normalizzato (diverso da zero) più piccolo (in modulo) è invece

$$(1\overbrace{00...0}^{52})_2 \cdot 2^{-1021} = 2^{-1022} \simeq 10^{-308} \quad (\beta^{-p_{min}-1}).$$

Tali numeri sono i valori delle costanti *realmax* e *realmin*:

```
>>realmax,realmin
ans=
    1.7977e+308
ans=
    2.2251e-308
```

Lo *standard IEEE 754* in *doppia precisione* di MATLAB permette anche la rappresentazione dei numeri denormalizzati, cioè quelli con la prima cifra significativa $d_1 = 0$ e esponente minimo $p = -p_{min} = -1021$ (in teoria i bit dell'esponente sono tutti nulli tranne l'ultimo, nella pratica invece sono tutti nulli per indicare che $d_1 = 0$). Così il numero minimo rappresentabile è

$$(\overbrace{00...0}^{52}\,1)_2 \cdot 2^{-1021} = 2^{-1021-53} = 2^{-1074} \simeq 10^{-324} \quad (\beta^{-p_{min}-t}).$$

```
>> xmin=2^(-1074)
xmin=
    4.9407e-324
```

Quando viene introdotto, oppure generato in una computazione, un numero in modulo più grande di *realmax* (cioè si ha *overflow*) questo viene memorizzato come ∞ o $-\infty$ a seconda del segno. Gli infiniti hanno nello standard IEEE una rappresentazione di macchina: i bit dell'esponente sono tutti 1, corrispondente al numero decimale 2047, e i bit della mantissa sono tutti 0. L'infinito positivo è il valore della costante *Inf*. Esempio:

```
>>realmax*2
ans=
    Inf
>>2+Inf
ans=
    Inf
>>-2*Inf
ans=
    -Inf
```

L'infinito è anche il risultato di una divisione per zero:

```
>>1/0
ans=
    Inf
```

Quando viene introdotto, oppure generato, un numero in modulo più piccolo di *xmin* (cioè si ha *underflow*) questo viene memorizzato come 0 (i bit della mantissa e qualli dell'esponente sono tutti 0, corrispondente al numero decimale 0).

Il risultato di operazioni non definite come $\frac{0}{0}$ e $\infty - \infty$ è posto uguale a "indeterminato" che, nello *standard IEEE 754* in *doppia precisione*, ha una rappresentazione di macchina ed è il valore della costante *NaN* (Not a Number): i bit dell'esponente sono tutti 1, corrispondente al numero decimale 2047, e almeno un bit della mantissa è $\neq 0$.

```
>>0/0
ans=
    NaN
>>Inf-Inf
ans=
    NaN
>>2+NaN
ans=
    NaN
```

segno 1 bit	caratteristica 11 bit	mantissa 52 bit	numero
0/1	$0\cdots\cdots 00$	$0\cdots\cdots 00$	0
0/1	$0\cdots\cdots 00$	$0\cdots\cdots 01$	x_{min}
0/1	$0\cdots\cdots 00$	$0\cdots 1\cdots 0$	denormalizzato
0/1	$0\cdots\cdots 01$	$0\cdots\cdots 01$	realmin
0/1	$0\cdots 1\cdots 0$	$0\cdots 1\cdots 0$	normalizzato
0/1	$1\cdots\cdots 10$	$1\cdots\cdots 11$	realmax
0/1	$1\cdots\cdots 11$	$0\cdots\cdots 00$	Inf
0/1	$1\cdots\cdots 11$	$0\cdots 1\cdots 0$	NaN

Tabella 1: Rappresentazione interna dello *standard IEEE 754* in *doppia precisione* di MATLAB.

Per la **precisione di macchina** vale $u := \beta^{1-t}$ nel caso di troncamento e $u := \frac{\beta^{1-t}}{2}$ nel caso dell'arrotondamento. Lo *standard IEEE 754* in *doppia precisione* di MATLAB utilizza l'arrotondamento, per cui

```
>>u=2^(-53)
u =
   1.1102e-016
```

La funzione *eps* ci fornisce la distanza tra $x = 1.0$ e il successivo numero floating point ed è uguale al doppio della precisione di macchina u, ovvero $eps = \beta^{1-t} = 2^{-52} = 2u$:

```
>>eps
eps =
     2.2204e-16
```

2.5 Formati

Le 53 cifre della mantissa binaria consentono di esprimere circa 15 cifre di mantissa decimale. La risposta MATLAB mostra invece solo 5 cifre di mantissa:

```
>>pi
ans=
    3.1416
```

Per avere il risultato con più cifre decimali bisogna cambiare il formato di visualizzazione. Questo viene fatto tramite il comando

```
format <tipo di formato>
```

Il formato di default, quello in cui siamo ora, è il formato *short*. Per vedere tutte le cifre decimali occorre passare al formato *long*:

```
>>format long
>>pi
ans=
    3.14159265358979
```

Per avere i numeri in notazione scientifica normalizzata occorre passare ai formati *long e* (per vedere tutte le cifre della mantissa) o *short e* (per vedere le prime cifre della mantissa):

```
>>1/30
ans=
    0.03333333333333
>>format long e
>>1/30
ans=
    3.333333333333333e-002
>>format short e
>>1/30
ans=
    3.3333e-002
>>format short
>>1/30
ans=
    0.0333
```

2.6 Help

In MATLAB è presente un *help* in linea. Il comando

```
>>help help
```

fornisce informazioni su come usare l'help. Il comando

```
>>help <argomento>
```

fornisce informazioni su uno specifico argomento *argomento*. Ad esempio, per avere informazioni sulla funzione *sqrt* si digiti al prompt

```
>>help sqrt
```

Per avere informazioni sul comando *format* e sui diversi tipi di formato si digiti

```
>>help format
```

(Può accadere che il testo dell'help occupi più di una schermata, come ad esempio in *help +*. In questo caso per vedere una schermata alla volta si usi il comando *more on* e poi si dia di nuovo il comando di *help*).
A documentazioni più approfondite di quelle fornite dall'help in linea si accede tramite il comando *helpdesk*. Questo apre un ipertesto in cui è possibile reperire le informazioni desiderate. Il comando

```
>>doc <argomento>
```

fornisce direttamente la documentazione sull'argomento *argomento* presente nell' ipertesto.

Terminiamo la sezione sottolineando che è possibile richiamare al prompt istruzioni precedentemente digitate tramite la freccia "↑". È anche possibile richiamare solo particolari istruzioni: digitando al prompt una stringa di caratteri e poi usando la freccia si richiamano solo quelle istruzioni che cominciano con quella stringa.

3 Matrici

3.1 Costruzione di matrici

Per assegnare la matrice 3×4

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

alla variabile A, si usa l'istruzione

```
>>A=[1 2 3 4; 5 6 7 8; 9 0 1 2]
A=
    1     2     3     4
    5     6     7     8
    9     0     1     2
```

Gli elementi della matrice sono racchiusi dalle parentesi quadre; le righe sono separate l'una dall'altra da un punto e virgola e all'interno di ogni riga gli elementi sono separati da uno spazio bianco (si possono separare anche con la virgola). Inoltre è possibile separare le righe anche con un "invio":

```
>>A=[1 2 3 4
5 6 7 8
9 0 1 2]
A=
    1     2     3     4
    5     6     7     8
    9     0     1     2
```

Si noti che MATLAB è *case sensitive*, cioè distingue tra lettere maiuscole e minuscole. Così, la matrice precedente è memorizzata nella variabile A e non nella variabile a. A differenza di altri linguaggi, come FORTRAN o C, non è necessario predefinire la variabile A come matrice 3×4. Inoltre la variabile A può diventare, da una matrice 3×4, una matrice di altre dimensioni. Esempio:

```
>>A
A=
    1     2     3     4
    5     6     7     8
    9     0     1     2
>>A=[0 1; 3 1]
A=
    0     1
    3     1
```

L'elemento a_{ij} della matrice A viene denotato con $A(i,j)$.

```
>>A(2,2)
ans=
    1
```

Per modificare un elemento della matrice A, ad esempio a_{22}, non occorre riscrivere l'intera matrice ma basta l'istruzione

```
>>A(2,2)=15
A =
    0     1
    3    15
```

Invece con

```
>>A(4,4)=2
A=
    0     1     0     0
    3    15     0     0
    0     0     0     0
    0     0     0     2
```

MATLAB allarga la matrice quanto basta per sistemare un elemento alla quarta riga e quarta colonna e riempie di zeri le altre nuove entrate della matrice.

