Introduzione alla

Programmazione in MATLAB 7

E. Cristiani, M. Sagona

Versione 1.9 – 18/11/2009

Queste dispense nascono dall'esperienza maturata dai due autori nell'uso di Matlab nell'ambito della ricerca e del calcolo scientifico. Sono rivolte agli studenti del Dipartimento di Matematica dell'Università di Roma "Sapienza" che abbiano già delle nozioni di base di programmazione (preferibilmente in C) e che cerchino una guida breve per iniziare a programmare in Matlab. In queste dispense non è compresa la parte grafica.

MATrix LABoratory è un *ambiente di lavoro* che include la possibilità di scrivere un proprio codice, creare delle interfacce user-friendly, usare una potente libreria grafica e una ricchissima libreria di funzioni matematiche, usare dei programmi built-in, visualizzare esperimenti fisici (Simulink), risolvere particolari problemi differenziali utilizzando una semplice interfaccia (Femlab).
E' usato in particolare da matematici, fisici ed ingegneri.

Pregi:
- rapidità di programmazione (linguaggio di altissimo livello)
- enormi potenzialità grafiche built-in
- facile debug
- help in linea chiaro e completo (al quale rimandiamo per ogni informazione non compresa in questo manualetto)
- è il linguaggio standard per l'analisi numerica insieme a C e Fortran
- è presente in tutte le università (versioni gratuite: Octave e Scilab)

Difetti:
- lentezza in fase di esecuzione (è un interprete, non un compilatore)
- prezzo
- aggiornamento relativamente rapido delle versioni
- non crea un file che sia eseguibile senza avere Matlab
- non è adatto per imparare a programmare.

Gestione delle finestre:
Current Directory: prima directory in cui Matlab cerca il file da eseguire.

Command Window: finestra in cui possono essere inseriti i comandi.
Ad esempio:

`pwd` oppure `cd`	per conoscere la Current Directory.
`ls` oppure `dir`	per conoscere il contenuto della Current Directory.
`help nome_comando`	per conoscere la sintassi di un comando.
`nome_file`	per lanciare in esecuzione un m-file (file di Matlab, estensione .m), ad esempio una demo.

La Command Window è la finestra dove viene visualizzato l'output di un programma.

Attenzione: la Command Window è l'unica finestra visibile in un ambiente non grafico.

Workspace: finestra nella quale si possono vedere tutte le variabili memorizzate. Con un doppio click sopra una di esse si apre l'Array editor (tipo foglio di Excel) dove si può leggere, modificare e stampare il contenuto delle variabili.

Editor: finestra dove si può scrivere un proprio codice.
Per eseguire un programma è necessario:

1) Salvare il file (estensione .m) nella directory X (solo la prima volta).
2) Cambiare la Current Directory in X.
3) Premere sulla tastiera il tasto F5 (salva il file sovrascrivendolo al precedente, segnala eventuali errori e, nel caso non ce ne siano, esegue) oppure digitare il nome del file nella Command Window.

Alcuni comandi fondamentali:

`clc`	pulisce la Command Window.
`;`	chiude l'istruzione senza mostrarne il risultato nella Command Window.
`%`	commento.
`who`	mostra il contenuto del workspace
`whos`	mostra il contenuto del workspace con maggior dettaglio
`clear`	svuota il workspace (cancella tutte le variabili in memoria)
`clear nome_variabile`	cancella solamente la variabile *nome_variabile*

Alcune osservazioni sul comando *clear*:
E' buona regola iniziare ogni programma Matlab con questo comando, in modo tale che in fase di esecuzione vengano utilizzate solamente le variabile introdotte nel codice. In caso contrario, Matlab terrà in memoria anche le variabili memorizzate dal programma precedentemente eseguito.
Inoltre è da notare che, **una volta eseguito un programma, le variabili memorizzate non vengono cancellate**. E' possibile quindi riutilizzarle direttamente dalla Command Window o in un altro m-file (senza usare il comando *clear*).

Dalla finestra principale di Matlab si accede al menù *Preferences* in cui è possibile scegliere - tra le tante cose - il formato numerico di output preferito (4 cifre decimali, 14 cifre decimali, notazione scientifica, ecc.).

Costanti:

realmax	1.7977e+308
realmin	2.2251e-308
eps	2.2204e-016
pi	3.14159265358979

Tutte le costanti di Matlab possono essere ridefinite a piacere dall'utente. Cancellandole dal Workspace ritornano ad assumere il valore predefinito.

In Matlab non è necessario dichiarare le variabili ed il loro tipo.

A ciascuna variabile viene riservato uno spazio in memoria nel momento in cui viene incontrata per la prima volta.
Esempio:

```
clear          (Workspace: vuoto)
n=1;           (Workspace: n)
p=23.67;       (Workspace: n,p)
```

Inoltre, sebbene in Matlab non esistano i puntatori, è permessa **un'allocazione dinamica della memoria** che non ne fa sentire la mancanza. Ad esempio, una volta memorizzato un vettore di *n* componenti, è possibile aggiungere altre componenti (*n*+1, *n*+2, ecc.) o cancellarne alcune run-time.

Attenzione: Matlab è case sensitive (*A* e *a* sono due variabili differenti).

