FILE, FUNZIONI E STRUTTURE DATI

Si possono distinguere tre tipi di file che vengono utilizzati in MatLab:

1. M-file
2. Mat-file
3. File Dati

Gli M-file hanno estensione .m e in essi vengono memorizzati i programmi e le funzioni di MatLab. I Mat-file hanno estensione .mat e sono utilizzati per salvare i nomi e i valori delle variabili create durante una sessione di lavoro MatLab: si tratta di file binari che possono essere letti solo dal software che li ha creati, quindi non è possibile utilizzare un word processor per leggerli. I file dati sono file di dati in formato ASCII utili per analizzare i dati registrati in un file creato da un programma per spreadsheet, da un word processor o da un sistema di acquisizione dati da laboratorio.

Il comando diary permette di registrare l'attività svolta durante una sessione di lavoro con MatLab, in un file speciale chiamato file diario. Una volta digitato il comando diary tutte le operazioni e i comandi eseguiti in una sessione di lavoro verranno memorizzati in un file diario di formato ASCII: per visualizzare il suo contenuto basta digitare type diary. È anche possibile caricare il file diario in un word processor per modificarlo, ed eventualmente includerlo in un altro documento.

Per salvare e caricare le variabili utilizzate in una sessione di MatLab si devono usare i comandi save e load. Il comando save permette di salvare le variabili dello spazio operativo (nomi, dimensioni e valori) in un file binario chiamato Matlab.lab, che MatLab è in grado di leggere. Per caricare variabili precedentemente salvate è sufficiente digitare il comando load con la possibilità di riprendere una sessione precedentemente interrotta.

FUNZIONI MATEMATICHE INTRINSECHE

Possono essere applicate a scalari, vettori e matrici ed agiscono elemento per elemento.

`sqrt(x)`	$\sqrt{x}$
`round(x)`	arrotondamento: x = 3.6 → 4
`fix(x)`	troncamento: x = 3.6 → 3
`sign(x)`	segno di x (vale 1,0 o -1)
`sin(x)`	sinx
`cos(x)`	cosx
`tan(x)`	tanx
`sinh(x)`	sinhx
`cosh(x)`	coshx
`tanh(x)`	tanhx
`asin(x)`	arcsinx
`acos(x)`	arccosx
`atan(x)`	arctanx
`exp(x)`	e^x
`log(x)`	$\log_e x$ (logaritmo naturale di x)
`log10(x)`	$\log_{10} x$ (logaritmo in base 10 di x)
`real(z)`	parte reale di

```
imag(z)  parte immaginaria di z
conj(z)  z* (complesso coniugato di z)
```

Per avere informazioni su una particolare funzione è possibile utilizzare il comando lookfor di MatLab. Ad esempio per avere un elenco delle funzioni che operano con i numeri immaginari, basta digitare lookfor imaginary e sullo schermo apparirà il seguente prospetto:

```
>> lookfor imaginary

   I  Imaginary unit.
   J  Imaginary unit.
        COMPLEX Construct complex result from real and imaginary parts.
   IMAG   Complex imaginary part.
   IMAG   Symbolic imaginary part.
```

Imaginary non è una funzione di MatLab, ma la parola si trova nella descrizione della guida relative alla funzione imag, e ai simboli speciali I e J. Se si conosce il nome esatto di una funzione di MatLab, per esempio disp, è possibile digitare help disp per avere informazioni sulla funzione.

FILE SCRIPT E EDITOR DEBUGGER

In MatLab è possibile operare in due modi diversi:

1. In MODALITÀ INTERATTIVA;
2. Eseguire un programma di MatLab registrato in un M-file.

Eseguire un M-file equivale a digitare tutti i comandi in esso contenuti, uno alla volta, dopo il prompt di MatLab e per eseguirlo basta digitare il suo nome dopo il prompt di Matlab.

Gli M-file si distinguono in:

1. File script, che contengono una sequenza di comandi di MatLab;
2. File di funzioni, utili per ripetere più volte una sequenza di comandi.

I file script possono contenere qualsiasi comando, incluse le funzioni definite dall'utente. I valori delle variabili prodotte dall'esecuzione di un file script sono resi disponibili nella sessione di MatLab, si tratta cioè di variabili globali.

Come creare e utilizzare un file script

Il procedimento che permette di creare, salvare ed eseguire un file script può essere così schematizzato:

- Selezionare New dal menu File e poi M-file;
- Digitare il file, ad esempio:

```
% Programma Example1.m
% questo programma calcola il seno della radice quadrata
% di vari numeri e visualizza i risultati sullo schermo

x=sqrt([3:2:11]);
y=sin(x)
```

- Selezionare l'opzione Save dal menu File e salvare il file nella directory corrente di MatLab;
- Digitare il nome del file nella finestra dei comandi, in questa caso Example1: i risultati vengono visualizzati nella finestra di comandi.