Sui vettori, essendo particolari matrici, si opera in maniera analoga. Le istruzioni

```
>>b=[1 3 5 7]
b=
    1     3     5     7
>>c=[1 3 5 7]'
c=
    1
    3
    5
    7
>>d=[2; 4; 6;  8]
d=
    2
    4
    6
    8
```

creano il vettore riga b ed i vettori colonna c e d. L'apice "′" è l'operatore di trasposizione. Più precisamente A' denota in MATLAB la matrice A^H trasposta coniugata della matrice A. La semplice trasposizione senza coniugazione degli elementi si ottiene con l'operatore ".′".

L'elemento v_i di un vettore v (riga o colonna) si indica con $v(i)$. Esempio:

```
>>b(2),c(3)
ans=
    3
ans=
    5
```

È possibile costruire matrici utilizzando blocchi già esistenti. Esempio:

```
>>B=[A c d; b 0 1]
B=
    0     1     0     0     1     2
    3    15     0     0     3     4
    0     0     0     0     5     6
    0     0     0     2     7     8
    1     3     5     7     0     1
```

Le seguenti funzioni permettono di costruire particolari matrici di dimensioni prefissate:

$eye(m,n)$	matrice $m \times n$ con 1 sulla diagonale principale e 0 fuori
$zeros(m,n)$	matrice $m \times n$ con 0 ovunque
$ones(m,n)$	matrice $m \times n$ con 1 ovunque

Chiamando le precedenti funzioni con un unico argomento si costruiscono matrici quadrate (cioè il secondo argomento è, per default, preso uguale al primo).

Le seguenti funzioni invece consentono di costruire particolari matrici a partire da una matrice data:

$tril(A,k)$	matrice ottenuta dalla matrice A azzerando gli elementi sopra la diagonale $k-$esima
$triu(A,k)$	matrice ottenuta da A azzerando gli elementi sotto la diagonale $k-$esima
$diag(u,k)$	matrice con il vettore u sulla diagonale $k-$esima e 0 fuori
$diag(A,k)$	vettore riga diagonale $k-$esima della matrice A

La diagonale k-esima, con k intero, è la diagonale (principale) se $k = 0$, la k-esima sopra-diagonale se $k > 0$, e la $|k|$-esima sotto-diagonale se $k < 0$. Chiamando le precedenti funzioni con il solo argomento matrice il secondo è, per default, preso uguale a zero.

Altre funzioni interessanti sono:

$reshape(A,p,q)$	riforma la matrice A di dimensioni $m \times n$ come matrice $p \times q$ a partire dal vettore ottenuto concatenando le colonne di A una dopo l'altra (deve essere $mn = pq$)
$size(A)$	fornisce, per la matrice A di dimensioni $m \times n$, il vettore riga $[m\ n]$; $size(A,1)$ restituisce il numero di righe, $size(A,2)$ il numero di colonne.
$length(A)$	fornisce per la matrice A di dimensioni $m \times n$ il numero $\max\{m,n\}$; applicata a vettori dà la lunghezza del vettore

Una caratteristica importante di MATLAB è la *notazione colon* (*notazione* :). La scrittura

$$a : s : b,$$

dove a, s e b sono reali, rappresenta il vettore riga di elementi $a, a+s, a+2s$ e così via fino a b. Se b non può essere raggiunto, perchè $b-a$ non è multiplo di

s, allora ci si ferma a quello immediatamente prima. Se il passo s è uguale a 1, esso può essere omesso scrivendo semplicemente $a : b$. Esempi

```
>>0:5
ans=
    0     1     2     3     4     5
>>t=0:-1:-5
t=
    0    -1    -2    -3    -4    -5
>>t=0:.5:2.25
t=
         0    0.5000    1.0000    1.5000    2.0000
```

Si è gia visto che $A(i,j)$ nel linguaggio di MATLAB sta per l'elemento a_{ij} della matrice A. Un'estensione di questa notazione è $A(u,v)$ dove $u = (u_1, ..., u_p)$ è un vettore di indici di riga della matrice A e $v = (v_1, ..., v_q)$ è un vettore di indici di colonna. La scrittura $A(u,v)$ rappresenta la matrice di dimensioni $p \times q$

$$B = \begin{pmatrix} a_{u_1 v_1} & . & . & . & a_{u_1 v_q} \\ . & & & & . \\ . & & & & . \\ . & & & & . \\ a_{u_p v_1} & . & . & . & a_{u_p v_q} \end{pmatrix}.$$

Tramite tale notazione è possibile ottenere in modo semplice sottomatrici di una matrice data. Basta specificare nei vettori u e v le righe e le colonne da selezionare per costruire la sottomatrice. Ad esempio, data la matrice

```
>>B
B=
    0     1     0     0     1     2
    3    15     0     0     3     4
    0     0     0     0     5     6
    0     0     0     2     7     8
    1     3     5     7     0     1
```

la sottomatrice individuata dalle righe 1 e 5 e dalle colonne 3, 4 e 5 è data da

```
>>B([1 5],[3 4 5])
ans=
    0     0     1
    5     7     0
```

È possibile utilizzare la notazione colon per indicare i vettori degli indici di riga e colonna. Esempio:

```
>>B(1:5,1:2:6)
ans=
    0     0     1
```

```
    3     0     3
    0     0     5
    0     0     7
    1     5     0
```

Se il simbolo ":" compare da solo esso indica tutte le righe o tutte le colonne (nell'ordine dalla prima all'ultima). Esempio:

```
>>B(:,5),B(1,:),B(:,:)
ans=
    1
    3
    5
    7
    0
ans=
    0     1     0     0     1     2
ans=
    0     1     0     0     1     2
    3    15     0     0     3     4
    0     0     0     0     5     6
    0     0     0     2     7     8
    1     3     5     7     0     1
```

Ancora, se ":" compare come unico argomento della notazione, si ottiene il vettore colonna concatenazione delle colonne della matrice

```
>>B(:)
ans=
    0
    3
    0
    .
    .
    .
    6
    8
    1
```

Volendo modificare nella matrice gli elementi che formano una particolare sottomatrice si può usare la precedente notazione a sinistra dell'operatore di assegnamento. Esempio

```
>>B([1 2],[1 2])=eye(2)
B=
    1     0     0     0     1     2
    0     1     0     0     3     4
    0     0     0     0     5     6
```

```
    0     0     0     2     7     8
    1     3     5     7     0     1
```

Con la notazione introdotta si possono anche permutare righe o colonne di una matrice data. Ad esempio:

```
>>C=eye(5)
C=
    1     0     0     0     0
    0     1     0     0     0
    0     0     1     0     0
    0     0     0     1     0
    0     0     0     0     1
>>C=C(5:-1:1,:)
C=
    0     0     0     0     1
    0     0     0     1     0
    0     0     1     0     0
    0     1     0     0     0
    1     0     0     0     0
```

3.2 Operazioni su matrici

Gli operatori $+$, $-$ e `*` denotano le usuali operazioni di addizione, sottrazione e moltiplicazione di matrici. Esempio:

```
>>A=[0 1; 2 1]; B=[3 -1; -1 0];
>>C=A+B, D=A*B
C=
    3     0
    1     1
D=
   -1     0
    5    -2
>>b=[1 3 5 7]; d=[2 4 6 8]';
>>s=b*d, S=d*b
s=
  100
S=
    2     6    10    14
    4    12    20    28
    6    18    30    42
    8    24    40    56
```

Gli operatori $+$, $-$ e `*` possono esere usati anche con uno scalare e una matrice, nel qual caso l'operazione con lo scalare viene fatta su ogni elemento della matrice. Esempio:

```
>>A, 1+A, A*2
A=
    0     1
    2     1
ans=
    1     2
    3     2
ans=
    0     2
    4     2
```

Sono presenti, per matrici due operatori di divisione: la divisione sinistra "\" e la divisione destra "/".