Numeri complessi:
In Matlab si può lavorare con numeri reali o complessi senza alcuna differenza. L'unità immaginaria è indicata con *i* o con *j*
Esempio:

```
a=3+5i;
b=2+2i;
c=a+b;
```

Matlab riesce a distinguere l'unità immaginaria da un'eventuale altra variabile di nome *i* o *j,* ma è meglio non abusare di queste "facilitazioni".
Esempio:

```
i=7;
a=3+5i;
b=2+2i;
c=a+b;
```

Il risultato è lo stesso del programma precedente.

Vettorizzazione:
Matlab nasce per lavorare con semplicità su matrici e vettori.
Presentiamo qui di seguito alcuni comandi per memorizzare una matrice A di dimensioni *n* x *m*:

Con un doppio ciclo *for* (scelta sconsigliata):

```
for i=1:n
  for j=1:m
    A(i,j)=i+j;
  end
end
```

Direttamente da tastiera:

```
A=[1 2 3; 4 5 6];
```

(*n*=2, *m*=3)
Oppure analogamente

```
A=[1 2 3;
   4 5 6];
```

Con uno dei seguenti comandi (o una combinazione di essi):

`A=ones(n,m);`	crea una matrice con tutti valori uguali a 1
`A=zeros(n,m);`	crea una matrice con tutti valori uguali a 0
`A=rand(n,m);`	crea una matrice random (A(i,j) in [0,1])
`A=eye(n,m);`	crea una matrice con tutti 1 sulla diagonale principale e 0 altrove
`A=diag(v);`	crea una matrice con le componenti del vettore v sulla diagonale principale e 0 altrove

Alcuni comandi per operare sulle matrici (o vettori):

`norm(v,1/2/p/inf);`	calcola la norma 1/2/p/infinito del vettore v
`norm(A,1/inf/'fro');`	calcola la norma 1/infinito/Frobenius della matrice A
`A';`	calcola la matrice trasposta (e coniugata) di A
`inv(A);`	calcola l'inversa della matrice A
`det(A);`	calcola il determinante della matrice A
`eig(A);`	calcola gli autovalori della matrice A
`[L,U]=lu(A);`	fattorizzazione LU della matrice A
`find;`	trova gli elementi diversi da 0 (v. sintassi nell'help)
`reshape(A,m,n);`	trasforma la matrice A (*mn* elementi) in una matrice *m* x *n*
`size;`	legge le dimensioni di una matrice (v. sintassi nell'help)
`gradient;`	calcola il gradiente numerico (v. sintassi nell'help)
`spy(A);`	disegna e conta gli elementi di A diversi da 0
`mesh(A);`	disegna la matrice A come funzione a 2 variabili
`imagesc(A);`	disegna la matrice A interpretandola come immagine (i colori usati dipendono dai valori di A)

Ovviamente in Matlab non c'è bisogno di utilizzare dei cicli per operare su matrici o vettori.
Esempio:
```
B=[1 2 3; 4 5 6];
A=B;
```
A è una matrice identica a B.

L'operatore due punti (:) è fondamentale e permette di creare rapidamente vettori.
Esempio:

`x=a:b;`	crea il vettore x=(a,a+1,a+2,…,b)
`x=a:h:b;`	crea il vettore x=(a,a+h,a+2h,…,b_0) dove b_0=a+*n*h per qualche *n* e b-h<b_0<=b
`v=A(i,:);`	crea un vettore v che ha per elementi quelli della *i*–ma riga di A
`v=A(1:4,j);`	crea un vettore v che ha per elementi quelli dalla prima alla quarta riga della *j*–ma colonna di A
`B=A(1:4,:);`	crea una matrice B che ha per elementi quelli dalla prima alla quarta riga di A
`A(1:n,1:m)=0;`	analogo a `A=zeros(n,m)` ma differentemente da quest'ultimo non cancella dal workspace un'eventuale variabile A già memorizzata

Concatenazione:

`C=[A B];`	C è una matrice composta dalle matrici A e B affiancate
`v=[a b];`	v è un vettore composto dai vettori *a* e *b* affiancati

ATTENZIONE: C non è un vettore di matrici ma è a tutti gli effetti una matrice. Ugualmente per v.

Cancellazione di un elemento:

`a(n)=[];`	l'elemento *n*-mo del vettore *a* viene eliminato ([] indica l' "insieme vuoto")

Operatori aritmetici:

+	somma
-	sottrazione
*	prodotto
/	divisione destra (6/2=3)
\	divisione sinistra (2\6=3)
^	elevamento a potenza

Usati in questa forma gli operatori agiscono matricialmente (o vettorialmente).

Preceduti da un punto (.) agiscono elemento per elemento.
Esempi:

`A*B;`	prodotto righe per colonne di A per B
`A.*B;`	prodotto elemento per elemento (C(i,j)=A(i,j)*B(i,j))
`a*b';`	prodotto scalare del vettore riga *a* ed il vettore riga *b* (se reali)
`A^2;`	analogo a `A*A;`
`A.^2;`	analogo a `A.*A;`
`A+B;`	somma delle matrici A e B
`A.+B;`	analogo a A+B
`A+2;`	somma 2 a tutti gli elementi di A

Un discorso a parte meritano i **sistemi lineari** (nella forma $Ax=b$, con x e b vettori colonna), che possono essere risolti utilizzando l'operatore di divisione. Si possono usare due comandi:

1) `x=A\b;`	Utilizza il metodo di Gauss (scelta consigliata)
2) `x=inv(A)*b;`	Formalmente corretto ma numericamente meno preciso e più lento

Altre operazioni possibili:

`X=A\B;`	Risolve l'equazione AX=B ($X=A^{-1}B$)
`X=A/B;`	Risolve l'equazione XB=A ($X=AB^{-1}$)
`A./B;`	A(i,j)/B(i,j)
`A.\B;`	B(i,j)/A(i,j)

Altri importanti comandi per operare sui vettori:

`max(v);`	calcola l'elemento massimo del vettore v
`min(v);`	calcola l'elemento minimo del vettore v
`sum(v);`	calcola la somma degli elementi del vettore v
`prod(v);`	calcola il prodotto degli elementi del vettore v
`mean(v);`	calcola la media aritmetica degli elementi del vettore v
`sort(v);`	ordina in modo crescente gli elementi del vettore v

In Matlab è possibile definire anche matrici a più dimensioni.