Un altro esempio di file script in cui sono memorizzate le istruzioni che permettono di calcolare l'approssimazione di e^2 usando i primi n termini dello sviluppo in serie di Taylor è il seguente:

```
%  file esponen.m
n=input('Numero dei termini della serie ');
s=0;
for k=0:n-1
        a=2^k/prod(1:k);
        s=s+a;
end
disp('il valore approssimato di e^2 è ');disp(s);
disp('errore assoluto = ');disp(exp(2)-s);
```

Debugging dei file script

Il termine debugging indica il processo di ricerca e correzione degli errori o "bug" di un programma, questi errori spesso appartengono ad una delle seguenti categorie:

ERRORI DI SINTASSI: omettere una parentesi o una virgola o digitare in modo errato il nome di un comando. MatLab è in grado di rilevare gli errori più evidenti e visualizza un messaggio che descrive l'errore e la sua posizione;

```
>> y=sen(pi)
     ??? Undefined function or variable 'sen'.
```

ERRORI DI RUNTIME: dovuti ad una procedura matematica sbagliata; si tratta di errori che si verificano quando un programma viene eseguito.

```
??? Input argument 'b' is undefined.

Error in ==> E:\LabCompNum\back_sost.m
On line 15  ==> nb=length(b);
```

Funzioni definite dall'utente

Nei file di funzione, diversamente dai file script, tutte le variabili utilizzate sono locali e cioè i loro valori sono disponibili solo all'interno della funzione stessa. La prima riga di un file di funzione deve iniziare con la definizione della funzione che contiene un elenco di variabili di Input e di Output. Questa riga distingue un file di funzione da un file script e la sua sintassi è la seguente:

```
function [variabili di output] = nome_function(var_input);
```

Si noti che le variabili di output sono racchiuse tra parentesi quadre, mentre quelle di input tra parentesi tonde. Il nome della funzione deve essere uguale al nome del file con cui verrà salvata la funzione (.m), la funzione verrà richiamata digitando il suo nome dopo il prompt di MatLab. Un esempio di struttura delle funzioni può essere:

```
function [y1, ... , yn] = nomefunction (x1, ... , xm)
%  commenti
istruzioni
% assegnazione  valori ai parametri di output
y1 = valore1;
y2 = valore2;
...
yn = valoren;
return
```

dove y1,...,yn sono i dati di output e x1,...,xm sono i dati di input.

Le righe di commento che iniziano con il simbolo % possono essere poste in qualsiasi punto all'interno del file di funzione. Utilizzando il comando help per ottenere informazioni sulla funzione, MatLab visualizza tutte le righe di commento che si trovano immediatamente dopo la riga di definizione della funzione.

Le variabili di input specificate nella riga di definizione della funzione sono locali, ciò significa che nella chiamata alla funzione è possibile usare nomi di variabili diversi quando la funzione viene chiamata. Tutte le variabili usate all'interno di una funzione vengono cancellate dopo che la funzione è stata eseguita, tranne quando gli stessi nomi di variabili figurano nell'elenco di variabili di output utilizzate nella chiamata alla funzione. Qualora sia necessario, è possibile dichiarare come globali alcune variabili e per farlo si deve utilizzare il comando global. I valori delle variabili dichiarate di tipo globale sono disponibili solo per l'area di lavoro principale di MatLab e per le funzioni che dichiarano queste variabili come globali. Ad esempio la sintassi per dichiarare globali le variabili a, x e y è la seguente:

```
global   a x q.
```

Un esempio di function che realizza l'algoritmo di sostituzione all'indietro per sistemi con matrice triangolare superiore che riceve in input la matrice triangolare superiore U e il vettore dei termini noti b è il seguente:

```
function x=back_sost(U,b)
nb=length(b);
[n,m]=size(U)
if((n ~=m)|(n ~=nb))
   disp('dimensioni non corrette');
   return
end
%   verifica che U non sia singolare
if prod(diag(U))==0
   disp('matrice singolare')
   return
end
%   sostituzione  all'indietro
x=zeros(nb,1)
x(nb)=b(nb)/U(n,n)
for i=n-1:-1:1,
   p=U(i,n:-1:i+1)*x(n:-1:i+1);
   x(i)=(b(i)-p)/U(i,i);
end
return
```

MATLAB COME LINGUAGGIO DI PROGRAMMAZIONE

La modalità interattiva di MatLab è molto utile per risolvere problemi semplici, mentre problemi più complessi richiedono l'utilizzo di file script. Un file script può essere considerato un "programma per computer": scrivere questo tipo di file significa "programmare " con MatLab.

MATLAB può essere considerato un linguaggio di programmazione, alla stregua del Fortran, del C o del Pascal. L'esistenza di strutture sintattiche tradizionali, unitamente all'uso appropriato delle funzioni intrinseche, consente di codificare in modo semplice e flessibile gli algoritmi classici dell'Analisi Numerica, così come programmi di calcolo dedicati al trattamento di problemi specifici.