- La divisione sinistra è definita come segue: se A è una matrice $m \times n$ e B è una matrice $m \times p$, allora $A \setminus B$ è una matrice X di dimensione $n \times p$ tale che $AX = B$.
- La divisione destra è definita come segue: se A è una matrice $n \times p$ e B è una matrice $m \times p$, allora B/A è una matrice X di dimensione $m \times n$ tale che $XA = B$.

Si noti che per scalari $b\backslash a = a/b$. In generale per matrici $A\backslash B$ è diverso da B/A essendo il prodotto di matrici non commutativo. Con l'operatore "\" si possono risolvere sistemi lineari di equazioni. Se M è una matrice $m \times m$ quadrata non singolare e b è un vettore $m \times 1$, allora la soluzione del sistema lineare

$$Mx = b$$

è data da $x = M\backslash b$. Esempio: dato il sistema lineare

$$\begin{cases} x + y - z = 3 \\ 2x - 5y + 7z = -3 \\ -x + 3y - 6z = 42 \end{cases}$$

la sua soluzione si ottiene tramite le istruzioni

```
>>M=[1 1 -1; 2 -5 7; -1 3 -6];
>>b=[3 -3 42]';
>>x=M\b
x=
   7.8462
 -26.3077
 -21.4615
```

Sono definite le espressioni A/α e $\alpha\backslash A$ con A matrice e α scalare: l'operazione con lo scalare viene fatta su ogni elemento della matrice. Esempio

```
>>A, A/2, 2\A
A=
    0     1
    2     1
ans=
         0    0.5000
    1.0000    0.5000
ans=
         0    0.5000
    1.0000    0.5000
```

L'elevamento a potenza intera di matrici quadrate viene effettuata con l'operatore "^". Esempio:

```
>>A, A^2, A^(-1)
A=
    0     1
    2     1
ans=
    2     1
    2     3
ans=
   -0.5000    0.5000
    1.0000         0
```

Si noti che l'ultima matrice in *ans* è l'inversa di A che può essere ottenuta anche con la funzione *inv*. Esempio:

```
>>inv(A)
ans=
   -0.5000    0.5000
    1.0000         0
```

Gli operatori sopra descritti agiscono elemento per elemento se sono preceduti dal punto ".". Così, assegnate le matrici A e B delle stesse dimensioni e $\Diamond \in \{+, -, *, /, \backslash, \hat{}\}$, la matrice $K = A.\Diamond B$ ha elementi $k_{ij} = a_{ij} \Diamond b_{ij}$. Analogamente se A è una matrice e α è uno scalare, allora la matrice $K = \alpha.\Diamond A$ ($K = A.\Diamond \alpha$) ha elementi $k_{ij} = \alpha \Diamond a_{ij}$ ($k_{ij} = a_{ij} \Diamond \alpha$). Esempio:

```
>>A, B
A=
    0     1
    2     1
B=
    3    -1
   -1     0
>>A.*B, A.^B
ans=
```

```
    0    -1
   -2     0
ans=
        0    1.0000
   0.5000    1.0000
>>1./A, 2.^A, A.^2
Warning: Divide by zero.
ans=
      Inf    1.0000
   0.5000    1.0000
ans=
    1     2
    4     2
ans=
    0     1
    4     1
```

Una funzione scalare f, come ad esempio *sqrt, abs, sin, cos, tan,* ecc., può essere applicata anche a matrici nel qual caso agisce su ogni elemento: cioè $f(A)$ è la matrice di elementi $f(a_{ij})$. Esempio:

```
>>A, sqrt(A), sin(A)
A=
    0     1
    2     1
ans=
        0    1.0000
   1.4142    1.0000
ans=
        0    0.8415
   0.9093    0.8415
```

Una funzione che agisce su vettori (funzione vettoriale), come ad esempio,

max	massimo elemento
min	minimo elemento
sum	somma degli elementi
prod	prodotto degli elementi
mean	media degli elementi
sort	vettore ottenuto ordinando in modo ascendente gli elementi

può essere applicata anche ad una matrice, nel qual caso agisce su ogni colonna:

```
>>B, sum(B), max(max(B)), sort(B)
B=
    3    -1
   -1     0
```

```
ans=
    2    -1
ans=
    3
ans=
   -1    -1
    3     0
```

4 Workspace di MATLAB

Lo spazio di lavoro di MATLAB, detto anche *workspace*, è l'insieme delle variabili create durante la sessione. Supponiamo che siano stati creati i vettori x e y e le matrici A e B mediante le istruzioni di assegnamento

```
>>x=[1 -2 pi 9.8];y=[-1.1 0 3]';
>>A=eye(3);B=[i 1+2i;0 1];
```

Per vedere le variabili del workspace si usa il comando *who*:

```
>>who
Your variables are:
A B x y
```

Per avere informazioni più dettagliate sul workspace si usa *whos*:

```
>>whos
Name      Size            Bytes  Class
A         3x3             72     double array
B         2x2             64     double array (complex)
x         1x4             32     double array
y         3x1             24     double array
Grand total is 20 elements using 192 bytes
```

La prima colonna contiene il nome della variabile, la seconda colonna le dimensioni della matrice contenuta nella variabile, la terza il numero di byte occupati, dato da "numero di elementi della matrice" $\times$ 8 se la matrice è reale e "numero di elementi della matrice" $\times$ 16 se la matrice è complessa (di un numero complesso si memorizza la parte reale e la parte immaginaria), e la quarta contiene il tipo di dato che è sempre un array di numeri in doppia precisione e inoltre segnala se la matrice è complessa.

Per cancellare una o più variabili dal workspace, si usa il comando

```
>>clear <lista>
```

che elimina le variabili specificate in *lista*. Le variabili in *lista* devono essere separate da spazi. Esempio:

```
>>who
Your variables are:
A B x y
>>clear x y
>>who
Your variables are:
A B
```

Se non vengono specificate delle variabili da cancellare *clear* cancella tutte le variabili dal workspace.

