

III. Programmazione in MATLAB

1. Esecuzione di funzioni e di M-files

Abbiamo già introdotto le funzioni MATLAB assimilandole a comandi operativi in grado di eseguire elaborazioni predeterminate e di fornire un certo risultato (numerico, simbolico, grafico, ecc.). Alcune di queste funzioni sono interne a MATLAB (sono quelle che devono essere ottimizzate al fine di minimizzare il tempo di calcolo, per es. la funzione **sqrt**), altre sono esterne nel senso che sono inserite, come codice, in file esterni chiamati M-files. In particolare gli M-files sono utilizzati nei **Toolboxes** come contenitori delle funzioni specializzate nella soluzione dei problemi connessi alle applicazioni dei Toolboxes stessi.

Quando MATLAB riconosce il nome di una funzione interna o il nome di un M-file esegue la funzione o il file, nel senso che esegue tutti i comandi contenuti in esso.

Gli M-files costituiscono inoltre il mezzo fondamentale che l'utente dispone per programmare nel linguaggio MATLAB, creando 'programmi' che potranno essere salvati ed eseguiti come normali funzioni.

2. Gli M-files

MATLAB dispone di un completo linguaggio di programmazione che ci permette di scrivere una serie di 'istruzioni' in un file e di eseguirlo come se fosse un singolo comando. Il file può essere scritto come un ordinario file di testo e salvato con l'estensione **.m** (ad esempio **prova.m**).

Il nome del file diviene un nuovo comando associato al programma. Vi sono due tipi di M-file che MATLAB accetta:

- *file Script*;
- *file Funzione*.

Gli *script* eseguono semplicemente una serie di istruzioni MATLAB, mentre i *file funzione* possono accettare degli argomenti di input e fornire degli output. La tabella 1 mostra le caratteristiche di questi tipi di file.

Script M-File	Function M-file
Non accetta argomenti	Può accettare argomenti di input e produrre output
Opera esclusivamente nello spazio di lavoro	Le variabili locali della funzione sono allocate per default
Usato per automatizzare una successione di istruzioni che si devono ripetere	Usato per estendere il linguaggio

Tab. 1 Caratteristiche dei diversi tipi di M-file.

- Un esempio di M-file

Per prendere confidenza con gli M-files esaminiamo un esempio che ci consentirà di illustrare alcuni elementi importanti nella creazione delle funzioni MATLAB. Supponiamo di costruire una funzione di utente che sia in grado di calcolare il fattoriale di un numero (***n*!**). Ricordiamo la definizione di fattoriale:

$$n! = n \cdot (n-1) \cdot (n-2) \cdot \dots \cdot 3 \cdot 2 \cdot 1$$

Il file che racchiude la funzione è il seguente:

```
function f = fatt(n)           % linea di definizione della funzione
% fatt : fattoriale           % prima linea dell'help
% fatt(n) restituisce il fattoriale di n % testo dell'help
% usualmente denotato con n!
% Il calcolo è ottenuto mediante la funzione prod(n: -1: n).

f = prod(n: -1: 1);           % Corpo della funzione
```

Il file contenente la funzione comprende alcuni elementi comuni alle funzioni MATLAB:

- una linea di definizione della funzione che contiene il nome, il numero e l'ordine degli argomenti di input e output;
- la prima linea dell'help che viene visualizzata quando si ricerca una funzione mediante il comando **lookfor**;
- il testo dell'help che viene visualizzato con il comando **help fatt**;
- il corpo della funzione che contiene il codice del programma e che fornisce l'output.

Nota. La funzione **prod(x)** esegue il prodotto degli elementi del vettore x. Poiché la notazione **n:-1:1** significa tutti i numeri interi tra n ed 1, otteniamo il risultato del fattoriale di n.

- Costruiamo il file con l'Editor/Debugger

Un M-file è un file ordinario di testo che possiamo creare con un qualunque text editor. Da MATLAB possiamo operare scegliendo dal menu principale: **File – New – M-file**.