Esempio (3 dimensioni):

```
for i=1:m
   for j=1:n
      for k=1:p
         A(i,j,k)=i+j+k;
      end
   end
end
```

In questo caso A(:,:,k) è una normale matrice bidimensionale per ogni k=1,...,p.

E' importante notare che molti comandi agiscono in maniera differente a seconda della dimensione del proprio argomento.
In particolare, le funzioni vettoriali (come ad es. *max, min, sum, prod, mean ,sort*) applicate alle matrici agiscono su ogni colonna e restituiscono un vettore.
Esempio:

`min(vettore_v);`	restituisce uno scalare (il valore minimo di *vettore_v*)
`min(matrice_A);`	restituisce un vettore costituito dal valore minimo di ogni colonna di *matrice_A*
`min(matrice3D_A3);`	restituisce una matrice costituita dal valore minimo di ogni colonna della matrice bidimensionale *matrice3D_A3(:,:,k)*

Quindi per ottenere il valore minimo degli elementi di una matrice si deve usare comando
`min(min(A));`

Funzioni matematiche (agiscono sempre elemento per elemento):

`sqrt`	radice quadrata
`exp`	esponenziale
`log, log10, log2`	log in base e, 10, 2
`sin, cos, tan`	seno, coseno, tangente (argomento in radianti)
`sind, cosd, tand`	seno, coseno, tangente (argomento in gradi)
`asin, acos, atan`	arcoseno, arcocoseno, arcotangente
`abs`	valore assoluto (o modulo ρ del numero complesso a+ib=(ρ,θ))
`angle`	restituisce l'angolo θ del numero complesso a+ib=(ρ,θ)
`factorial`	fattoriale
`rem(x,y), mod(x,y)`	resto della divisione di *x* per *y* (v. help per differenze)
`real, imag`	restituisce la parte reale e la parte immaginaria di un numero complesso
`conj`	complesso coniugato
`floor`	arrotonda all'intero più basso ([x])
`ceil`	arrotonda all'intero più alto ([x] se x è in **Z**, [x]+1 altrim.)
`round`	arrotonda all'intero più vicino

Stringhe di caratteri:
In Matlab una stringa di caratteri viene assegnata attraverso gli apici ` `.
Esempio:
`a='ciao';` *a* è una stringa di caratteri

La riga di comando:
`'ciao'`
fa scrivere nella Command Window "ans=*ciao"* come se fosse il contenuto di una qualsiasi variabile.

Per avere un output formattato si usa il comando *disp*:
`disp('ciao');` scrive nella Command Window "ciao".

Per scrivere a video il contenuto di una variabile numerica, si deve usare il comando *num2str* (da numero a stringa):

```
n=3;
disp(num2str(n));          scrive a video "3"
```

Se il comando *disp* ha più di un argomento, si devono usare le parentesi quadre []:
`disp(['n= ',num2str(n)]);` scrive a video "n= 3"

Per stabilire in modo più preciso il formato di scrittura delle variabili si può usare quest'altro comando:
```
formato='%.numerolettera';
num2str(n,formato);
```
dove `numero` indica il numero di cifre significative che si vuole visualizzare e `lettera` indica il tipo di scrittura desiderata (scelte possibili: d,e,f,g, ecc.)

Per assegnare il valore di una variabile da tastiera si usa invece il comando *input*:

`a=input('inserire n. n= ');` scrive nella Command Window "inserire n. n= " e attende l'introduzione di un numero (es. 3) o di un vettore (es. [1 2 3]) o di una matrice (es. [1 2 3; 4 5 6] o di una stringa (es. 'ciao') dall'utente. Infine assegna ciò che è stato inserito alla variabile *a.*

for

La sintassi del ciclo *for* è semplicissima:
```
for i=1:n
   ...istruzioni...
end
```
L'istruzione è eseguita *n* volte.

```
for i=1:p:n
   ...istruzioni...
end
```
In questo caso il contatore viene incrementato di *p* ad ogni ciclo ed il ciclo termina quando *i*>*n*.

while

La sintassi del ciclo *while* è analoga:
```
while condizione
   ...istruzioni...
end
```

Se la condizione è multipla si ha
```
while (condizione1) £ (condizione2) £ (condizione3) ...
   ...istruzioni...
end
```
dove £ è un qualsiasi operatore logico.