Se fossimo interessati a mantenere alcune delle variabili del workspace per successive sessioni, la seguente istruzione

```
>>save <nomefile> <lista>
```

salva nel file *nomefile.mat* le variabili (separate da spazi) specificate in *lista.* Se nessuna variabile da salvare è specificata, allora tutte le variabili vengono salvate. Se nessuna variabile e nessun file sono specificati *save* salva nel file *matlab.mat* tutte le variabili. In una successiva sessione, per caricare le variabili memorizzate nel file *nomefile.mat*, si utilizza l'istruzione

```
>>load <nomefile> <lista>
```

dove *lista* contiene le variabili da recuperare (separate da spazi). Se nessuna variabile da recuperare è specificata, allora tutte le variabili vengono caricate. Se nessuna variabile e nessun file sono specificati *load* recupera nel file *matlab.mat* tutte le variabili. Esempio

```
>>who
Your variables are:
A B
>>save var1
>>clear
>>who
Your variables are:

>>load var1
>>who
Your variables are:
A B
```

5 Grafica bidimensionale

La finestra in cui si trova il prompt di MATLAB è nota come *finestra di comando.* MATLAB ha anche la possibiltà di lavorare con delle *finestre grafiche* sulle quali si possono fare disegni bidimensionali o tridimensionali. Una finestra grafica viene aperta con il comando $figure$ (in ambiente Windows anche con File-New-Figure). Si esegua

```
>>figure
```

Tale finestra è la finestra grafica corrente, cioè quella in cui disegneranno le istruzioni di MATLAB per la grafica. Ogni finestra grafica è dotata di un numero (che si legge in alto sul contorno). Per far diventare la finestra grafica di numero N la finestra corrente si usa il comando $figure\,(N)$ (in ambiente Windows è sufficiente cliccare con il mouse sulla finestra).

L'istruzione per fare disegni bidimensionali è

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

dove $x = (x_1, ..., x_n)$, $y = (y_1, ..., y_n)$ sono vettori della stessa dimensione n. L'effetto è che sulla finestra grafica corrente (se non ci sono finestre grafiche ne viene aperta una) viene introdotto un sistema di cooordinate cartesiane bidimensionale e si congiunge (x_1, y_1) con (x_2, y_2), (x_2, y_2) con (x_3, y_3), ..., (x_{n-1}, y_{n-1}) con (x_n, y_n) con una linea continua di colore blu. Si provi

```
>>plot([0 0 1 2 2],[0 1 1 1 2])
```

Le modalità di rappresentazione dei punti possono essere cambiate dando un terzo argomento stringa s a *plot* che individua la modalità di rappresentazione. L'istruzione *plot* nella sua forma completa è

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

Per vedere tutte le possibili modalità di rappresentazione si veda l'help in linea su *plot*. Esempio:

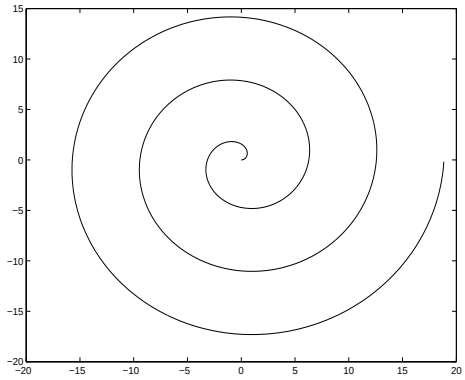
```
>>plot([0 0 1 2 2],[0 1 1 1 2],'--')
>>plot([0 0 1 2 2],[0 1 1 1 2],'r+')
>>plot([0 0 1 2 2],[0 1 1 1 2],'ko')
```

L'istruzione *plot* può essere usata per tracciare curve bidimensionali e grafici di funzioni reali di una variabile reale. Si voglia, ad esempio, tracciare la curva bidimensionale detta *spirale di Archimede*, che è il moto di un punto materiale in moto rettilineo uniforme di velocità v su una semiretta che esce dall'origine e ruota attorno ad esso con velocità angolare ω. La curva ha le equazioni parametriche

$$\begin{cases} x = vt\cos\omega t \\ y = vt\sin\omega t \end{cases}.$$

Supponendo $v = 1$ e $\omega = 1$, il disegno di $n = 3$ giri di spirale si ottiene con le seguenti istruzioni

```
>>v=1; omega=1;
>>n=3; h=0.01;
>>t=0:h:2*pi*n/omega;
>>x=v*t.*cos(omega*t);
>>y=v*t.*sin(omega*t);
>>plot(x,y)
```



Si noti che h è il passo di campionamento del parametro temporale t.

Il grafico della funzione $f(x) = \sin x$ nell'intervallo $[0, 2\pi]$ si ottiene con le istruzioni

```
>>h=0.01;
>>x=0:h:2*pi;
>>y=sin(x);
>>plot(x,y)
```

h è il passo di campionamento della variabile indipendente x. L'istruzione *plot* può fare più disegni sulla finestra grafica. Basta dare più coppie (x, y) (o terne (x, y, s)) come argomento a *plot*. Ad esempio se si vuole tracciare in $[0, 2\pi]$ il grafico del seno e del coseno si possono usare le istruzioni

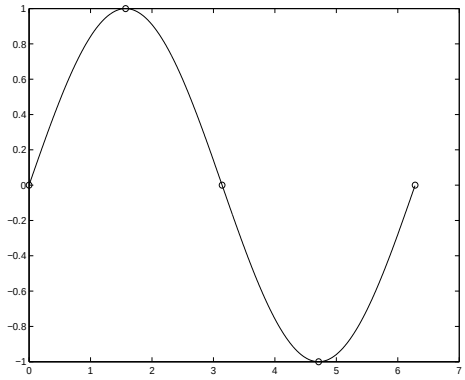
```
>>h=0.01;
>>x=0:h:2*pi;
>>y=sin(x);
>>z=cos(x);
>>plot(x,y,x,z)
```

Si noti come, automaticamente, MATLAB disegni i due grafici con colori diversi. Volendo, invece, evidenziare nel grafico del seno i punti con ascissa multipla di $\frac{\pi}{2}$, si possono usare le istruzioni

```
>>h=0.01;
>>x=0:h:2*pi;
>>y=sin(x);
>>X=[0,pi/2,pi,3/2*pi,2*pi];
>>Y=sin(X);
>>plot(x,y,X,Y,'o')
```

5.1 Gestione della finestra grafica

L'istruzione *plot*, prima di disegnare sulla finestra grafica corrente, cancella ogni disegno preesistente. Per evitare questo si usa il comando *hold on*. Dopo



la sua esecuzione, sulla finestra grafica corrente viene mantenuto ogni disegno preesistente. Volendo tracciare in $[0, 2\pi]$ il grafico del seno e del coseno si possono usare allora anche le istruzioni

```
>>h=0.01;
>>x=0:h:2*pi;
>>y=sin(x);
>>plot(x,y);
>>hold on
>>z=cos(x);
>>plot(x,z,'r')
```

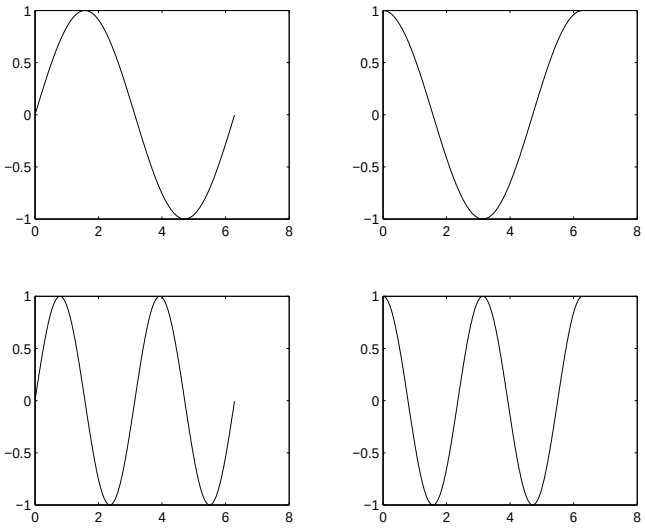
Per tornare alla situazione in cui vengono cancellati i disegni preesistenti si usa *hold off*.