Comparirà la finestra di fig. 1 che mostra l'interfaccia grafica di Editor/Debugger nella quale potremo inserire il codice della funzione **fatt** precedentemente definita. Nella fig. 1 vediamo il risultato dell'inserimento.

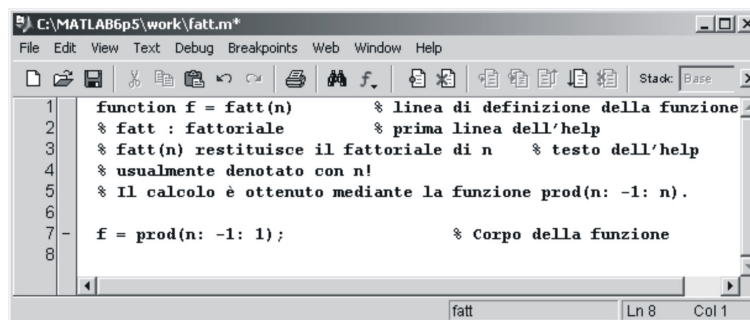


Fig. 1 Creazione del file **fatt.m**.

Nota. Nella finestra di Editor sono visibili i numeri delle linee di programma che servono soprattutto nella fase di Debugger per inserire dei '*breakpoints*' o punti di fermata dell'esecuzione del programma. In questo modo è possibile verificare i valori delle variabili utilizzate nel programma e scoprire eventuali errori di programmazione.

Possiamo ora salvare il file mediante il menu **File – Save As...** scrivendo il nome del file nella finestra di dialogo che appare. Da questo momento la funzione **fatt(n)** risulta utilizzabile in MATLAB.

Proviamo a scrivere nel **Command Window** la stringa **fatt(4)**; il risultato sarà

```
>> fatt(4)
```

```
ans =
```

Infatti $n! = 4 * 3 * 2 * 1 = 24$.

Scrivendo invece **help fatt** avremo:

```
>> help fatt

fatt: fattoriale                                % prima linea dell'help
fatt(n) restituisce il fattoriale di n    % testo dell'help
usualmente denotato con n!
Il calcolo è ottenuto mediante la funzione prod(n: -1: 1).
```

In tal modo l'utente può recuperare le informazioni relative alla funzione e quindi utilizzarla nel modo corretto.

- Gli Script e le Funzioni

I prossimi due esempi si occupano di mostrare i differenti tipi di M-files utilizzabili in MATLAB. Gli *script* offrono limitate possibilità funzionali rispetto alle *funzioni* in quanto non possono accettare parametri e si limitano quindi ad una esecuzione di un certo insieme di istruzioni. Nel primo esempio vengono calcolati gli elementi della serie di Fibonacci minori del numero 880. La serie citata ha la caratteristica che ogni elemento è pari alla somma dei due elementi precedenti:
1 1 2 3 5 8 13 ...

ESEMPIO 1

Calcolare gli elementi della serie di Fibonacci inferiori al numero 880.

Soluzione: la prima operazione è quella di creare un vettore iniziale (vettore f) che contiene i primi due elementi della serie. La seconda operazione consiste in un ciclo che calcola l'elemento successivo fino a che l'elemento si mantiene di valore inferiore a 880.

```
% Uno script per calcolare I numeri di Fibonacci
f = [1 1]; % Creazione del vettore iniziale
n = 1; % Inizializzazione dell'indice usato nel ciclo seguente
while (f(n) + f(n + 1)) < 880 % Con f(n) elemento del vettore f di indice n
    f(n + 2) = f(n) + f(n + 1); % Calcolo dell'elemento successivo
    n = n + 1;
end
```

L'esecuzione dello script calcola i primi 15 elementi della serie:

```
>> f
f =
Columns 1 through 10
1 1 2 3 5 8 13 21 34 55

Columns 11 through 15
89 144 233 377 610
```

Nota. All'interno dello script si osserva una delle strutture algoritmiche fondamentali: la **while do** che verrà esaminata in dettaglio nei paragrafi successivi.