Si può assegnare la precedenza alla valutazione di certe espressioni con le parentesi tonde ()

If
```
if condizione
 ... istruzioni ...
end
```

If else
```
if condizione
   ... istruzioni ...
else
   ... istruzioni ...
end
```

If elseif
```
if condizione1
```

```
   ... istruzioni ...
elseif condizione2
   ... istruzioni ...
elseif condizione3
   ... istruzioni ...
else
   ... istruzioni ...
end
```

If else if

```
if condizione1
   ... istruzioni ...
else
   if condizione2
      ... istruzioni ...
   end
end
```

switch, case

```
metodo='lineare';

switch (metodo)
case 'quadratico'
disp('il metodo è quadratico');
case 'lineare'
disp('il metodo è lineare');
otherwise
disp('il metodo è sconosciuto');
end
```

In questo caso l'output sarà "Il metodo è lineare".

OSSERVAZIONE: nel caso in cui la scelta deve essere effettuata valutando il contenuto di stringhe di lunghezza diversa (come nel precedente esempio) la scelta di "switch, case" è obbligatoria (non si può usare "if").

Operatori relazionali:

`<`	minore
`<=`	minore o uguale
`>`	maggiore
`>=`	maggiore o uguale
`==`	uguale
`~=`	diverso

Operatori logici:

`A & B`	A and B
`~A`	not A
`A | B`	A or B
`xor(A,B)`	A xor B (or esclusivo)

Altri comandi:

`all`	agisce su vettori. vale 1 se tutti gli elementi sono non nulli, 0 altrimenti
`any`	agisce su vettori. vale 1 se esiste almeno un elemento non nullo, 0 altrimenti

Variabili booleane:
In Matlab le variabili booleane, cioè che assumono solamente i valori 0 (false) o 1 (true) sono trattate alla stregua di una qualsiasi altra variabile.
Esempio:

```
a=(3==3); oppure a=1; oppure a='qualsiasi_stringa_non_vuota'; oppure a=5>1;
if a
   disp('ciao');
end
```

In ognuno dei casi descritti l'output sarà "ciao".

Gli operatori relazionali operano anche su vettori e matrici:
Esempio:

```
A=[1 2;
   3 4];
B=[1 2;
   3 5];
M=A==B;
```

In M sarà memorizzata la matrice
1 1
1 0

Altri comandi utili:

`pause`	l'esecuzione si arresta fino a che non si preme un tasto
`pause(n)`	l'esecuzione si arresta per *n* secondi
`break`	arresta l'esecuzione del programma o esce dal ciclo più interno in cui ci si trova
`return`	arresta l'esecuzione della function corrente
`CTRL-C`	digitato nella Command Window, arresta l'esecuzione del programma run-time
`...`	permette di spezzare una riga di comando per andare a capo (sia nell'editor che nella Command Window)

Tempo di CPU:

`tic`	inizia a contare il tempo
`toc`	finisce di contare il tempo e lo scrive a video
`t=toc;`	finisce di contare il tempo e lo memorizza nella variabile t senza scriverlo a video

Attenzione! In Matlab i cicli vengono eseguiti molto più lentamente rispetto al C o al Fortran. E' quindi buona regola – quando possibile – usare le function predefinite e vettorizzare.

Campi di matrici:
In Matlab è possibile definire matrici o vettori che hanno per elementi matrici, vettori o stringhe. Supponiamo ad esempio di voler memorizzare una matrice 2 x 2 i cui elementi sono vettori, matrici e stringhe:
M=(9 'ciao'

(5,6) (1 2;3 4))
Si procede nel seguente modo:
```
M(1,1).field=9;
M(1,2).field ='ciao';
M(2,1).field =[5 6];
M(2,2).field =[1 2;3 4];
```
Il nome del campo (in questo caso *field*) è arbitrario.
E' anche possibile accedere direttamente a qualsiasi elemento di *M.*
Ad esempio, con il comando
```
h=M(2,1).field(1);
```
si memorizza in *h* il valore 5.

Supponiamo ora di voler memorizzare un vettore tridimensionale in cui ogni elemento è composto da una stringa e da un numero:
v=('pippo' 234, 'pluto' 54, 'topolino' 567)
Si procede nel seguente modo:
```
v(1).nome='pippo';
v(1).numero=234;
v(2).nome='pluto';
v(2).numero=54;
v(3).nome='topolino';
v(3).numero=567;
```

E' anche possibile (e a volte più comodo) usare il comando `struct` (digitare "help struct" nella Command Window per la sintassi esatta).

Script:
Uno script di Matlab è un *m-file* contenente una sequenza di istruzioni. Esso può essere richiamato da un qualsiasi altro *m-file* o direttamente dalla Command Window. Non è ammesso alcun passaggio di variabili. Gli script sono usati principalmente per aumentare la leggibilità del codice.
Esempio:
Creare il file "incrementa_a.m" contenente l'istruzione `a=a+1;`
La sequenza di istruzioni
```
a=1; incrementa_a;
```
è a tutti gli effetti equivalente a
```
a=1; a=a+1;
```

Osservazione: gli script devono essere salvati in una cartella "conosciuta" da Matlab, in modo che esso possa localizzarli nel momento in cui vengono richiamati. E' possibile quindi salvare i file:
1) nella Current Directory (scelta più semplice)
2) in una directory a piacere che deve poi essere aggiunta alla lista delle directory predefinite nella lista dei path

Function:
In Matlab è possibile definire delle proprie funzioni, come negli altri principali linguaggi di programmazione; ma al contrario di linguaggi come il Pascal, C, Fortran, esse sono scritte in un file a parte rispetto al *main* (programma principale). Il file contenente la function è a sua volta un *m-file* e deve essere salvato col nome della function stessa, per permettere a Matlab di trovarlo nel momento in cui è chiamato dal *main.* E' comunque da notare che in un file contenente una function - ad esempio il file function1.m - possono essere presenti anche tutte le altre function chiamate all'interno della function1. In altre parole, non è necessario salvare un file per function.