E' possibile anche spezzare la finestra grafica corrente in una matrice $m \times n$ di sottofinestre grafiche. Il comando

```
subplot(m,n,k)
```

spacca la finestra corrente (ne crea una se non esistono finestre grafiche) in una matrice $m \times n$ e fa diventare la k-esima, contata seguendo le righe, la finestra corrente. Per cambiare la sottofinestra corrente dalla k-esima alla l-esima si usa *subplot* (m, n, l).

```
>>x=0:h:2*pi;
>>subplot(2,2,1)
>>plot(x,sin(x))
>>subplot(2,2,2)
>>plot(x,cos(x))
>>subplot(2,2,3)
>>plot(x,sin(2*x))
>>subplot(2,2,4)
>>plot(x,cos(2*x))
```



È possibile aggiungere del testo ad un grafico tramite le istruzioni *xlabel, ylabel, title* and *text* (si veda l'help in linea). Per stampare il disegno sulla finestra grafica corrente si usa il comando *print* senza argomenti (si veda comunque l'help in linea).

6 Valori di verità ed espressioni logiche

In MATLAB (come nel linguaggio C) ogni valore scalare reale ha un valore di verità: precisamente lo zero è falso e ogni numero non zero è vero. (Per scalari complessi il valore di verità viene determinato dalla parte reale). Le *espressioni logiche* sono costrutti sintattici che hanno come risultato valori di verità. Esse sono costruite a partire da

- *operatori relazionali* e *funzioni logiche* che producono valori di verità, e
- *operatori* (o *connettivi*) *logici* che associano a valori di verità altri valori di verità.

Gli operatori relazionali sono

$<$	minore
$>$	maggiore
$<=$	minore o uguale
$>=$	maggiore o uguale
$==$	uguale
$\tilde{}=$	non uguale

Essi sono definiti per scalari reali (su complessi si confrontano le parti reali) e producono 1 per vero e 0 per falso. Quando applicati a matrici delle stesse

dimensioni, o tra una matrice e uno scalare, agiscono elemento per elemento producendo una matrice di valori di verità. Ad esempio:

```
>>1>5
ans=
    0
>>A, B, A~=B, A>1
A=
    0     1
    2     1
B=
    3    -1
   -1     0
ans=
    1     1
    1     1
ans=
    0     0
    1     0
```

Le funzioni logiche agiscono su vettori e sono

any { 1 se esiste almeno un elemento vero (non zero) nel vettore; 0 altrimenti

all { 1 se tutti gli elementi del vettore sono veri (non zero); falso altrimenti

Quando applicati a matrici le funzioni logiche agiscono colonna per colonna producendo un vettore riga di valori di verità.

Gli operatori logici sono

~	not
&	and
|	or
xor	or esclusivo

con ordine di precedenza dall'alto verso il basso. Essi sono definiti per scalari reali (su complessi si opera con le parti reali) e producono 1 per vero e 0 per falso. Quando applicati a matrici delle stesse dimensioni, o tra una matrice e uno scalare, agiscono elemento per elemento producendo una matrice di valori di verità. Ad esempio:

```
>>1<2|1==2|1>2
ans=
    1
>>x=[1 0 0 1];
>>any(x)&~all(x)
```

```
ans=
    1
>>A,any(~all(A))
A=
    0     1
    2     1
ans=
    1
```

Infine segnaliamo la funzione $find$. Se X è una matrice, $find\,(X)$ restituisce il vettore colonna degli indici degli elementi veri (non zero) nel vettore $X\,(:)$. Il seguente esempio trova in un vettore random gli indici degli elementi compresi tra 0.25 e 0.75:

```
>>x=rand(6,1)
x=
   0.9501
   0.2311
   0.6068
   0.4860
   0.8913
   0.7621
>>ind=find(x>0.25&x<0.75)
ind=
    3
    4
```

Se invece si richiedono i valori di tali elementi, allora basta digitare:

```
>>val=x(find(x>0.25&x<0.75))
val=
    0.6068
    0.4860
```

7 Programmazione

7.1 M-file

Come si è già accennato nella premessa, MATLAB non è solo un interprete di istruzioni, ma dà anche la possibilità all'utente di scrivere propri programmi. I programmi MATLAB sono detti *M-file* e sono file di testo che vanno memorizzati con estensione *.m* (da cui il nome M-file). Un M-file può essere creato con un *editor* qualsiasi. Una volta che un M-file è stato creato e memorizzato in una directory, è opportuno aggiungere la directory al path di MATLAB. Il path è l'insieme di directory in cui MATLAB cerca gli M-file che deve eseguire, dopo aver cercato nella directory corrente. Per vedere il path di MATLAB si usa il comando *path*, per aggiungere una directory al path si usa il comando

addpath, per rimuovere il comando *rmpath* (per dettagli si veda l'help in linea). In ambiente Windows, MATLAB ha un proprio editor/debugger che può essere richiamato sciegliendo nel menù a tendina File-New-M-file (per crearne uno nuovo) oppure File-Open (per aprirne uno già esistente). Per vedere il path in ambiente Windows si usa File-Set Path che apre il *path browser*.

Vi sono due tipi di M-file: gli *script* e le *function*.

7.1.1 M-file script

Uno *script* è un file di testo dove ogni riga contiene una (o più di una) istruzione MATLAB. Per far eseguire lo script si scrive al prompt il nome del file, senza l'estensione .m, e si preme il tasto "Invio". MATLAB eseguirà automaticamente le righe del file dalla prima all'ultima come se fossero state introdotte, in quell'ordine, manualmente al prompt.

Come esempio si calcolino le radici dell'equazione di secondo grado

$$ax^2 + bx + c = 0$$

con $a = 5$, $b = 2$ e $c = 1$. Il file *primo.m* esegue tale compito e contiene le istruzioni

```
a=5, b=2, c=1
delta=b^2-4*a*c;
x1=(-b+sqrt(delta))/(2*a)
x2=(-b-sqrt(delta))/(2*a)
```

Si ottiene dunque:

```
>>who
Your variables are:

>>primo
a=
    5
b=
    2
c=
    1
x1=
 -0.2000 + 0.4000i
x2=
 -0.2000 - 0.4000i
>>who
Your variables are:
a      b      c      delta  x1     x2
```

Si noti come alla fine dell'esecuzione dello script il workspace inizialmente vuoto contenga tutte le variabili create durante l'esecuzione.

7.1.2 M-file function

Una *function* è un M-file che accetta valori in input e restituisce un valore in output. Una funzione ha la forma

```
function <var. output>=<nome funzione>(<lista var. input>)
<linee di commento>
<corpo della funzione>
```

Ad esempio

```
function f=fatt(n)
% FATT fattoriale
% FATT(n) e' il fattoriale di n
f=prod(1:n);
```

La prima linea è la *linea di definizione* della funzione. Essa comincia con la parola riservata *function* (che serve a MATLAB per capire che si tratta di una funzione e non di uno script), segue poi la variabile di output, il nome della funzione (che non deve necessariamente essere uguale al nome del file) ed infine la lista delle variabili di input che vanno separate da virgole. Le successive sono eventuali *linee di commento*. Il simbolo % è usato per i commenti in quanto MATLAB ignora, su una linea, tutto ciò che lo segue. Vi è infine il *corpo della funzione* che consiste in una sequenza di istruzioni MATLAB esattamente come in uno script. Una chiamata di funzione va fatta scrivendo il nome del file (e non il nome della funzione) senza l'estensione .m e poi gli argomenti di input tra parentesi tonde. Dal momento che si effettua una chiamata di funzione per ottenere un valore, questa si troverà sempre dentro un'espressione, come, ad esempio, in

```
>>fatt(90)/(fatt(12)*fatt(78))
ans=
 2.7390e+014
```

Una funzione può anche restituire più valori di output. In tal caso la sua linea di definizione è

```
function [<lista var. output>]=<nome funzione>(<lista var. input>)
```