Nel prossimo esempio sarà mostrata la creazione di una funzione che esegue lo stesso calcolo dello script precedente, ma che permette di calcolare l'elemento n -esimo della serie di Fibonacci.

ESEMPIO 2

La funzione **fibonacci(n)** che vogliamo costruire, e che sarà memorizzata nel file MATLAB **fibonacci.m**, deve rispettare la sintassi delle funzioni. Una volta che la funzione sia memorizzata essa può essere 'richiamata' semplicemente digitando il suo nome, assieme ai suoi parametri, all'interno della finestra di comando. Nel nostro caso l'unico parametro passato alla funzione è il numero dell'elemento della serie che si vuole calcolare. Ad esempio fibonacci(5) calcola il quinto elemento della serie.

```
function f = fibonacci(n)
% Calcola i numeri di Fibonacci.
% Incidentalmente, il nome Fibonacci viene da Filius Bonassi, o figlio di Bonassi
if n > 2
    f = fibonacci(n - 1) + fibonacci(n - 2);
else
    f = 1;
end;
```

Calcoliamo il 22-esimo elemento della serie:

```
>> fibonacci(22)
ans =
    17711
```

3. Il controllo del flusso di elaborazione

All'interno di un programma le istruzioni sono automaticamente eseguite in sequenza. Se però vi sono delle strutture che permettono di controllare il flusso, è possibile variare la sequenza di esecuzione delle istruzioni e modificare quindi il flusso di elaborazione.

MATLAB dispone di otto tipi di controllo del flusso di elaborazione simili a quelli presenti nel linguaggio C. Vediamo in breve quali sono:

Strutture per il controllo del flusso Descrizione

1	if espressione logica insieme di istruzioni End	La struttura if esegue le istruzioni se l'espressione è vera o diversa da zero.
2	if espressione logica insieme di istruzioni1 Else insieme di istruzioni2 End	La struttura if - else esegue due insiemi di Istruzioni: la prima se l'espressione è vera o diversa da zero, la seconda se l'espressione è falsa o zero
3	if espressione logica1 insieme di istruzioni1 elseif espressione logica2 insieme di istruzioni2 elseif espressione logica3	È la struttura if – else – end più generale, solo un gruppo di istruzioni della struttura viene eseguito.

	insieme di istruzioni3 else insieme di istruzioni End	Se nessuna espressione precedente è risultata vera.
4	switch espressione case valore1 insieme di istruzioni case { valore2, valore3,} insieme di istruzioni otherwise insieme di istruzioni end	Seleziona il caso in base al risultato dell'espressione valore1, valore 2, ..., possono essere anche delle espressioni. Anche gruppi di casi possono dare origine all'esecuzione dello stesso insieme di istruzioni
5	while espressione insieme di istruzioni End	Il ciclo while esegue le istruzioni sino a che l'espressione è vera o diversa da zero.
6	for indice = inizio: incremento: fine insieme di istruzioni End	Il ciclo for esegue l'insieme di istruzioni un predeterminato numero di volte specificato dal valore dell'indice.
7	for x = vettore v insieme di istruzioni End	Nel ciclo for ad ogni iterazione viene assegnato ad x il valore di un elemento del vettore v ed eseguite le istruzioni.
8	Continue	Passa il controllo alla nuova iterazione all'interno di un ciclo for o while e salta le rimanenti istruzioni contenute nel corpo del ciclo.
9	Break	Termina l'esecuzione dei cicli for e while
10	try...catch	Modifica il controllo del flusso di esecuzione se viene riscontrato un errore.
11	return	Causa il ritorno immediato alla funzione chiamante.