Inoltre ogni m-file contenente una o più function può essere eseguito come un qualsiasi m-file, purché contenga la definizione di tutte le variabili usate al proprio interno.

In Matlab – salvo diversa indicazione - **le variabili sono sempre locali**. E' possibile definire variabili globali (con il comando *global*, v. sintassi nell'help) ma è bene limitare questa possibilità alle sole variabili che non devono essere modificate (vale a dire quelle variabili che negli altri linguaggi di programmazione si definiscono come *costanti*). Infatti, in Matlab, una function può ricevere e/o restituire un numero qualsiasi di variabili (siano esse matrici, vettori, stringhe, ecc.). E' quindi previsto che tutte le variabili che devono essere modificate siano passate alla function e riprese da essa *esplicitamente*.

Esempi di chiamata della function "funz" dal *main*:

`funz;`	la function non riceve né restituisce niente
`a=funz;`	la function non riceve niente e restituisce *a*
`funz(b);`	la function riceve *b* e non restituisce niente
`a=funz(b);`	la function riceve *b* e restituisce *a*
`[a,b]=funz(c,d,e);`	la function riceve le tre variabili *c,d,e* e restituisce due valori che sono assegnati alle variabili *a* e *b*
`a=funz(a);`	questo è il modo migliore per passare una variabile per "indirizzo" (cioè modificarla all'interno della function)

Osservazione: nell'ultimo esempio (`a=funz(a);`) è possibile che le due variabili *a* in entrata e in uscita siano di tipo diverso. Ad esempio, la function *funz* può ricevere in input uno scalare *a* e restituire una matrice *m x n* che viene memorizzata di nuovo nella variabile *a.*

Sintassi di una function:

Chiamata nel main:
```
funz;
```
Function:
```
function funz
... istruzioni ...
```

Chiamata nel main:
```
a=funz(b);
```
Function:
```
function z=funz(d)
... istruzioni ...
```
tra le quali ci deve essere l'assegnazione della variabile z (che diventerà a)

Chiamata nel main:
```
[a,b]=funz(c,d,e)
```
Function:
```
function [h,j]=funz(cc,dd,ee)
... istruzioni ...
```
tra le quali ci deveno essere le assegnazioni delle variabili h,j (che diventeranno a,b)

Osservazione: i file nei quali vengono scritte le varie function devono essere salvati in una cartella "conosciuta" da Matlab, in modo che esso possa localizzarli al momento della chiamata dal *main*. E' possibile quindi salvare i file:
1) nella Current Directory (scelta più semplice)
2) in una directory a piacere che deve poi essere aggiunta alla lista delle directory predefinite nella lista dei path

Puntatori:

In Matlab non esistono i puntatori, ad eccezione dei puntatori a funzioni. E' possibile passare ad una function il nome di un'altra function che verrà richiamata dalla prima function.
Esempio: il main chiama `funz1(@funz2)`, funz1 riceve la variabile "funz2" che può usare per chiamare la function funz2. Vedi anche `feval`.

Funzioni ricorsive:
In Matlab sono ammesse le function ricorsive, cioè le function che chiamano sé stesse.

Numero di variabili in uscita da una function (nargin, nargout):
Con le istruzioni *nargin* e *nargout* Matlab è in grado di riconoscere il numero di variabili rispettivamente in entrata e in uscita con cui è stata chiamata una function. Ad esempio, il comando *toc* (function predefinita il cui listato è visibile e modificabile) può essere chiamato in due modi diversi:

```
toc;
t=toc;
```

Nel primo caso, la function riconosce che non è stata richiesta nessuna variabile in uscita, ed il comando ha l'effetto di scrivere a video il tempo di esecuzione. Nel secondo caso, la function riconosce che è stata richiesta una variabile in uscita ed il comando ha l'effetto di memorizzare nella variabile *t* il tempo trascorso senza scrivere niente a video. Questa doppia possibilità è resa possibile dall'istruzione *nargout:* Infatti la function *toc* è strutturata sostanzialmente nel seguente modo:

```
function t=toc
tempo= ora attuale meno ora della chiamata a tic
if nargout==0         % nessuna variabile in uscita
   disp(['tempo trascorso = ',num2str(tempo)]);
elseif nargout==1     % 1 variabile in uscita
   t=tempo;
end
```

nargin è un comando analogo per le variabili in entrata.

Cenni di grafica (plot):
Per disegnare il grafico di una funzione $f:[a,b] \rightarrow R$ con passo di discretizzazione h, si può procedere nel seguente modo:

```
x=a:h:b;
y=f(x);
plot(x,y)
```

La prima istruzione crea un vettore x=[a a+h a+2h ... b_0] dove b_0=b se esiste un numero naturale *n* tale che b=a+*n*h, altrimenti b-h<b_0<b.
La seconda istruzione crea un vettore y=[f(a) f(a+h) ... f(b_0)].

oppure (quando è impossibile vettorizzare)

```
x=a:h:b;
for i=1:size(x,2)
   y(i)=f(x(i));
end
plot(x,y)
```

Il comando *plot* disegna sullo schermo i punti (x_i,y_i), i=1,…,size(x,2) *unendoli con una linea retta.*

Esistono numerose opzioni collegate al comando *plot*:

`plot(x,y,'r')`	colora il grafico di rosso
`plot(x,y,'g')`	colora il grafico di verde
`plot(x,y,'-o')`	disegna il grafico a pallini uniti da una linea retta

`plot(x,y,'gs')` disegna il grafico a quadratini e lo colora di verde

Il comando *hold on* blocca la chiusura della finestra nella quale è comparsa l'ultima figura e permette di disegnare nuove figure sovrapponendole alla precedente.