I valori restituiti sono quelli delle variabili specificate in *lista variabili output* alla fine dell'esecuzione della funzione. Si consideri, ad esempio, la funzione

```
function [x1,x2]=secondo(a,b,c)
% SECONDO calcola le radici x1 e x2
% dell'equazione di secondo grado
% ax^2+bx+c=0.
delta=b^2-4*a*c;
x1=(-b+sqrt(delta))/(2*a);
x2=(-b-sqrt(delta))/(2*a);
```

che calcola le radici dell'equazione di secondo grado

$$ax^2 + bx + c = 0$$

con a, b e c forniti in input. La chiamata

```
>>[x1,x2]=secondo(5,2,1)
```

restituisce entrambe le variabili di output

```
x1=
 -0.2000 + 0.4000i
x2=
 -0.2000 - 0.4000i
```

Se invece si esegue

```
>>secondo(5,2,1)
```

si ottiene (per default) solamente la prima variabile della lista di output:

```
ans=
 -0.2000 + 0.4000i
```

Molte istruzioni predefinite restituiscono più valori di output. Ad esempio la funzione *max* restituisce oltre all'elemento massimo anche l'indice di tale elemento (si veda l'help in linea)

```
>>x=[1 3 -2 0];
>>max(x)
ans=
    3
>>[m,i]=max(x)
m=
    3
i=
    2
```

Quando MATLAB incontra una chiamata di funzione in un'espressione fa quanto segue:

- crea un apposito workspace locale per l'esecuzione della funzione;
- crea in questo workspace le variabili di input specificate in *lista variabili di input* inizializzandole con i valori corrispondenti forniti nella chiamata;
- esegue le istruzioni nel corpo della funzione dalla prima all'ultima;
- quando l'esecuzione delle istruzioni termina, MATLAB ritorna, come valore della funzione, il contenuto della lista delle variabili di output specificata in *lista var. output* e distrugge il workspace locale.

Ad esempio:

```
>>clear
>>who
Your variables are:

>>[x1,x2]=secondo(5,2,1);
>>who
Your variables are:
x1 x2
```

Il precedente esempio dimostra come l'esecuzione della funzione sia fatta in un workspace locale, dal momento che, nel workspace principale, non c'è traccia delle variabili create durante l'esecuzione della chiamata di *secondo*, come ad esempio la variabile *delta*.

Una volta che una funzione è stata creata è possibile chiamare l'help in linea con argomento il nome della funzione. Esempio

```
>> help secondo
    SECONDO calcola le radici x1 e x2
    dell'equazione di secondo grado
    ax^2+bx+c=0.
```

L'help in linea mostra le righe di commento che seguono la linea di definizione della funzione.

A differenza degli script, che vengono interpretati, le funzioni sono invece compilate. La prima volta che una funzione è chiamata nel corso di una sessione viene compilato il programma sorgente e le successive chiamate della funzione usano il programma oggetto risultato della compilazione. Tutte le funzioni predefinite che si trovano in MATLAB sono M-file funzioni. Per alcune è presente il sorgente mentre per altre (le più frequentemente usate) è presente solo il programma oggetto. Il comando

```
>>type <nome>
```

permette di vedere tutto il contenuto di un M-file. Esempio

```
>>type rank
function r = rank(A,tol)
%   RANK   Matrix rank.
%   RANK(A) provides an estimate of the number of linearly
%   independent rows or columns of a matrix A.
%   RANK(A,tol) is the number of singular values of A
%   that are larger than tol.
%   RANK(A) uses the default tol = max(size(A)) * norm(A) * eps.

%   Copyright 1984-2001 The MathWorks, Inc.
%   $Revision: 5.10 $  $Date: 2001/04/15 12:01:33 $
```

```
s=svd(A);
if nargin==1
   tol=max(size(A)')*max(s)*eps;
end
r=sum(s>tol);

>>type rand
rand is a built-in function.
```

Nel secondo esempio è presente solo il programma oggetto.

Ogni qualvolta si crea una nuova funzione questa si va ad aggiungere alle funzioni predefinite estendendo così le capacità di MATLAB. Ad esempio non esiste in MATLAB una funzione predefinita che calcola il fattoriale di un numero e nemmeno esiste una funzione specifica per le radici delle equazioni di secondo grado (ma esiste per le radici di polinomi in generale). Dopo la creazione di *fatt* e *secondo*, MATLAB ha delle funzioni (che non si differenziano da quelle predefinite) per svolgere tali compiti. Anche le toolbox di MATLAB a cui si è accennato nella premessa non sono altro che raccolte di M-file che svolgono particolari compiti.

7.2 Istruzioni di selezione e di iterazione

In MATLAB è presente l'istruzione di selezione del controllo *if*. La forma dell'istruzione è

```
if <condizione if>
    <istruzioni if>
elseif <condizione 1>
    <istruzioni 1>
elseif <condizione 2>
    <istruzioni 2>
elseif <condizione n>
    <istruzioni n>
else <istruzioni else>
end
```

La semantica dell'istruzione è la seguente. Si valuta l'espressione logica *condizione if*: se è vera, allora sono eseguite le istruzioni *istruzioni if* e si esce proseguendo dopo *end*; se è falsa si valuta *condizione 1*. Se questa è vera sono eseguite le istruzioni *istruzioni 1* e si esce proseguendo dopo *end*, se è falsa si valuta *condizione 2* e così via.... Se nessuna tra *condizione 1*, *condizione 2*...*condizione n* è vera si eseguono le istruzioni *istruzioni else*. Le parti *elseif* e *else* possono non comparire: possiamo cioè avere istruzioni *if* del tipo

```
if <condizione if>
    <istruzioni if>
```

```
else <istruzioni else>
end
```

e

```
if <condizione if>
    <istruzioni if>
end
```

Le istruzioni di iterazione del controllo sono l'istruzione *while* e l'istruzione *for*. La forma dell'istruzione *while* è

```
while <condizione>
    <istruzioni>
end
```

La semantica dell'istruzione è la seguente. Si continua a valutare l'espressione logica *condizione* e ad eseguire le successive istruzioni *istruzioni* fino a quando non la si trova falsa. La forma dell'istruzione *for* è

```
for <variabile>=<espressione>
    <istruzioni>
end
```

La semantica dell'istruzione è la seguente. Si valuta l'espressione *espressione*: il risultato è una matrice. Si assegnano alla variabile *variabile* successivamente le colonne della matrice risultato, dalla prima all'ultima, e per ciascuna di queste assegnazioni si eseguono le istruzioni *istruzioni*. Si noti che il classico ciclo for in cui *variabile* assume i valori da a a b con passo s si ottiene con

```
for <variabile>=a:s:b
    <istruzioni>
end
```

L'istruzione *break* se incontrata all'interno di un ciclo *while* o *for* causa l'immediata uscita dal ciclo. Questa istruzione permette la costruzione di iterazioni del tipo

```
while <condizione 1>
    <istruzioni 1>
    if <condizione 2>
        break
    end
    <istruzione 2>
end
```

Queste iterazioni permettono di uscire dal ciclo in un punto qualsiasi e non solo all'inizio. L'istruzione *return* se incontrata nel corpo di una funzione causa la terminazione della funzione stessa e il ritorno dei valori di output anche se non è stata raggiunta l'ultima istruzione. Di seguito si hanno esempi di funzioni che usano if, $while$ e for. La function

```
function f=fattr(n)
% FATT fattoriale calcolato in modo ricorsivo
if n<=1
    f=1;
else f=n*fattr(n-1);
end
```

calcola il fattoriale in modo ricorsivo: la function $fattr$ richiama se stessa. Bisogna prestare attenzione all'uso delle funzioni ricorsive in MATLAB in quanto l'apertura di parecchi workspace locali per l'esecuzione delle chiamate può riempire la porzione di memoria che MATLAB ha a disposizione.