Tab. 2 Tipi di controllo del flusso

- Esame dettagliato ed esemplificazione delle strutture di controllo del flusso

If, then else

La prima struttura che consideriamo viene indicata sinteticamente come **if**, **else**, e **elseif** e, come la successiva **switch**, appartiene alla tipologia delle *strutture di selezione*. Queste strutture eseguono determinati gruppi di istruzioni sulla base di qualche condizione logica. Le strutture indicate con i numeri **1**, **2** e **3**, nella tabella 2 sono versioni diverse della stessa struttura. La prima, la più semplice, ha la sintassi seguente

```
if espressione logica      % L'espressione fornisce i valori logici Vero o Falso.
    insieme di istruzioni  % Corpo del programma.
end
```

Se l'espressione logica è vera (valore = 1), MATLAB esegue l'insieme di istruzioni contenute tra l'**if** e l'**end**.

Se la condizione è falsa (valore = 0), MATLAB salta tutte le istruzioni contenute tra l'**if** e l'**end**.

ESEMPIO 3

```
if rem(a, 2) == 0    % La funzione rem calcola il resto della divisione tra a e 2.
```

```

disp(' a è pari ') % La funzione disp visualizza la stringa tra gli apici.
b = a / 2;         % Assegna a b il valore a/2 se a è pari.
end

```

Se la variabile *a* è pari, la divisione per 2 darà un resto nullo, per cui la condizione sarà verificata e verrà mostrata la stringa: *a* è pari.

Nota. MATLAB permette di innestare un qualsiasi numero di **if**; per cui la sintassi seguente è corretta

```

if espressione logica
    if espressione logica
        if espressione logica
            ....
            ....
        end
    end
end
end
end

```

Nota. Se **X** è una matrice, il codice seguente

```

if X
    istruzioni
end

```

è equivalente a

```

if all(X(:)) % la funzione all determina se tutti gli elementi della matrice
              % sono diversi da zero.
    istruzioni
end

```

ovvero il test viene fatto su tutti gli elementi di **X**. Se anche uno solo di essi è nullo, la funzione **all** restituisce il valore logico falso.

La struttura **2** ha la seguente sintassi

```

if espressione logica
    insieme di istruzioni1
else
    insieme di istruzioni2
end

```

Se l'espressione logica è vera (1), MATLAB esegue l'insieme di istruzioni contenute tra l'**if** e l'**else**.

Se la condizione è falsa (0), MATLAB esegue quelle tra l'**else** e l'**end**.

ESEMPIO 4

Il programma visualizza una delle due stringhe: la prima se il contenuto di *A* è nullo, altrimenti la seconda (vedi nota precedente).

```

if A

```

```

    disp(' L'insieme A non è vuoto ')
else
    disp(' L'insieme A è vuoto ')
end

```

La struttura **3**, che realizza la struttura più completa dal punto di vista della selezione, ha la seguente sintassi

```

if espressione logica1
    insieme di istruzioni1
elseif espressione logica2
    insieme di istruzioni2
else
    insieme di istruzioni3
end

```

Se l'espressione logica 1 è vera (1), MATLAB esegue l'insieme di istruzioni contenute tra l'**if** e l'**elseif**. Altrimenti verifica l'espressione logica2: se è vera MATLAB esegue l'insieme di istruzioni2 (tra l'**elseif** e l' **else**); altrimenti esegue quelle tra l'**else** e l'**end** (istruzioni3).

ESEMPIO 5

Il programma visualizza un messaggio d'errore nel caso in cui la variabile **n** sia negativa. Nel caso opposto si comporta in modo analogo al programma dell'esempio 14.

```

if n < 0 % Se n è negativo, visualizza un messaggio di errore.
    disp(' l'input deve essere positivo ');
elseif rem(n,2) == 0 % Se n è positivo e pari, lo divide per 2.
    A = n / 2;
else
    A = (n+1) / 2; % Se n è positivo e dispari, lo incrementa e lo divide per 2.
end