Il comando *figure,* al contrario, fa si che il nuovo grafico venga disegnato in una nuova finestra.

ATTENZIONE: nelle assegnazioni vettoriali tipo *y=f(x);* è fondamentale distinguere tra le operazioni
`*  /  \  ^`
e
`.* ./  .\  .^`

Altre potenzialità di Matlab:

inf

Nei più comuni linguaggi di programmazione come Pascal, C, Fortran, esiste la possibilità di *overflow.* L'*overflow* consiste nel tentativo di assegnare ad una variabile un valore più grande del massimo valore memorizzabile nello spazio di memoria riservato a quella variabile. Quando ciò accade, il compilatore dà un messaggio di errore e arresta l'esecuzione del programma.
In Matlab, invece, in caso di *overflow* l'esecuzione non viene arrestata. Infatti, alla variabile incriminata viene assegnato il valore *inf* (infinito) ed essa può continuare ad essere utilizzata come una qualsiasi variabile, con le seguenti accortezze:
inf + *inf* = *inf*
inf + *a* = *inf* per ogni numero reale *a*
inf * *inf* = *inf*
a / *inf* = 0 per ogni numero reale *a*
a / *0* = *inf* per ogni numero reale *a*
inf - *inf* = *NaN* (vedi paragrafo seguente)
inf / *inf* = *NaN* (vedi paragrafo seguente)

NaN

Come *inf, NaN* è un valore che può essere assegnato ad una variabile e sta per "Not a Number". E', ad esempio, il risultato di
inf - inf
inf / inf
0 / 0

Esempio:
```
a=NaN;
if a<0
   disp('a<0');
elseif a>=0
   disp('a>=0');
else
   disp('non so');
end
```
Il risultato di questo programma è "non so".

Eval, Feval

Questi comandi permettono di valutare una stringa di caratteri come un'espressione matematica o come il nome di un m-file. Sono utilizzati - ad esempio - quando si vuole inserire l'espressione di una funzione run-time.

Esempio (Eval):
```
s=input('Inserire una funzione: f(x)= ');  %qua si inserisce una stringa
x=input('Inserire un valore per x. x= ');
y=eval(s);  % Ad y viene assegnato f(x)
```
Se s='x.^2+1' e x=2, si avra' y=5.
Se s='x.^2+1' e x=[2 4], si avra' y=[5 17].
Se s='sum(x)' e x=[1 2], si avra' y=3.

Esempio (Feval):
```
f=inline('x+(3*y)');
y=feval(f,2,4)  %feval valuta la funzione f per x=2 e y=4
```
Il risultato è y=14.

Esempio (Feval):
```
y=feval('FUNZ',5);
```
In questo caso `feval` chiama la function FUNZ.m passando come parametro di input il numero 5.

Si consiglia di leggere l'help in linea per le funzionalità più avanzate di questi comandi.
Debug:
Matlab dispone di un facile e potente debugger. Dall'Editor è possibile inserire e disinserire i *breakpoint* per segnalare le righe in cui si vuole sospendere l'esecuzione. Con il comando RUN (o F5 dalla tastiera) viene eseguito il codice fino al raggiungimento del primo *breakpoint* e l'editor entra automaticamente nella modalità DEBUG. Con i comandi *step* e *step in* si esegue il programma riga per riga, con il tasto *continue* si esegue il programma fino al successivo *breakpoint*.
Tenendo aperto il Workspace e/o l'Array Editor è possibile vedere il contenuto delle variabili run-time.
Dalla finestra *Breakpoints* dell'Editor si accede ad altri utili comandi per il DEBUG.

E' inoltre possibile eseguire solo una parte di un m-file. Per farlo è sufficiente selezionare la parte di codice da eseguire, cliccare con il tasto destro del mouse e selezionare la voce *Evaluate Selection*. Si deve però tenere presente che il testo selezionato deve contenere in sé l'assegnazione di tutte le variabili di cui si fa uso (deve cioè essere "sensato" eseguirlo senza il resto del programma).

Osserviamo infine che un primo semplice debug può essere ottenuto semplicemente facendo uso del comando *break* oppure con un uso intelligente del punto e virgola (`;`).

Scrittura su file:
Per salvare un file di dati si usa il comando *save*.
Esempio:
```
save nomefile variabile1 variabile2 variabile3
```
Dopo aver eseguito questo comando (nell'Editor o nella Command Window) viene creato nella Current Directory il file *nomefile.mat* (con estensione *mat*) dove sono memorizzate le variabili *variabile1 variabile2 variabile3* con il loro nome, tipo e valore.

Con il comando
```
save nomefile
```
viene salvato tutto il contenuto del workspace.

Attenzione! I file *.mat possono essere letti solo da Matlab.