INPUT/OUTPUT

MatLab dispone di vari comandi che permettono di ottenere l'input degli utenti e formattare i dati di output

Comando	Descrizione
disp(A)	Visualizza il contenuto, non il nome, dell'array A.
disp('testo')	Visualizza la stringa di teso racchiusa tra i due apici '.
format	Controlla il formato di visualizzazione dell'output.
fprintf	Controlla il formato dell'output sullo schermo o su file.
x=input('testo')	Visualizza il testo specificato, attende l'input dell'utente dalla tastiera e registra il valore nella variabile x.
x=input('testo',' s')	Visualizza il testo specificato, attende l'input dell'utente dalla tastiera e lo registra come stringa nella variabile x.

Il comando disp permette di visualizzare una stringa di testo sullo schermo, la sua sintassi è:

disp(stringa di caratteri)

Un primo esempio: disp(a)

Visualizza il contenuto, non il nome, dell'array a

```
>> a=[2 1 ; -1 3];
>> disp(a)
    2    1
   -1    3

>> a=[1:0.1:1.5];
>> disp(a)
  Columns 1 through 5

    1.0000    1.1000    1.2000    1.3000    1.4000

  Column 6

    1.5000
>>
```

Il comando input permette di leggere il valore di una variabile da tastiera, la sua sintassi è:

Variabile = input (stringa di caratteri)

```
>>  n = input ( `Dimensione massima delle matrici: `);
Dimensione massima delle matrici:   21
```

N.B. Nei vettori colonna è importante ricordarsi che le stringhe devono avere tutte la stessa dimensione.

Un modo più completo per l'output di testo con formato è fornito dalle funzioni fprintf e sprintf, la loro sintassi è:

```
sprintf (fid , formato, variabili)
stringa = sprintf (formato, variabili)
```

La seguente tabella indica i formati di visualizzazione del comando fprintf.

Comando	Descrizione
fprintf('formato',A,....)	Visualizza gli elementi dell'array A e tutti gli argomenti aggiuntivi dell'array, secondo il formato specificato nella stringa 'formato'
struttura di 'formato'	%[-][numero1.numero2]C, dove numero1 specifica la lunghezza minima di campo, numero2 specifica il numero di cifre decimali e C contiene i codici di controllo e di formattazione. Gli elementi fra parentesi quadre sono facoltativi. [-] specifica l'allineamento a sinistra.

Codici di controllo		Codici di formattazione	
\n	Avvia una nuova riga	%s	Formato stringa
\r	Inizio di una nuova riga	%d	Formato intero
\b	Backspace	%f	Formato decimale
\t	Tabulatore	%e	Notazione scientifica con e minuscola
''	Apice o apostrofo	%E	Notazione scientifica con E maiuscola
\\	Backslash	%g	Il più corto tra i formati %e e %f

```
% TABELLA.m
%  realizza una tabella delle funzioni:
%         f_1(x)=x^2       f_2(x)=x^3
%  nell'intervallo [0,1]
%-----------------------------------------------
%
np=input('dammi il numero dei nodi :  ');
x=linspace(0,1,np);
```

```
f1=x.^2;
f2=x.^3;
disp('---------------------------------');
fprintf('i\t  x(i)\t  x(i)^2\t  x(i)^3\n');
disp('---------------------------------');
fprintf('%d\t  %1.4f\t  %1.5f\t  %1.5f\n',[1:np;x;f1;f2]);
```

che produce il seguente output

```
>> tabella
dammi il numero dei nodi :  5
-------------------------------------------------
i          x(i)          x(i)^2          x(i)^3
-------------------------------------------------
1        0.0000        0.00000        0.00000
2        0.2500        0.06250        0.01563
3        0.5000        0.25000        0.12500
4        0.7500        0.56250        0.42188
5        1.0000        1.00000        1.00000
```

È possibile immettere i dati in MatLab digitandoli direttamente in un array, o in alternativa è possibile registrare i dati in un M-file; entrambi i metodi quando i dati da elaborare sono numerosi. Di solito questi dati vengono generati da applicazioni esterne a MatLab e siccome la maggior parte delle applicazioni supportano il formato ASCII, è molto probabile che i file dati vengano registrati in questo formato: per caricarli in MatLab è sufficiente digitare il comando load nomefile.