Supponiamo di non conoscere il numero di bit riservati da MATLAB per la mantissa della rappresentazione floating point dei numeri reali e di sapere però che MATLAB effettua l'arrotondamento per rappresentare i numeri reali come numeri di macchina. La seguente funzione (priva di argomenti di input) calcola la precisione di macchina come la metà della distanza tra 1 e il successivo numero di macchina, e il numero di bit riservati per la mantissa:

```
function [u,t]=prec
% PREC calcolo della precisione di macchina
u=1;
t=0;
while 1+u>1
    u=u/2;
    t=t+1;
end
```

In MATLAB la precisione di macchina è la metà del valore della costante *eps*.

```
>>[u,t]=prec
u=
   1.1102e-016
t=
   53
>>eps
ans=
   2.2204e-016
```

La function

```
function Y=tab_rank_magic(n)
% TAB_RANK_MAGIC rango delle matrici magiche
Y=zeros(n,2);
for k=1:n
    Y(k,1)=k;
    Y(k,2)=rank(magic(k));
end
```

calcola il rango delle prime n matrici magiche. Ricordiamo che la matrice magica di ordine k è quella matrice con elementi i numeri naturali da 1 a k^2 disposti in modo tale che la somma degli elementi su ogni riga, su ogni colonna e sulla diagonale principale sia sempre lo stessa. La matrice magica di ordine k è ottenuta con $magic(k)$.

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

7.3 Stringhe

Una *stringa* in MATLAB è una sequenza finita di caratteri racchiusi tra apici. Ad esempio

```
'Buongiorno Maria.'
```

è una stringa. Le stringhe possono essere assegnate a variabili

```
>>a='Buongiorno Maria.'
a=
    Buongiorno Maria.
>>whos
Name      Size            Bytes  Class
a         1x17               34  char array
Grand total is 17 elements using 34 bytes
```

Si noti che per MATLAB una stringa è un vettore riga di caratteri e ogni carattere richiede 2 byte di memoria. Si possono costruire anche matrici di caratteri come in

```
>>A=['Rosso di sera     '; 'bel tempo si spera']
A=
Rosso di sera
bel tempo si spera
>>B=[a;a;a]
B=
Buongiorno Maria.
Buongiorno Maria.
Buongiorno Maria.
```

Bisogna fare attenzione a collocare lo stesso numero numero di caratteri su ogni riga (per questo ci sono caratteri blank nella prima riga della prima assegnazione). La funzione logica *ischar* restituisce vero se il suo argomento è una matrice di caratteri, falso altrimenti.

```
>>ischar(A)
ans=
1
>>ischar(1)
ans=
0
```

La codifica di memoria di un carattere è un valore intero ASCII a 16 bit. La funzione *double* converte una matrice di caratteri nella corrispondente matrice dei valori interi ASCII memorizzati però a 64bit in doppia precisione

```
>>x=double(a), X=double(A)
x=
 Columns 1 through 11
   66   117   111   110   103   105   111   114   110   111    32
 Columns 12 through 17
   77    97   114   105    97    46
X=
 Columns 1 through 11
   82   111   115   115   111    32   100   105    32   115   101
   98   101   108    32   116   101   109   112   111    32   115
 Columns 12 through 18
  114    97    32    32    32    32    32
  105    32   115   112   101   114    97
>>whos
Name      Size            Bytes  Class
A         2x18               72  char array
B         3x17              102  char array
X         2x18              288  double array
```

```
a         1x17              34  char array
x         1x17             136  double array
Grand total is 181 elements using 686 bytes
```

La funzione inversa della *double* è la *char* che converte invece i valori interi a 64 bit in caratteri

```
>>char(1:256)
ans=
..........
```

La funzione $num2str$ fornisce la stringa di rappresentazione decimale in formato *short* e del numero x; $num2str\,(x, N)$ fornisce la stringa di rappresentazione decimale di x con N cifre. Esempio

```
>>num2str(pi)
ans=
3.1416
>>num2str(pi,10)
ans=
3.141592654
>>ischar(ans)
ans=
    1
```

L'operazione di *concatenazione* di stringhe si esegue concatenando i vettori riga oppure usando la funzione *strcat*

```
>>b=[a,' Come stai?']
b=
Buongiorno Maria. Come stai?
>>b=strcat(a,' Come stai?')
b=
Buongiorno Maria. Come stai?
```

7.4 Le funzioni eval, feval e inline

La funzione

```
eval(<espressione>)
```

dove *espressione* è una stringa rappresentante una espressione MATLAB, restituisce il valore dell'espressione. Esempio

```
>>x=2;
>>eval('x^2-1'), eval('sqrt(x)-1')
ans=
    3
ans=
   0.4142
```

La funzione *eval* si usa in funzioni aventi come argomenti di input funzioni matematiche definite tramite espressioni. Ad esempio la seguente funzione riceve in ingresso una funzione data tramite un'espressione nella variabile x e restituisce il valore di questa funzione in 0.

```
function y=F1(f)
x=0;
y=eval(f);
```

Si ha

```
>>F1('x'), F1('cos(x)')
ans=
    0
ans=
    1
```

La funzione

```
feval(<nome_M-file>,<lista_argomenti>)
```

dove $nome\_M-file$ è una stringa, restituisce il valore dell'M-file funzione di nome $nome\_M-file.m$ con argomenti di input $lista\_argomenti$. Ad esempio:

```
>>feval('sin',pi/2), feval('fatt',5)
ans=
  1
ans=
  120
```

La funzione $feval$ si usa in funzioni aventi come argomenti di input altri M-file funzioni. Ad esempio la seguente funzione riceve in ingresso una M-file funzione con un argomento scalare di input e restituisce il valore di questa funzione in 0.

```
function y=F2(f)
y=feval(f,0);
```

Si ha

```
>>F2('sin'), F2('fatt')
ans=
    0
ans=
    1
```

Un'alternativa per valutare all'interno di una *function* altre funzioni passate come input è quella di definire queste ultime mediante l'istruzione *inline*, ad esempio:

```
>> f=inline('x^2+2')
f=
     Inline function:
     f(x) = x^2+2
```

Gli oggetti *inline* possono essere valutati all'interno di una *function* nel seguente modo:

```
function y=F3(f)
y=f(0);
```

```
>>y=F3(f)
y=
    2
```

Si veda l'help in linea di MATLAB per ulteriori informazioni sul comando *inline*.

7.5 Input e output

Per ricevere input dall' esterno durante l'esecuzione di un programma MATLAB si usano istruzioni della forma

```
<variabile>=input(<stringa>)
```

La funzione *input* mostra in output sullo schermo la stringa *stringa* e attende che venga digitato un valore da assegnare alla variabile *variabile* . Esempio:

```
>>a=input('Dammi il numero: a= ')
Dammi il numero: a=
```

Inserendo dopo l'uguale il numero 5 e dando "Invio" si ottiene

```
>>a=
5
```

Per dare output all'esterno durante l'esecuzione di un programma si usa l'istruzione

```
disp(<stringa>)
```

che mostra in output sullo schermo la stringa *stringa*. L'istruzione *disp* può anche essere usata con argomento una matrice di numeri nel qual caso mostra in output sullo schermo la matrice

```
>>disp('Buongiorno')
Buongiorno
>>disp(eye(5))
    1     0     0     0     0
    0     1     0     0     0
    0     0     1     0     0
    0     0     0     1     0
    0     0     0     0     1
```

Citiamo infine l'istruzione *pause* che, quando incontrata in un programma, blocca l'esecuzione fino a che non viene premuto un tasto qualsiasi. Invece $pause(n)$ blocca l'esecuzione per n secondi. Come esempio dell'uso di queste istruzioni si esegua lo script che mostra sullo schermo una tabella delle prime 10 potenze di un numero fornito in input

```
alpha=input('Dammi la base: base= ');
for k=1:10
    p=alpha^k;
    disp(['esponente= ',num2str(k),'potenza= ',num2str(p)])
pause
end
```