```

Switch

La struttura **switch** esegue certe istruzioni sulla base del valore di una variabile o espressione. La sua forma base, dal punto di vista della struttura algoritmica, si presenta come una selezione ad *n* possibilità. Esaminiamo la sua sintassi

```

switch espressione (scalare o stringa)
    case valore1 % Esegue le istruzioni alfa se l'espressione è % uguale a valore1.
        insieme di istruzioni alfa
    case {valore2, valore3, ...} % Esegue le istruzioni beta se l'espressione è % uno dei valori tra le
        parentesi {...}.
        insieme di istruzioni beta
    ...
    otherwise % Esegue le istruzioni omega se l'espressione non è % uguale a
        nessuno dei valori dei casi precedenti.
        insieme di istruzioni omega
end

```

La struttura è costituita da

- una parola chiave, la **switch** seguita da una espressione da valutare;
- un qualsiasi gruppo di casi. Questo gruppo contiene un insieme di linee che iniziano con la parola chiave **case** seguita da uno o più possibili valori per l'espressione. Questi valori possono essere il risultato di una qualsiasi istruzione valida MATLAB, compreso un ulteriore blocco switch. MATLAB esegue solamente le istruzioni del caso il cui valore coincide con quello dell'espressione, oppure quelle dell'istruzione **otherwise** se nessuno dei casi precedenti è verificato.

ESEMPIO 6

L'esempio mostra una tipica struttura switch nella quale l'input, cioè la variabile **input_num**, determina, sulla base del proprio valore, la visualizzazione di differenti messaggi per mezzo della funzione **disp**.

```
switch input_num % Variabile o espressione da valutare.
    case -1 % Caso in cui input_num = -1.
        disp('uno negativo');
    case 0 % Caso in cui input_num = 0.
        disp('zero');
    case 1 % Caso in cui input_num = 1.
        disp('uno positivo');
    otherwise % Tutti gli altri casi.
        disp('altro valore');
end
```

ESEMPIO 7

In questo esempio è mostrata la possibilità della struttura **switch** di raggruppare casi che provocano le stesse azioni.

```
switch var % Il valore della variabile var determina la selezione dei casi.
    case 1
        disp('1')
    case {2,3,4} % Gruppo di casi.
        disp('2 or 3 or 4')
    case 5
        disp('5')
    otherwise
        disp('qualche altro valore')
end
```

While

Sia la struttura **while** come la successiva **for** sono entrambe *strutture algoritmiche iterative*. Nella struttura while, la cui sintassi è

```
while espressione % Il ciclo while esegue le istruzioni sino a che il
                  % valore della espressione è vero o diverso da zero.
    insieme di istruzioni % Corpo del programma.
end
```

l'espressione iniziale determina l'effettuazione ciclica dell'insieme di istruzioni contenute all'interno della struttura stessa.

ESEMPIO 8

Determiniamo con un ciclo il numero intero n tale che $n!$ (ovvero n fattoriale) sia minore di 1000000. La funzione **prod** esegue il prodotto specificato dal suo argomento. In questo caso moltiplica $1*2*3*\dots*n$ (vedi la definizione dell'operatore **:**).

```
n = 1;
while prod(1:n) < 1e6          % 1e6 = 1*10^6 = 1000000
    n = n + 1;
end
n - 1
```

Il risultato è il numero $n = 9$. Infatti utilizzando la funzione **fatt** del paragrafo 2 abbiamo

```
>> fatt(9)
ans =    362880      % < 1e6 = 1000000

>> fatt(10)
ans =    3628800
```

Il ciclo continua ad eseguire la funzione **prod(1:n)** con n crescente sino a che la condizione è verificata. Il valore cercato è $n-1$, poiché è l'ultimo valore che soddisfa la condizione.