Alcuni programmi per spreadsheet registrano i dati nel formato wk1; per importare questi dati in MatLab e registrarli in una matrice M si deve utilizzare il comando:

```
>> M = wk1read('nomefile');
```

Il comando

```
>> A = xlsread('nomefile');
```

registra nella matrice A il file nomefile.xls di una cartella di Microsoft Excel.

```
% TABELLA.m
%  realizza una tabella delle funzioni:
%          f_1(x)=x^2       f_2(x)=x^3
%  nell'intervallo [0,1]
%----------------------------------------
np=input('dammi il numero dei nodi :');
x=linspace(0,1,np);
f1=x.^2;
f2=x.^3;
disp('---------------------------------');
fprintf('i\t   x(i)\t   x(i)^2\t   x(i)^3\n');
disp('---------------------------------');
fprintf('%d\t   %1.4f\t   %1.5f\t   %1.5f\n',[1:np;x;f1;f2]);
```

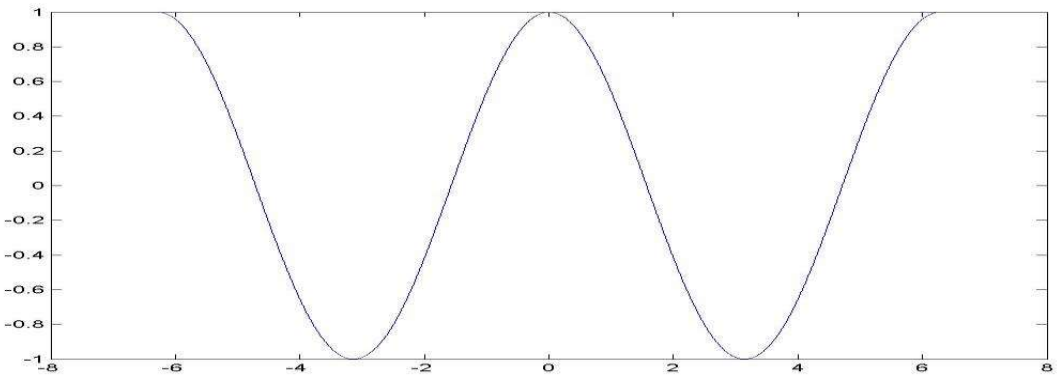
che produce il seguente output

```
>> tabella
dammi il numero dei nodi :5
-------------------------------------------------
i            x(i)              x(i)^2            x(i)^3
-------------------------------------------------
1          0.0000            0.00000           0.00000
2          0.2500            0.06250           0.01563
3          0.5000            0.25000           0.12500
4          0.7500            0.56250           0.42188
5          1.0000            1.00000           1.00000
```

(Output di tipo grafico)

Per rappresentare il grafico di f(x) = cos(x) con $x \in [-2\pi, 2\pi]$ diamo i seguenti comandi:

```
>>  x = -2*pi : 0.01 : 2*pi ;
>>  y = cos(x);
>> plot(x,y);
```



Strutture di programmazione elementari di MATLAB

MatLab dispone di sei **operatori relazionali** che consentono di confrontare variabili ed array:

Operatore Relazionale	Descrizione
<	Minore
<=	Minore o uguale
>	Maggiore
>=	Maggiore o uguale
==	Uguale
~=	Diverso (Il simbolo **~** si ottiene tenendo premuto *Alt* e digitando 126 sul tastierino numerico)

Quando gli operatori relazionali vengono applicati tra array (della stessa dimensione) mettono a confronto i singoli elementi di un array con i corrispondenti elementi dell'altro array. Il confronto può avvenire anche tra un singolo scalare ed un array: in tal caso tutti gli elementi dell'array vengono confrontati con lo scalare. La seguente sessione di MatLab illustra alcuni esempi:

```
>> x=[6 3 9];
>> y=[14 2 9];
>> z=(x<y)

z =
    1    0    0

>> z=(x~=y)

z =
    1    1    0

>> z=(x>8)

z =
    0    0    1
```

Gli operatori relazionali sono anche usati per selezionare gli elementi di un array. Sempre nell'esempio precedente digitando

```
>> z=x(x<y)
```

si ottengono tutti gli elementi di x che sono minori dei corrispondenti elementi di y, e cioè:

```
z =

    6
```

MatLab possiede inoltre tre operatori logici che agiscono a livello dei singoli elementi degli array:

Operatore	Nome	Definizione
~	NOT	L'istruzione ~A restituisce un array delle stesse dimensioni di A; gli elementi del nuovo array sono pari a 1 se quelli di A sono nulli, altrimenti sono pari a 0.
&	AND	L'istruzione A & B restituisce un array delle stesse dimensioni di A e B; gli elementi del nuovo array sono pari a 1 se i corrispondenti elementi di A e B sono entrambi diversi da a 0, altrimenti sono pari a 0.
|	OR	L'istruzione A & B restituisce un array delle stesse dimensioni di A e B; gli elementi del nuovo array sono pari a 1 se almeno uno dei due elementi corrispondenti di A e B è diverso da a 0; sono pari a 0 se entrambi altrimenti di A e B sono nulli.

