

Corso di Laurea in Ingegneria Aerospaziale
Calcolo Numerico (con Laboratorio)
Laboratorio in MATLAB

a.a. 2007/2008

1	Laboratorio introduttivo al MATLAB	1
1.1	Introduzione	1
1.2	Primi comandi in MATLAB. Vettori e matrici.	2
1.2.1	Sul formato	8
1.2.2	Sul nome delle variabili	8
1.3	Cenni sulle funzioni in linea e sulla grafica	8
1.4	Functions	12
1.5	Cicli di controllo	14
1.5.1	Ciclo for	14
1.5.2	Ciclo while	14
1.5.3	Il ciclo if-else-end	15
1.5.4	La costruzione switch-case	16
1.5.5	Relazioni e Operatori logici	16
1.6	Gli scripts .m	17
1.7	Applicazioni al Calcolo Numerico	17
1.7.1	Sulla precisione di macchina	17
1.7.2	Sul fenomeno di cancellazione numerica	19
2	Laboratorio MATLAB su instabilità numerica e zeri di funzione	21
2.1	Sull'instabilità numerica	21
2.2	Sugli zeri di funzione	23
2.2.1	Esempio in cui Newton-Raphson fallisce	23
2.2.2	Altri esempi	24
3	Esercitazione 1. Sulla Soluzione di Equazioni non Lineari	25
4	Laboratorio MATLAB sull'interpolazione	28
4.1	L'interpolazione in MATLAB	28
4.2	Il metodo dei coefficienti indeterminati	29

4.3	Interpolazione di Lagrange	31
4.4	Formula di Newton con le differenze divise	32
4.5	Applicazioni	34
4.5.1	Esempio	34
4.5.2	Esempio di Runge	34
4.5.3	Effetti del malcondizionamento	34
5	Esercitazione 2. Su Interpolazione e Approssimazione di dati	37

Imparare ad usare il MATLAB e *non* metterlo da parte.... Un'introduzione al MATLAB e sue prime applicazioni nel Calcolo Numerico. Parte del materiale è preso e liberamente adattato da[3].

Sommario

1.1	Introduzione	1
1.2	Primi comandi in MATLAB. Vettori e matrici.	2
1.2.1	Sul formato	8
1.2.2	Sul nome delle variabili	8
1.3	Cenni sulle funzioni in linea e sulla grafica	8
1.4	Functions	12
1.5	Cicli di controllo	14
1.5.1	Ciclo for	14
1.5.2	Ciclo while	14
1.5.3	Il ciclo if-else-end	15
1.5.4	La costruzione switch-case	16
1.5.5	Relazioni e Operatori logici	16
1.6	Gli scripts .m	17
1.7	Applicazioni al Calcolo Numerico	17
1.7.1	Sulla precisione di macchina	17
1.7.2	Sul fenomeno di cancellazione numerica	19

1.1 Introduzione

- Il MATLAB è un ambiente interattivo che, tramite la finestra dei comandi (Command Window), permette l'esecuzione e la successiva visualizzazione

dei risultati di operazioni matematiche che coinvolgono variabili scalari e/o vettoriali.

- ↯ La grafica in ambiente MATLAB è molto buona e permette la creazione di grafici di dati, funzioni, risultati... in maniera molto efficace.
- ↯ Vi sono un numero notevole di funzioni proprie del MATLAB (*functions*) che permettono di risolvere problemi complicati mediante poche righe di comandi (per esempio, trovare le radici di una funzione, risolvere un sistema lineare, calcolare il valore di un integrale...).
- ↯ È possibile scrivere dei programmi MATLAB (propriamente degli *script .m*) grazie ai quali si possono scrivere algoritmi di calcolo che possono essere eseguiti ogni volta che si vuole.
- ↯ MATLAB sta per **MATrix LABORatory** ed è prodotto dalla MathWorks Inc. - per utilizzarlo bisogna comprarlo e avere una licenza.

Tuttavia esistono diverse piattaforme che possono essere scaricate sul proprio computer gratuitamente - essendo *open source* - e che hanno funzioni e modalità equivalenti al MATLAB. Tra queste, ricordiamo OCTAVE e SCILAB. SCILAB si trova in rete su www.scilab.org.