For

Esistono due forme della struttura iterativa **for**. La prima, simile alla forma sintattica del C, è la seguente

```
% Il ciclo for esegue le istruzioni un predeterminato numero di volte specificato dal
% valore dell'indice. Se l'incremento non è indicato viene considerato unitario.
for indice = inizio: incremento: fine
    insieme di istruzioni          % Corpo del programma.
end
```

La seconda, che utilizza la natura vettoriale di MATLAB, assegna alla variabile x , quando esegue le istruzioni, tutti i valori del vettore v

```
% Nel ciclo for ad ogni iterazione viene assegnato ad  $x$  il valore di un elemento
% del vettore  $v$  (iniziando dal primo) ed eseguite le istruzioni.
for  $x$  = vettore  $v$ 
    insieme di istruzioni          % Corpo del programma.
end
```

ESEMPIO 9

Per generare le potenze 2^n con $0 \leq n \leq 8$ possiamo utilizzare il codice seguente

```
x = 1;
for i = 1:8
```

```
x = 2 * x;
end
>> x
x = 1 2 4 8 16 32 64 128 256
```

Il ciclo viene ripetuto otto volte.

ESEMPIO 10

Il codice dell'esempio calcola la radice quadrata degli elementi del vettore *x* dell'esempio 9. La funzione utilizzata è la **sqrt**.

```
j=1;
for i = x % L'indice i assume tutti i valori del vettore x dell'esempio 9.
    y(j) = sqrt(i); % Calcola la radice quadrata.
    j = j+1;
end
```

Il risultato è

```
>> y
y = 1.0000 1.4142 2.0000 2.8284 4.0000 5.6569 8.0000 11.3137 16.0000
```

ESEMPIO 11

L'esempio crea una matrice *A* di *m* righe ed *n* colonne. Per fare ciò è indispensabile utilizzare due cicli, uno innestato nell'altro. Nel ciclo più interno viene incrementato l'indice delle colonne, mentre in quello più esterno quello delle righe. In tal modo la matrice viene creata riga per riga.

```
% Si possono innestare cicli for uno dentro l'altro
m=4; n=3; % m = numero di righe; n = numero di colonne
for i = 1:m
    for j = 1:n
        A(i,j) = 1/(i + j - 1);
    end
end
```

Ecco il risultato del programma

```
>> A
A =
1.0000 0.5000 0.3333
0.5000 0.3333 0.2500
0.3333 0.2500 0.2000
0.2500 0.2000 0.1667
```

Continue

Passa il controllo alla nuova iterazione all'interno di un ciclo **for** o **while** e salta le rimanenti istruzioni contenute nel corpo del ciclo. In un ciclo innestato in un altro, passa il controllo alla nuova iterazione del ciclo immediatamente più esterno. Nell'esempio seguente è mostrato un programma che conta le linee di codice di un file (**magic.m** definito da MATLAB) mediante un

ciclo nel quale devono venire eliminate le linee bianche (linee vuote) e le linee di commento che iniziano con il carattere '%'. L'istruzione `continue` serve proprio per saltare il conteggio all'interno del ciclo.

ESEMPIO 12

Questo esempio è interessante anche perché ci dà la possibilità di esaminare alcune importanti funzioni di MATLAB:

- **fopen**: utilizzata per aprire un file. I parametri sono: il nome del file ed il tipo di accesso permesso al file. Nell'esempio 'r' significa di sola lettura;
- **feof**: fine del file trovato;
- **fgetl**: legge la linea corrente del file e si sposta sull'altra;
- **isempty**: verifica se la stringa è vuota;
- **strncmp**: confronta due stringhe per un certo numero di caratteri. Nel nostro caso le stringhe sono: la linea letta e il carattere '%', per cui il numero 1 indica il confronto tra il primo carattere della linea ed il carattere '%'.

```
fid = fopen('magic.m' , 'r'); % Funzione per aprire un file per la sola lettura: se
                             % l'operazione non riesce fid viene posto a -1,
                             % altrimenti assume un valore positivo associato al file aperto
cont = 0;
while ~feof(fid) % Se non si trova la fine del file (end of file) si cicla.
    linea = fgetl(fid); % Legge o recupera la linea corrente e la pone nella stringa line.
    if isempty(linea) | strncmp(linea,'% ',1)
        %Se la linea è vuota o inizia con il simbolo '% '
        continue % salta l'incremento di count e va all'end del while
    end
    cont = cont + 1;
end
sprintf('%d linee',cont)
```

Nota. La funzione `sprintf('%d linee',cont)` ha un comportamento simile alla funzione del linguaggio C `printf`. In questo caso il risultato del programma sarebbe
25 linee

poiché secondo la sintassi tipica della funzione `printf` verrebbe visualizzato il valore della variabile `cont`, con formato decimale, e la stringa `linee`. In effetti il file **magic.m** ha 25 linee di codice.