Esempi

```
>> x=[6 3 9];
>> y=[14 2 9];
>> z=~x

z =
    0    0    0

>> z=~(x>y)
z =
    1    0    1

>> z=(x<=y)
z =
    1    0    1
```

La seguente sessione di MatLab genera la tabella di verità in termini di 1 (vero) e 0 (falso).

```
>> x=[1,1,0,0]';
>> y=[1,0,1,0]';
>> Tabella_verita=[x, y, ~x, x&y, x|y, xor(x,y)]

Tabella_verita =

    1    1    0    1    1    0
    1    0    0    0    1    1
    0    1    1    0    1    1
    0    0    1    0    0    0
```

Strutture condizionali

- La forma base dell'istruzione **if** è la seguente:

```
if condizione
    istruzioni
end
```

La condizione può essere uno scalare, un vettore o una matrice. Per esempio se x è uno scalare e si volesse calcolare $y = x^{1/2}$ soltanto per x>0 si potrebbe implementare la seguente procedura:

```
if x>=0
   y=sqrt(x)
end
```

La condizione può essere anche un'espressione composta; le istruzioni possono essere formate da un singolo comando o da una serie di comandi separati da un punto e virgola, come illustra il seguente esempio:

```
z=0; w=0;
if (x>=0)&(y>=0)
   z=sqrt(x)+sqrt(y)
   w=log(x)-3*log(y)
end
```

- É possibile annidare le istruzioni **if** come segue:

```
if condizione 1
   istruzioni 1
else
   istruzioni 2
end
```

Ad esempio supponendo di voler calcolare $y = x^{1/2}$ per x>=0 e $y = e^x - 1$ per $x<0$, le seguenti istruzioni calcolano il valore di y:

```
if x>=0
   y=sqrt(x)
else
   y=exp(x)-1
end
```

Un altro esempio: cerco il massimo tra tre numeri a, b, c

```
if a > b
   max=a;
else
   max=b;
end
if c > max
   max=c;
end
```

- La struttura base dell'istruzione **elseif** è la seguente:

```
if  condizione 1
      istruzioni 1
elseif  condizione 2
      istruzioni 2
end
```

Se per esempio si vuole calcolare $y = ln\ x$ per $x \geq 5$ e $y = x^{1/2}$ per $0 \leq x < 5$, le seguenti istruzioni calcolano il valore di y:

```
if x>=5
   y=log(x)
elseif x>=0
   y=sqrt(x)
end
```

Dati i valori x e h si vuole calcolare la quantità w=y+h dove:

$$y = \begin{cases} x^3 & se \quad x < 0 \\ x(x-1) & se \quad 0 \leq x \leq 1 \\ x-3 & se \quad x > 1 \end{cases}$$

si ha:

```
if x < 0
 y=x^3;
elseif x < 1
 y=x*(x-1);
else
y=x-3;
end
w=y+h
```

```
%-------------------------------------------------------------------
%  Sottoprogramma che calcola il massimo e il minimo
%  fra tre numeri reali a, b, c
```

```
%
%  INPUT   i parametri di ingresso sono i numeri:
%              a, b, c
%
%  OUTPUT  i parametri di uscita sono i valori:
%                 min, max
%-------------------------------------------------------------------
function [min,max]=maxmin(a,b,c)
if a > b
   max=a;
else
   max=b;
end
if c > max
   max=c;
end
if a > b
   min=b;
else
   min=a;
end
if c < min
   min=c;
end
return
%-------------------------------------------------------------------


%-------------------------------------------------------------------
%  programma principale per il sottoprogramma maxmin
%-------------------------------------------------------------------
disp('determino il minimo e il massimo tra 3 numeri reali')
disp('mi devi dare tre numeri reali: a,b,c')
a=input('dammi a  ');
b=input('dammi b  ');
c=input('dammi c  ');
[min,max]=maxmin(a,b,c);
fprintf('min\t max\t')
fprintf('%4.1f\t %4.1f\t', [min,max])
%-------------------------------------------------------------------
```

Le strutture decisionali possono essere annidate, cioè una struttura può essere inclusa all'interno di un'altra struttura che, a sua volta può contenerne un'altra e così via.

- La struttura di base del ciclo a contatore **for** è la seguente:

```
for i = valoreiniziale : incremento : valorefinale
    istruzioni
end
```

Un semplice esempio di ciclo for potrebbe essere il seguente:

```
for k=5:10:35
   x=k^2
end
```

La variabile di ciclo k è inizialmente uguale a 5; il valore di x viene calcolato mediante la formula x=k^2 (k elevato al quadrato). Ogni passaggio del ciclo incrementa k di 10 e calcola x finché k non supera 35; dunque k assume i valori 5, 15, 25, 35, mentre x assume i valori 5, 225, 625 e 1225.
È possibile annidare i cicli for e le istruzioni condizionali, come illustra il seguente esempio, in cui viene costruita una matrice quadrata che ha il valore 1 in tutti gli elementi della prima colonna e della prima riga e che i restanti elementi siano la somma di due elementi: l'elemento sopra e quello a sinistra, se la somma è minore di 20; altrimenti l'elemento è il massimo tra i due elementi.

```
function A = specmat (n)
A = ones(n);
for r = 1:n
   for c = 1:n
      if (r>1)&(c>1)
         s = A(r-1,c)+A(r,c-1);
         if s < 20
            A(r,c) = s;
         else
            A(r,c) = max(A(r-1,c),A(r,c-1));
         end
      end
   end
end
```