Si ottiene:

```
>>alpha
Dammi la base: base= 3
esponente= 1, potenza= 3
esponente= 2, potenza= 9
esponente= 3, potenza= 27
esponente= 4, potenza= 81
esponente= 5, potenza= 243
esponente= 6, potenza= 729
esponente= 7, potenza= 2187
esponente= 8, potenza= 6561
esponente= 9, potenza= 19683
esponente= 10, potenza= 59049
```

8 Grafica tridimensionale

L'analogo tridimensionale del plot è l'istruzione *plot*3. L'istruzione

```
plot3(x,y,z,s)
```

dove $x = (x_1, \ldots, x_n)$, $y = (y_1, \ldots, y_n)$ e $z = (z_1, \ldots, z_n)$ sono vettori della stessa dimensione n e s è una stringa, introduce sulla finestra grafica corrente un sistema di cooordinate cartesiane tridimensionale e disegna i punti $(x_1, y_1, z_1), \ldots, (x_n, y_n, z_n)$ secondo le modalità individuate da s. Si provi

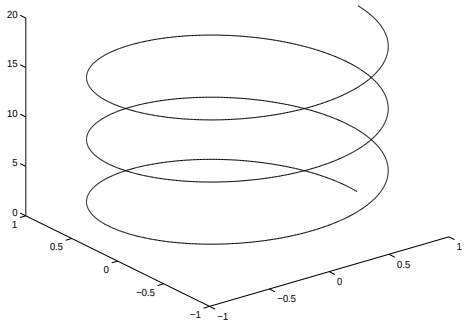
```
>>plot3([0 0 1 2 2],[0 1 1 1 2],[-1 -1 0 1 1])
>>plot3([0 0 1 2 2],[0 1 1 1 2],[-1 -1 0 1 1],'ko')
```

L'istruzione *plot3* può essere usata per tracciare curve tridimensionali. Si voglia, ad esempio, tracciare la *spirale tridimensionale* che è il moto di un punto materiale in moto circolare uniforme di raggio r e velocità angolare ω sul piano xy e rettilineo uniforme di velocità v lungo l'asse z. La curva ha le equazioni parametriche

$$\begin{cases} x = r\cos\omega t \\ y = r\sin\omega t \\ z = vt \end{cases}$$

Supponendo $v = 1$, $r = 1$ e $\omega = 1$, il disegno, sulla finestra grafica di MATLAB, di $n = 3$ giri di spirale si ottiene con le seguenti istruzioni

```
>>v=1; omega=1; r=1;
>>n=3; h=0.01;
>>t=0:h:2*pi*n;
>>x=r*cos(omega*t);
>>y=r*sin(omega*t);
>>z=v*t;
>>plot3(x,y,z)
```



L'istruzione *plot3* non permette di disegnare il grafico di funzioni reali di due variabili reali in quanto tale grafico è una superficie e *plot3* disegna solo linee. Per disegnare superfici si usano le istruzioni $mesh$ e $surf$. L'istruzione

```
mesh(x,y,Z)
```

dove $x = (x_1, \ldots, x_m)$, $y = (y_1, \ldots, y_n)$ sono vettori e Z è una matrice $m \times n$ disegna i punti (x_i, y_j, z_{ij}) e poi congiunge ogni punto (x_i, y_j, z_{ij}) ai punti vicini (x_{i-1}, y_j, z_{i-1j}), (x_{i+1}, y_j, z_{i+1j}), (x_i, y_{j-1}, z_{ij-1}) e (x_i, y_{j+1}, z_{ij+1}) con dei segmenti, ottenendo così un reticolo. L'istruzione

```
surf(x,y,Z)
```

fa la stessa cosa di *mesh* solo che colora i rettangoli del reticolo. La colorazione (dei segmenti in *mesh* e dei rettangoli in *surf*) dipende dal valore z e i colori usati sono gli stessi delle carte geografiche. Esempio

```
>>x=0:10; y=0:10 ;Z=zeros(11);
>>mesh(x,y,Z)
>>surf(x,y,Z)
>>Z(3,8)=1; Z(8,3)=-1;
>>mesh(x,y,Z)
>>surf(x,y,Z)
>>Z=rand(11);
```

```
>>mesh(x,y,Z)
>>surf(x,y,Z)
```

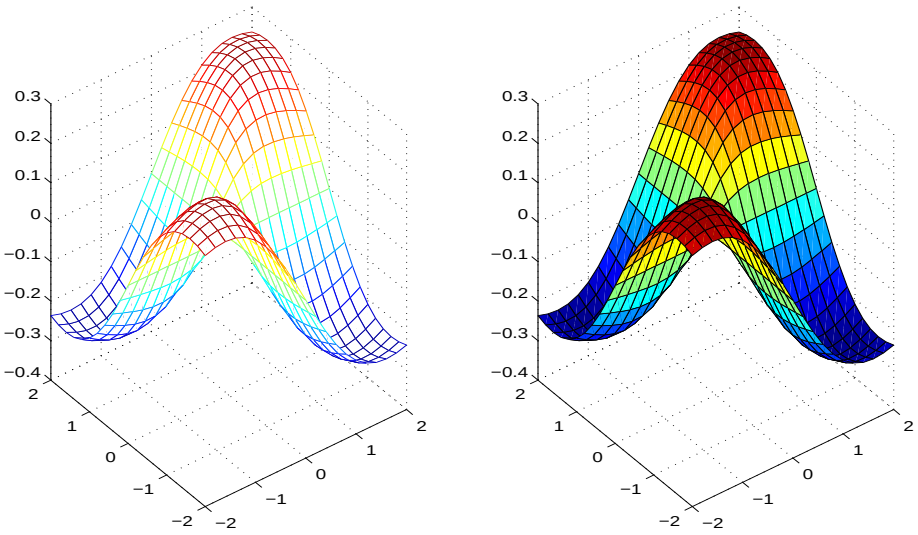
Se ora si vuole disegnare il grafico di una funzione $f : [a,b]\times[c,d] \to \mathbb{R}$ occorrerà disporre oltre a vettori di campionamento $x = (x_1, \ldots, x_m)$ e $y = (y_1, \ldots, y_m)$ di $[a,b]$ e $[c,d]$ rispettivamente anche della matrice $m \times n$ Z di elementi $z_{ij} = f(x_i, y_j)$, $i = 1, \ldots, m$, $j = 1, \ldots, n$. Il grafico della funzione è allora ottenuto con $mesh(x,y,Z)$ o $surf(x,y,Z)$. Per facilitare la costruzione della matrice Z è presente la funzione *meshgrid*. Con

```
[X,Y]=meshgrid(x,y)
```

si ottengono le matrici $m \times n$ X e Y di elementi $X(i,j) = x_i$, $Y(i,j) = y_j$, $i = 1, ..., m$, $j = 1, ..., n$. Si voglia, ad esempio, tracciare il grafico della funzione

$$f(x,y) = xye^{-\sqrt{x^2+y^2}},\ x,y \in [-2,2].$$

Questo è ottenuto tramite le seguenti istruzioni




```
>>x=-2:.2:2;
>>y=-2:.2:2;
>>[X,Y]=meshgrid(x,y);
>>Z=X.*Y.*exp(-sqrt(X.^2+Y.^2));
>>subplot(1,2,1)
>>mesh(x,y,Z)
>>subplot(1,2,2)
>>surf(x,y,Z)
```

Riferimenti bibliografici

[1] Maset, S. e Torelli, L. (2000), *Introduzione a MATLAB®*, Quaderno Matematico, Dipartimento di Scienze Matematiche, Università degli Studi di Trieste.