Per esportare dati leggibili anche da altri compilatori come Fortran o C, è necessario salvare dei file con estensione .dat o .txt (file di testo).
Ad esempio, per salvare la variabile *a* in formato ascii in doppia precisione si usa il comando

```
save nomefile.dat a –ascii -double
```
Successivamente è possibile aprire il file *nomefile.dat* con un qualsiasi editor di testo (ad es. BLOCCO NOTE) per controllare il tipo di formattazione usata da Matlab.

Lettura da file:
Con il comando
```
load nomefile
```
si caricano nel Workspace tutte le variabili memorizzate in *nomefile.mat* conservandone il nome, il tipo ed il valore.

Per caricare nel Workspace un file *nomefile.dat* e memorizzarne il contenuto nella variabile *a*, si procede nel seguente modo:
```
a=load('nomefile.dat');
```

OSSERVAZIONE: in Matlab esistono anche i comandi *fscanf* e *fprintf* con una sintassi del tutto analoga a quella del C.

Generazione automatica dei nomi dei file
Nel caso in cui fosse necessario salvare un grosso numero di file (ad es. *file_1.dat, file_2.dat, file_3.dat, ... , file_n.dat*) è possibile una generazione automatica dei nomi dei file.
Esempio:
```
for i=1:n
    nome=['file_' num2str(i) '.dat'];
    pf=fopen(nome,'w');
    fprintf(pf,'%d',10+i);
    fclose(pf);
end
```

Funzioni predefinite di altissimo livello:
Ricerca del minimo di una funzione (fminsearch)
Il comando *fminsearch* calcola il minimo (locale) di una funzione $f:R^n \rightarrow R$ con il metodo del simplesso (per $n=1$ si ottiene una variante del metodo di bisezione). Non si fa uso del gradiente (analitico o numerico) della funzione.
Oltre alla funzione da minimizzare, si deve indicare un punto *x* di partenza per l'algoritmo, che deve ovviamente essere il più vicino possibile al minimo che si vuole localizzare.
E' inoltre possibile specificare numerose opzioni.
Esempio:
Si vuole calcolare il *min* e l'*argmin* della funzione *sin(x)*, visualizzando il risultato ad ogni iterazione e arrestando il procedimento quando si è raggiunta una tolleranza di 10^-5 su $f(x_{min})$ e di 10^-3 su x_{min}. Il punto iniziale è $x=0$.

```
options = optimset('Display','iter','TolFun',1.e-5,'TolX',1.e-3);
[x,val]=fminsearch('sin(x)',0,options);
x       %argmin
val     %min
```

Se *x* è un vettore *n*-dimensionale, è sufficiente specificare come punto di partenza un vettore *n*-dimensionale. La funzione *f(x)* deve comunque restituire uno scalare.
Esempio:
```
[x,val] = fminsearch('x(1)^2+x(2)^2',[1 1]);
```

Ricerca degli zeri di un polinomio (roots, poly)
Il comando
```
r=roots(p);
```
riceve in input un vettore *p* e restituisce in output un vettore *r* i cui elementi sono le radici del polinomio che ha per coefficienti gli elementi del vettore *p* (*p(1)* è il coefficiente del termine di grado massimo).
L'algoritmo fa uso degli autovalori della *companion matrix.*

Il comando
```
p=poly(r);
```
è l'inverso di *roots*. Esso riceve in input un vettore *r* e restituisce in output un vettore *p* i cui elementi sono i coefficienti del polinomio le cui radici sono gli elementi di *r*.

Ricerca degli zeri di una funzione generica (fzero)
Il comando *fzero* calcola uno zero di una funzione $f:R \rightarrow R$ assegnata, partendo da un punto vicino.
L'algoritmo è basato su una combinazione del metodo di bisezione, delle secanti e dell'interpolazione quadratica inversa.
A partire dal punto iniziale x_0, l'algoritmo calcola con successive iterazioni un intervallo [a,b] tale che comprenda x_0 e che $f(a)f(b)<0$. Successivamente, riduce progressivamente questo intervallo fino a localizzare lo zero della funzione.

Esempio:
```
options = optimset('Display','iter');
[x,val]=fzero('x^2-1',4,options);    % 4 è x0
x       % approssimazione dello zero della funzione
val     % f(x)
```

Attenzione: il comando *fzero* considera zeri di una funzione *f* solamente i punti in cui *f* passa da valori positivi a valori negativi e non i punti in cui la funzione tocca solamente l'asse delle *x*. Ad esempio, la funzione x^2 non ha zeri.

Formule di quadratura (quad, quadL, dblquad, triplequad)

`quad`	calcola l'integrale di una funzione $f:R \rightarrow R$ o $f:R \rightarrow R^n$ con la regola di Simpson adattiva
`quadL`	calcola l'integrale di una funzione $f:R \rightarrow R$ o $f:R \rightarrow R^n$ con una formula di Newton-Cots adattiva più accurata di quella usata in *quad*
`dblquad`	calcola integrali doppi
`triplequad`	calcola integrali tripli

Esempio:
Calcolare l'integrale di sin(x) tra 0 e 1 con una tolleranza minima di 10^-6.
```
q=quad('sin(x)',0,1,10^(-6));
```

Interpolazione (interp1, interp2, ecc.)
Con Matlab è possibile interpolare una funzione conosciuta solo in un numero finito di punti, sia essa di una, due, tre o *n* variabili. E' inoltre possibile scegliere tra diversi metodi (interpolazione lineare, cubica, spline, ecc.).
Per una funzione di una variabile, il comando è
```
yy=interp1(x,y,xx);
```
dove
x e *y* sono due vettori contenenti le ascisse e le ordinate della funzione (i « dati » del problema),
xx è il vettore contenente i punti nei quali si vuole interpolare la funzione e
yy è il vettore risultante contenente i valori della funzione nei nodi specificati in *xx*.