Digitando A=specmat(5) si ottiene la seguente matrice:

```
>> A=specmat(5)

A =

    1    1    1    1    1
    1    2    3    4    5
    1    3    6   10   15
    1    4   10   10   15
    1    5   15   15   15
```

Il seguente codice calcola il valore di: S=N(N-1)...(N-K)

```
n=input('dammi il valore iniziale  ');
f=input('dammi il valore finale  ');
s=n;
for i=n-1:-1:f
```

```
s=s*i;
end
stringa=['s=  ',num2str(s)];  % conversione
disp(stringa)                          % output


%-------------------------------------------------------------
%   I cicli  FOR...END   possono essere nidificati.
%   In questo esempio stampo le terne (i,j,k)
%   con la proprietà:
%       1<= i <= j <= 5  e k=i^2+j^2
%-------------------------------------------------------------
for i=1:5
   for j=i:5
      k=i^2+j^2;
      fprintf('%d\t %d\t %d\t\n', i,j,k);
   end
end
%-------------------------------------------------
1         1         2
1         2         5
1         3         10
1         4         17
1         5         26
2         2          8
2         3         13
2         4         20
2         5         29
3         3         18
3         4         25
3         5         34
4         4         32
4         5         41
5         5         50
```

Scrivere un programma, che dato n>1, calcoli:

$$m = \sum_{i=1}^{n} i^2 \qquad \ell = \sum_{i=5}^{n} 2i \qquad k = \sum_{i=1}^{n} \frac{i-1}{2}$$

```
%-------------------------------------------------------------
%   Programma SOMMA
```

```
%
%   m=SUM_i=1^n (i^2);  l=SUM_i=5^n (2*i);  k=SUM_i=1^n (i-1)/2
%
%-------------------------------------------------------------
n=input('dammi il numero n  ');
% inizializzo  le variabili
m=0;
l=0;
k=0;
for i=1:n;
   m=m+i^2;
   if i>= 5
   l=l+2*i;
end
k=k+(i-1)/2;
end

% preparazione dell'output
stringa=sprintf('m=  %d l= %d  k= %f \n',m,l,k);
% ouput
disp(stringa)

dammi il numero n  8
m=  204 l= 52  k= 14.000000
```

- La struttura base del ciclo condizionato **while** è la seguente:

```
while  condizione
      istruzioni
end
```

Il ciclo while viene usato quando non si conosce in anticipo il numero di passaggi da effettuare: MatLab prima verifica se la condizione è vera, condizione che deve contenere una variabile di ciclo, e fintanto che tale condizione è verificata le istruzione specificate vengono eseguite una volta per ogni passaggio del ciclo, usando il valore corrente della variabile di ciclo.
È importante verificare che la variabile di ciclo abbia un valore appropriato prima che sia avviata l'elaborazione del ciclo. Per esempio, il seguente ciclo può dare risultati imprevisti se x ha già assunto erroneamente un valore che non è adeguato al compito corrente:

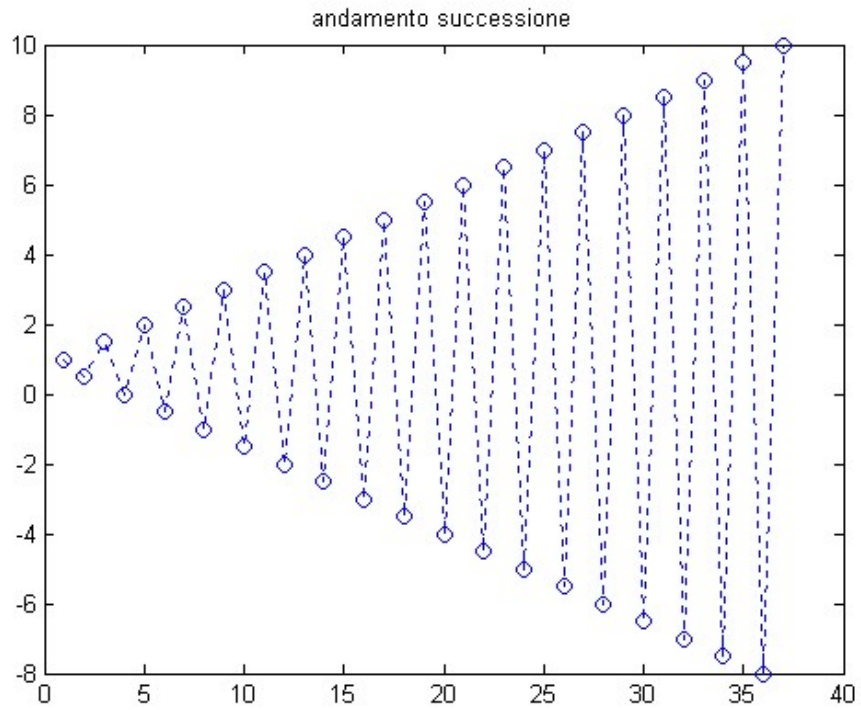
```
while x < 10
   x = x + 1;
   y = 2 * x;
end
```