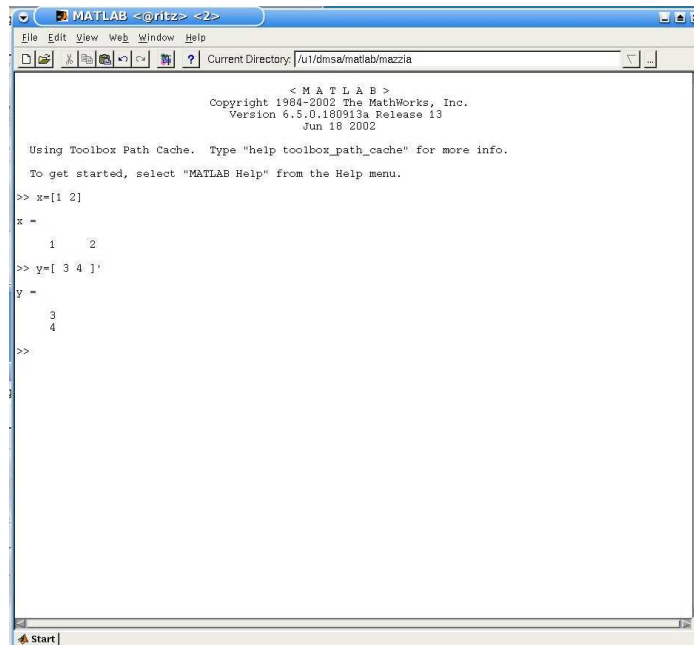
OCTAVE si trova su www.octave.org.

1.2 Primi comandi in MATLAB. Vettori e matrici.

In ambiente unix, si lancia il comando `matlab` da una finestra di shell, oppure (sia per l'ambiente unix che per windows) dal menu delle applicazioni si cerca l'icona del MATLAB e si avvia il MATLAB in questo modo.

A questo punto, sul monitor si aprirà la finestra dei comandi, simile a quella mostrata in Figura, con il simbolo `>>` seguito dalla barra `|` lampeggiante.

Ora possiamo scrivere i nostri primi comandi come il vettore `x=[1 2]` o `y=[3 4]'` che risultano, rispettivamente, vettore riga e vettore colonna (si veda la Figura e si provi direttamente in MATLAB).



```

MATLAB <@ritz> <2>
File Edit View Web Window Help
Current Directory: /u1/dmsa/matlab/mazzia

< M A T L A B >
Copyright 1984-2002 The MathWorks, Inc.
Version 6.5.0.180913a Release 13
Jun 18 2002

Using Toolbox Path Cache. Type "help toolbox_path_cache" for more info.
To get started, select "MATLAB Help" from the Help menu.

>> x=[1 2]
x =
     1     2
>> y=[3 4]
y =
     3
     4
>>

```

Si ha infatti

```
>>x=[1 2]
```

```
x =
```

```
     1     2
```

```
>>y=[3 4]'
```

```
y =
```

```
     3
```

```
     4
```

Quindi, per scrivere un vettore riga basta scrivere le sue componenti su un riga tra parentesi quadre $x = [1 \ 2]$, mentre per un vettore colonna basta mettere un apice $'$ al vettore stesso scritto come vettore riga $y = [3 \ 4]'$. Un'altra maniera per scrivere un vettore colonna si ha con l'ausilio del punto e virgola $;$, che permette di creare più righe e, quindi, di creare un vettore colonna: $y = [3; 4]$.

A questo punto, nella Command Window del MATLAB abbiamo due variabili, che abbiamo chiamato x e y .