Break

Termina l'esecuzione dei cicli **for** e **while**. Quando l'istruzione **break** è incontrata l'esecuzione continua con la successiva istruzione subito dopo la fine del ciclo.

Try...Catch

Questa struttura è stata sviluppata appositamente per gestire gli errori. La sua sintassi è

```
try
    istruzioni
    ...
    istruzioni
catch
```

```
    istruzioni
    ...
    istruzioni
end
```

Le istruzioni comprese tra la parola **try** e la parola **catch** vengono eseguite fino a che non si verifichi un errore, nel qual caso vengono eseguiti i comandi compresi tra la parola **catch** e la parola **end**.

Return

Termina la sequenza di comandi correnti e restituisce il controllo alla funzione ‘invocante’ (gerarchicamente dominante sul codice chiamato) o alla tastiera dell’utente.

4. La valutazione delle stringhe

La valutazione delle stringhe aggiunge potenza e flessibilità al linguaggio MATLAB, poiché permette di eseguire operazioni costruendo una sorta di ‘*stringhe eseguibili*’, ottenute anche concatenando tra loro stringhe diverse.

Le funzioni MATLAB che consentono tale possibilità sono:

- **eval**;
- **feval**.

La funzione **eval** valuta una stringa contenente delle espressioni, istruzioni o funzioni MATLAB. Nella sua forma più semplice la sua sintassi è:

```
eval('string')
```

Consideriamo nuovamente il codice dell’esempio 11, nel quale ora il calcolo dell’elemento della matrice viene eseguito mediante la valutazione di una stringa, che contiene esattamente l’espressione che calcolava l’elemento della matrice.

```
m=4; n=3;
for i = 1:m
    for j = 1:n
        a(i,j) = eval('1/(i+j-1)');
    end
end
```

Come ulteriore esempio di utilizzo della funzione **eval**, la semplice istruzione

```
>> eval('t = clock'); % La funzione clock restituisce il tempo dell’orologio del
                        % calcolatore come t=[anno mese giorno ora minuto secondi]
t = 1.0e+003 *
    2.0030    0.0110    0.0130    0.0120    0.0480    0.0388
                        %Gli elementi di t sono tutti moltiplicati per 1.0e+003=1000
```

restituisce il tempo corrente nell’attimo di attivazione del comando (gli elementi del vettore, cioè 2.0030 0.0110 0.0130 0.0120 0.0480 0.0388, risultano tutti moltiplicati per il coefficiente 1.0e+003).

La funzione **feval** differisce dalla **eval** in quanto esegue una funzione piuttosto che una espressione MATLAB. La funzione eseguita è posta come primo argomento della **feval**. L'esempio seguente mostra l'utilizzo della funzione **feval** per il calcolo del seno a 90 gradi.

```
fun = 'sin';
x = input('Inserisci il valore: '); % Recupera l'input x
feval(fun,x) % valuta la funzione indicata dal parametro k per il valore x
```

Il risultato dell'esecuzione del codice è il seguente

Inserisci il valore: 90

ans = 1

Nota. È sempre meglio utilizzare la funzione **feval** che è eseguita molto più rapidamente della **eval**.

5. Esempi di utilizzo di MATLAB

Per comprendere la potenzialità di MATLAB basta esaminare l'help in linea e provare qualche esempio proposto. Praticamente tutti i campi della matematica e della scienza trovano in MATLAB un utile ausilio anche mediante i **Toolboxes**, che prendono in considerazione aspetti specialistici del calcolo e della rappresentazione dei risultati.