Oppure si potrebbe generare un ciclo infinito:

```
x = 8;
while x ~= 0
   x = x - 3;
end
```

All'interno del ciclo la variabile x assume i valori 5, 2, -1, -4, …. , e la condizione x ~= 0 è sempre soddisfatta; quindi il ciclo non termina mai.

```
%-------------------------------------------------------------
%  Esempio del costrutto
%     WHILE   Condizione
%           blocco di istruzioni
%     END
%-------------------------------------------------------------
%  Calcolo la successione:
%
%  a(n+1)=a(n)+(-1)^n*n/2
%
%  partendo da a(1)=a_i, fino a quando a(n)<a_f
%-------------------------------------------
function [a,n]=successione(a_i,a_f)
a(1)=a_i;
n=1;
while a(n) < a_f
   a(n+1)=a(n)+(-1)^n*n/2;
   n=n+1;
end
return
%-------------------------------------------------------------
```



- Uscita incondizionata: **break**

- Blocco: **switch … case … end**

```
switch   Espressione
    case  Valore1
              blocco di istruzioni
   case  Valore 2
              blocco di istruzioni
```

```
        …
        otherwise
                blocco di istruzioni
    end
```

dove i relativi blocchi di istruzioni sono eseguiti solo se l'Espressione assume il corrispondente Valore. L'ultimo blocco di istruzioni sarà eseguito solo nel caso in cui Espressione non abbia assunto nessuno dei precedenti valori.

```
%---------------------------------------
%  switch ... case  ... end
%---------------------------------------
x=[0:0.01:2*pi]; y=sin(2.*x);
risposta=input('digita  1. minimo, 2. massimo , 3. sommma ')
switch risposta
case 1
   minimo=min(y)
case 2
   massimo=max(y)
case 3
   somma=sum(y)
otherwise
   disp('non hai fatto una scelta corretta')
end
%-------------------------------------------------------------------------
```

OSSERVAZIONI

1. La struttura condizionata non necessita dell'apposizione then dopo il comando if, a differenza di quanto accade in altri linguaggi.
2. Ad ogni comando if, for, while deve necessariamente corrispondere un comando end; questo è di particolare importanza in presenza di cicli annidati.
3. Tutte le istruzioni sopra introdotte si possono digitare su linee diverse oppure sulla stessa linea di comando, separate da virgole. La prima modalità di scrittura favorisce, evidentemente, la leggibilità di un codice, mediante uso opportuno dell'indentazione.
4. Quando le istruzioni di un programma sono molte è necessario salvarle su file, per poterle poi eseguire con un eventuale passaggio di parametri di input/output.