Per vedere quali sono le variabili presenti, si usa il comando `who`. Per vedere quali e come sono fatte le variabili presenti, si usa il comando `whos`.

```
>> who
```

```
Your variables are:
```

```
x y
```

```
>> whos
```

Name	Size	Bytes	Class
x	1x2	16	double array
y	2x1	16	double array

```
Grand total is 4 elements using 32 bytes
```

Con queste variabili x e y , possiamo fare delle operazioni matematiche. Per esempio:

```
>>x*y
```

```
ans =
```

```
11
```

```
>>x+y'
```

```
ans =
```

```
4 6
```

```
>>x'+y
```

```
ans =
```

```
4
```

```
6
```

Facendo $x*y$ si esegue il prodotto scalare tra i due vettori x e y , prendendo il primo come vettore riga e il secondo come vettore colonna, vale a dire il prodotto $x_1y_1 + x_2y_2$.

Con $x+y'$ si esegue la somma di due vettori riga (notiamo che abbiamo scritto y' , quindi abbiamo fatto la trasposizione del vettore colonna).

Con $x'+y$ eseguiamo la somma di due vettori colonna (con x' si fa la trasposta del vettore x ottenendo un vettore colonna).

Osserviamo che il risultato di queste operazioni prende il nome di `ans`: viene creata, cioè, una variabile “ausiliara” che serve a memorizzare il risultato dell’operazione fatta. Prima `ans` vale il risultato del prodotto $x*y$, poi vale il risultato di $x+y'$ e infine vale $x'+y$. Quindi ogni volta `ans` cambia dimensione e significato.

Con le stesse modalità seguite per creare i vettori x e y , in MATLAB si possono creare delle matrici. Si usa il punto e virgola per separare le righe di una matrice mentre gli elementi di una stessa riga sono scritti uno di seguito all’altro lasciando un spazio bianco tra essi.

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

```
A =
```

```
     1     2     3
    -4    -5    -6
     7     8     9
```

```
>> B=A'
```

```
B =
```

```
     1    -4     7
     2    -5     8
     3    -6     9
```

```
>> C=A+B
```

```
C =
```

```
     2     -2    10
    -2    -10     2
    10     2    18
```

```
>> C=A-B
```

```
C =
```

```
     0     6    -4
    -6     0   -14
     4    14     0
```

```
>> C=A*B
```

```
C =
```

```
    14   -32    50
```

```
-32    77   -122
   50  -122   194
```

```
>> C=A.*B
```

```
C =
```

```
    1    -8    21
   -8    25   -48
    21   -48    81
```

Dall'esempio fatto possiamo notare che:

- ↘ La trasposta di una matrice si fa mediante il comando `'`. A' è la trasposta della matrice A .
- ↘ La somma e la sottrazione di matrici si fa mediante il `+` e il `-` ($A+B$ e $A-B$).
- ↘ Il prodotto tra matrici viene fatta mediante `*`.
- ↘ Il comando `.*` non esegue la moltiplicazione tra matrici come nella definizione classica, ma esegue il prodotto dei corrispondenti elementi di due matrici (o di vettori).

A questo punto, possiamo pensare di usare il MATLAB non come una calcolatrice - ovvero, non solo per fare tutti quei calcoli che possono essere fatti tramite una calcolatrice - ma di usare altre potenzialità come quella di risolvere un sistema lineare.

Con il comando `clear` cancelliamo tutte le variabili dalla Command Window e scriviamo i seguenti comandi:

```
>> A=[10 -7 0; -3 2 6; 5 -1 5]
```

```
A =
```

```
    10    -7     0
    -3     2     6
     5    -1     5
```

```
>> b=[7; 4; 6]
```

```
b =
```

```
    7
    4
```

```
6
>> x=A\b
```

```
x =
```

```
0
-1
1
```

Dopo aver creato la matrice A e il vettore b, risolviamo il sistema $Ax=b$, cercando il vettore soluzione x, mediante il comando `A\b`: l'istruzione `\` - *backslash* permette di fare la *divisione a sinistra* di una matrice A per un vettore b o per una matrice B, dando come risultato $A^{-1}b$ o $A^{-1}B$, mediante il procedimento di eliminazione di Gauss.

Si faccia `help slash` per vedere tutte le possibili operazioni che possono essere eseguite con `\` e con `/`.

Osserviamo che il comando `help` fornisce la lista di tutti gli argomenti di cui si trova l'help in linea nella Command Window del MATLAB.