Gli esempi proposti riguardano la costruzione di programmi mediante il linguaggio di MATLAB e l'utilizzo di alcune funzioni relative ad argomenti come i polinomi e l'algebra in genere.

- Polinomi

Il calcolo polinomiale è un argomento che ci consente di mostrare l'utilità di un programma come MATLAB per ridurre fortemente l'impegno umano nell'ottenimento di risultati. Ad esempio chi avesse provato a moltiplicare o dividere tra loro polinomi di grado elevato si renderà subito conto del vantaggio.

Le principali funzioni polinomiali di MATLAB sono riportate in tabella 3

Funzione	Descrizione
conv	Moltiplica polinomi
deconv	Divide polinomi
poly	Determina i polinomi una volta che siano assegnate determinate radici
polyfit	Calcola i coefficienti del polinomio che interpola valori assegnati
polyval	Valuta il valore di un polinomio
roots	Calcola le radici di un polinomio

Tab. 3 Tabella delle funzioni polinomiali.

Rappresentazione dei polinomi in MATLAB

MATLAB rappresenta i polinomi mediante un vettore riga contenente i valori dei coefficienti ordinati in modo decrescente. Per esempio il polinomio

$$p(x) = x^3 - 12x - 5$$

si inserisce in MATLAB nel modo seguente:

```
p = [1 0 -12 -5];
```

Radici dei polinomi

Per calcolare le radici del polinomio, ovvero le soluzioni dell'equazione

$$x^3 - 12x - 5 = 0$$

basta utilizzare la funzione **roots**

```
>> roots(p)
r =
    3.6562
   -3.2332
   -0.4230
```

I valori contenuti in **r** sono le soluzioni dell'equazione algebrica che, sostituiti a x , annullano il polinomio.

Valutazione dei polinomi

Per controllare ciò possiamo applicare la funzione **polyval** che calcola il valore di un polinomio per uno o più valori della x .

```
>> polyval(p, r)    % p è il polinomio, r le radici appena determinate.
ans =
    1.0e-013 *      % Coefficiente moltiplicatore dei tre risultati successivi.

    0.2487
   -0.4263
   -0.0089
```

Come si vede dal calcolo il polinomio si annulla per tutte le radici r (i valori trovati sono inferiori a 10^{-13} : non sono esattamente uguali a 0 perché il calcolo numerico non può avere una precisione infinita).

È possibile, assegnate le radici di un polinomio, trovare i suoi coefficienti. La funzione **poly** è in grado di ottenere tale risultato:

```
>> p2 = poly(r)      % Le radici sono contenute nel vettore r
p2 =
    1.0000    0.0000   -12.0000   -5.0000
```

Come si vede, il polinomio **p2** è identico al polinomio **p** precedentemente definito.

Moltiplicazione e divisione di polinomi

Le funzioni che realizzano la moltiplicazione e la divisione tra polinomi sono: **conv** e **deconv**. Consideriamo per esempio i polinomi seguenti:

$$p_1(x) = x^3 - 12x - 5 \quad \text{e} \quad p_2(x) = 4x^2 + 7x - 9$$

e calcoliamo il loro prodotto

```
>> p1 = [1  0 -12 -5]; p2 = [4  7 -9]; % Definizione dei due vettori
>> p3 = conv(p1,p2) % Moltiplicazione dei vettori

p3 =
    4     7   -57  -104    73    45
```

Ovvero $p_3(x) = 4x^5 + 7x^4 - 57x^3 - 104x^2 + 73x + 45$.

Per provare la funzione **deconv** che esegue la divisione tra polinomi possiamo scrivere:

```
>> [quoz, resto] = deconv(p3, p1) % Divide il polinomio p3 per p1: il risultato è
                                   % il polinomio p2
quoz =                             % quoz è il polinomio quoziente
    4     7    -9
resto =                             % resto è il polinomio resto della divisione
    0     0     0     0     0     0
```

Come si nota, la divisione tra i polinomi ha fornito il risultato desiderato.