ESEMPI DI FUNZIONI

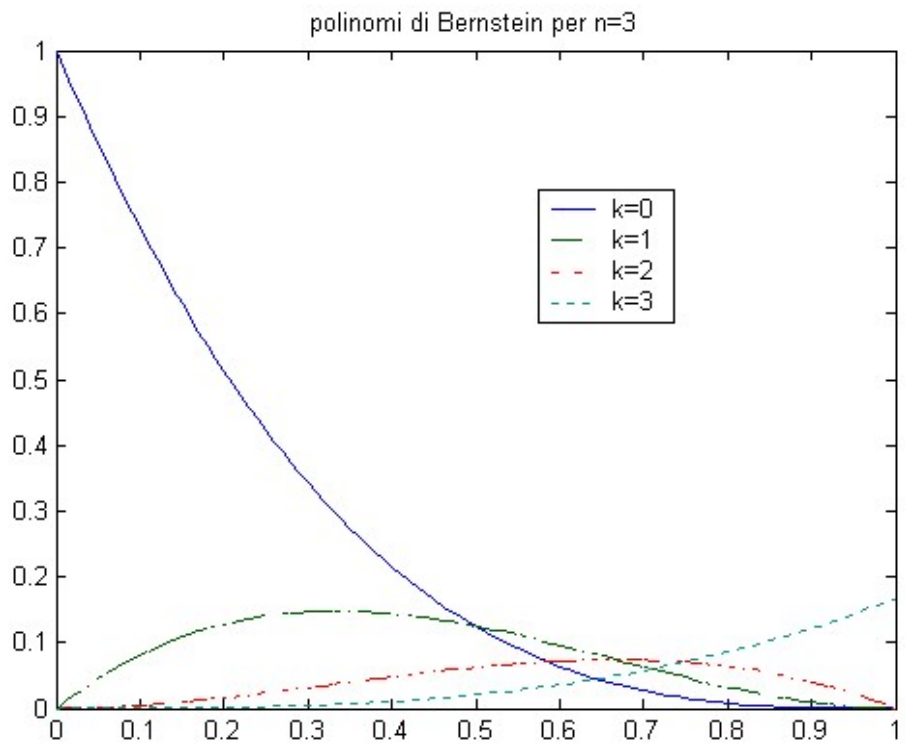
```
function x=back_sost(U,b)
%
%  x=back_sost(U,b)
%
%  Algoritmo di sostituzione all'indietro per
%  sistemi con matrice triangolare superiore
%
%  Input
%     U  matrice triangolare superiore (n x n)
%     b  vettore (n x 1)
```

```
%   Output
%       x   vettore soluzione
%-------------------------------------------------
%
nb=length(b);
[n,m]=size(U)
if((n ~=m)|(n ~=nb))
   disp('dimensioni non corrette');
   return
end
%   verifica che U non sia singolare
if prod(diag(U))==0
   disp('matrice singolare')
   return
end
%   sostituzione all'indietro
x=zeros(nb,1)
x(nb)=b(nb)/U(n,n)
for i=n-1:-1:1,
   p=U(i,n:-1:i+1)*x(n:-1:i+1);
   x(i)=(b(i)-p)/U(i,i);
end
return
```

```
function [bern]=bernstein(k,n)
%
%   Costruzione e calcolo dei polinomi di BERNSTEIN
%   sull'intervallo [0,1].
%
%   Input:
%       k   indice del polinomio
%       n   grado del polinomio
%
%   Output:
%       bern    vettore contenente i valori del polinomio
%               sull'intervallo [0,1]
%
%-------------------------------------------------------
```

```
j=0;
if k==0
   p=1;
else
   p=prod([1:n])/(prod([1:k])*prod([1:n]));
end
for z=0:0.01:1;
   j=j+1;
bern(j)=p*z^k*(1-z)^(n-k);
end
return
%-------------------------------------------------------
```

```
%-------------------------------------------------------
%  Programma per la costruzione del grafico dei polinomi
%                 di Bernstein di grado n
%
%      B_0,0(z)=1
%      B_n,k(z)=(1-z)B_n-1,k(z)+zB_n-1,k-1(z)      k=0,...,n
%                                          0<= z <=1
%-------------------------------------------------------
n=3;
t=[0:0.01:1]
[m1,m2]=size(t);
for k=0:n
   bern=bernstein(k,n);
   for j=1:m2,
   x(k+1,j)=bern(j);
   end
end
plot(t,x(1,1:m2),'-',t,x(2,1:m2),'--',t,x(3,1:m2),'-.',t,x(4,1:m2),':')
legend('k=0','k=1','k=2','k=3')
title('polinomi di Bernstein per n=3')
```




```
%  file hilbert.m
%
%         genera la matrice di Hilbert di ordine n
%-------------------------------------------------------
n=input('Ordine della matrice   ');
for i=1:n;
          for j=1:n;
                    h(i,j)=1/(i+j-1)
          end
end
```

```
%  file esponen.m
%
%   calcola un'approssimazione di e^2 usando i primi n
%   termini della serie di Taylor
%-----------------------------------------------------------------
n=input('Numero dei termini della serie ');
s=0;
for k=0:n-1
          a=2^k/prod(1:k);
          s=s+a;
end
disp('il valore approssimato di e^2 è ');disp(s);
disp('errore assoluto = ');disp(exp(2)-s);
```

```
%-------------------------------------------------------------------------------
%  Il seguente sottoprogramma calcola le radici
%  di una equazione di secondo grado:
%        ax^2+bx+c=0     con a diverso da 0
%  utilizzando, per limitare la propagazione degli errori
%  di arrotondamento, le formule:
%
%        x1=-(b+sgn(b)sqrt(DELTA)/2a
%        x2=c/(ax1)
%
%  dove DELTA=b^2-4ac
%
%  INPUT:  a,b,c
%            ep tolleranza dipendente dalla precisione macchina
%  OUTPUT: x1, x2 radici
%        ind = 0 discriminante negativo
%            = 1 discriminante nullo
%            = 2 discriminante positivo
%-------------------------------------------------------------------
function [x1,x2,ind]=secondo(a,b,c,ep)
delta=b^2-4*a*c;
if abs(delta) <= ep
   ind=1;
   x1=-0.5*b/a;
   x2=x1;
elseif delta > ep
   ind=2;
   if b < 0
      s=-1;
   else
      s=1;
   end
   x1=-0.5*(b+s*sqrt(delta))/a;
   x2=c/(a*x1);
else
   ind=0;
   x1=-0.5*b/a;
   x2=+0.5*sqrt(-delta)/a;
end
return
%-------------------------------------------------------------------
```