ODE (ode45, ode23, ecc.)
In Matlab esistono numerosi comandi per risolvere un'equazione differenziale ordinaria di primo grado. Essi possono essere usati per risolvere singole equazioni o sistemi di ODE e possono essere richiamati con un enorme numero di "options" diverse.
Per semplicità proponiamo qui di seguito solo la versione base dei comandi *ode23* e *ode45.*

Sia *ode23* che *ode45* implementano il metodo di Runge-Kutta, ma *ode45* è più accurato. Entrambi sono schemi esplici ad un passo.

Esempio 1:
Supponiamo di voler risolvere l'equazione:
$y'(t) = F(t,y) = t+y, \quad t \text{ in } [0,T_f]$
$y(0) = y_0$

Si procede nel seguente modo:

1) si definisce una function F.m con l'espressione della funzione *F*:
```
function dy=F(t,y)
dy=t+y ;
```
2) Nel *main* si richiama il comando:
```
[T,Y]=ode45(@F,[0 Tf], y0);
plot(T,Y,'-o')
```

Esempio 2 (modello preda-predatore):
Supponiamo di voler risolvere l'equazione:
$dy_1(t)/dt = 2y_1-3(y_1*y_2), \quad t \text{ in } [0,T_f]$
$dy_2(t)/dt = -3*y_2+2*(y_1*y_2), \quad t \text{ in } [0,T_f]$
$y_1(0) = 0.5$
$y_2(0) = 0.6$

Si procede nel seguente modo:

1) si definisce una function F.m con l'espressione del campo vettoriale:
```
function dy=F(t,y);
dy(1)=2*y(1)-3*(y(1)*y(2));
dy(2)=-3*y(2)+2*(y(1)*y(2));
dy=dy';   %l'output deve essere un vettore colonna
```

2) Nel *main* si richiama il comando:
```
[T,Y]=ode45(@F,[0 Tf],[0.5 0.6]);
plot(T,Y(:,1),'-o')   %disegna y1(t)
hold on
plot(T,Y(:,2),'-ro')  %disegna y2(t) in rosso
figure
plot(Y(:,1),Y(:,2),'g')  %disegna la curva (y1(t),y2(t)) in verde
```

Calcolo parallelo e distribuito in Matlab:
Il parallel computing toolbox
http://www.mathworks.co.kr/access/helpdesk/help/toolbox/distcomp/
permette le operazioni basilari di calcolo parallelo. Il comando `pmode start` indica a Matlab di parallelizzare tutti i comandi che supportano questa funzione.

OSSERVAZIONE: i processori **dual/quad core** sono considerati dal sistema operativo come due/quattro processori a tutti gli effetti.

Matlab da remoto con SSH:
Se si possiede un account sulle macchine SUN del dipartimento, si può utilizzare Matlab da remoto (cioè da un qualsiasi altro computer con connessione ad Internet) attraverso il programma SSH (o simili, ad es. PUTTY) scaricabile gratuitamente dalla rete.
Una volta collegati al server dove si ha l'account (ad es. plutone.mat.uniroma1.it) e inseriti il proprio nome utente e password, si può lanciare Matlab (in modalità NON grafica) con il comando
matlab
In questo modo si apre esclusivamente la Command Window, dalla quale si possono lanciare gli m-file precedentemente salvati nella Current Directory.
ATTENZIONE: la versione di Matlab installata sulle macchine SUN è in generale diversa da quella presente sui PC con Windows e Linux.

Per lanciare un programma in modalità BATCH (cioè mandare in esecuzione un m-file e poi disconnettersi dal server senza interrompere l'esecuzione) si usa il comando *at* (digitare in un x-terminal *man at* o *man batch* per conoscere la sintassi).

Esempio (lancio del file prova.m):
Digitare nella finestra
`at -short` (oppure `medium` oppure `long`, vedi sotto per il significato) `now`
e premere ENTER.
Digitare
`matlab < prova.m`
e premere ENTER.
Premere la sequenza di tasti CTRL-D.

`short`	se si prevede una durata inferiore a 20 minuti
`medium`	se si prevede una durata tra 20 minuti e 4 ore
`long`	se si prevede una durata superiore a 4 ore

Per ritardare l'esecuzione di può aggiungere un'indicazione temporale:
`at -short now + n minutes` (oppure `hours` oppure `days`)
`matlab < prova.m`
CTRL-D

Una volta terminata l'esecuzione del programma, il server invia automaticamente una e-mail all'indirizzo *nomeutente@mat.uniroma1.it* con la schermata finale della Command Window.

Per ottenere in output anche il tempo impiegato dal sistema per eseguire il programma (real, user, sys) si usa il seguente comando:
```
at -short now
/usr/bin/time matlab < prova.m
```
CTRL-D

Altri comandi utili:

`atq`	per vedere la lista dei job in coda.
`atrm -a`	rimuove i job in coda non ancora eseguiti.

Bibliografia:

- D.F. Griffiths, "An Introduction to MATLAB", scaricabile da http://www.maths.dundee.ac.uk/~ftp/na-reports/MatlabNotes.pdf
- MATLAB The Language of Technical Computing, The Math. Works, Inc