Con il comando `help argomento_su_cui_vogliamo_help` il MATLAB ci fornisce informazioni su quell'argomento - se provvisto di help.

Per esempio:

```
>> help sin
```

```
SIN    Sine.
```

```
SIN(X) is the sine of the elements of X.
```

```
>> help log
```

```
LOG    Natural logarithm.
```

```
LOG(X) is the natural logarithm of the elements of X.
```

```
Complex results are produced if X is not positive.
```

```
See also LOG2, LOG10, EXP, LOGM.
```

```
>> help log10
```

```
LOG10  Common (base 10) logarithm.
```

```
LOG10(X) is the base 10 logarithm of the elements of X.
```

```
Complex results are produced if X is not positive.
```

```
See also LOG, LOG2, EXP, LOGM.
```

Tornando alle matrici, con il MATLAB si può anche calcolare l'inversa di una matrice, mediante il comando (function del MATLAB) `inv`

```
inv(A)

ans =

    -1.0323e-01    -2.2581e-01     2.7097e-01
    -2.9032e-01    -3.2258e-01     3.8710e-01
     4.5161e-02     1.6129e-01     6.4516e-03

inv(A)*b

ans =

         0
    -1.0000e+00
     1.0000e-00
```

Notiamo che non abbiamo posto l'inversa di A uguale ad una matrice, perciò si ha `ans` come inversa di A. Facendo `inv(A)*b` risolviamo il sistema e otteniamo la stessa soluzione avuta prima, anche se scritta in formato esponenziale.

1.2.1 Sul formato

A proposito del formato, di *default* il MATLAB esprime i risultati in `format short` - formato a punto fisso scalato con 5 cifre decimali. Per cambiare il formato, basta porre l'istruzione `format long` se si vuole il formato a punto fisso scalato ma con 15 cifre decimali o `format long e` se si vuole il formato floating point a 15 cifre decimali.

Si faccia `help format` per vedere tutti i diversi formati. Si cambi tipo di formato e si visualizzino nuovamente le variabili presenti nella Command Window per vedere come sono ora rappresentate le variabili.

1.2.2 Sul nome delle variabili

Un'altra osservazione importante in MATLAB è che le lettere minuscole e maiuscole non sono uguali in MATLAB: una variabile chiamata A è diversa da a, una variabile `XOld` è diversa da `xOld`. Quindi bisogna prestare molta attenzione a come sono definite le variabili per non incorrere in errori.

1.3 Cenni sulle funzioni in linea e sulla grafica

In MATLAB esistono moltissime funzioni intrinseche - ad esempio `sin`, `cos`, `log`, `log10`, `exp`, `abs`, `asin`, `acos`, `tan`, `atan`,

Si scriva `>> help matlab/elfun` per conoscere tutto l'elenco delle funzioni matematiche elementari presenti in MATLAB.

Ma possiamo anche noi creare delle funzioni.

La via più semplice è data dalla creazione di funzioni cosiddette in linea, mediante la function del MATLAB `inline` che permette facilmente di convertire una stringa di caratteri in una funzione.

Vediamo un esempio, lavorando sulla Command Window.

```
>> f=inline('x^3 - 0.001*x - log(x+1)*sin(x)')
```

```
f =
```

```
Inline function:
```

```
f(x) = x^3 - 0.001*x - log(x+1)*sin(x)
```

Con il comando `inline(' ')` scrivendo tra gli apici l'espressione della nostra funzione dipendente, in questo caso, dalla sola variabile x , abbiamo definito una funzione in linea a cui abbiamo dato il nome `f`.

Con questa funzione possiamo fare tutti i calcoli che desideriamo. Ad esempio, possiamo valutare questa funzione per determinati valori di x :

```
>> f(0)
```

```
ans =
```

```
0
```

```
>> f(10)
```

```
ans =
```

```
1.0013e+03
```

```
>> f(-1)
```

```
Warning: Log of zero.
```

```
> In /usr/local/matlab13/toolbox/matlab/funfun/inlineeval.m at line 13
```

```
In /usr/local/matlab13/toolbox/matlab/funfun/@inline/subsref.m at line  
25
```

```
ans =
```

```
-Inf
```

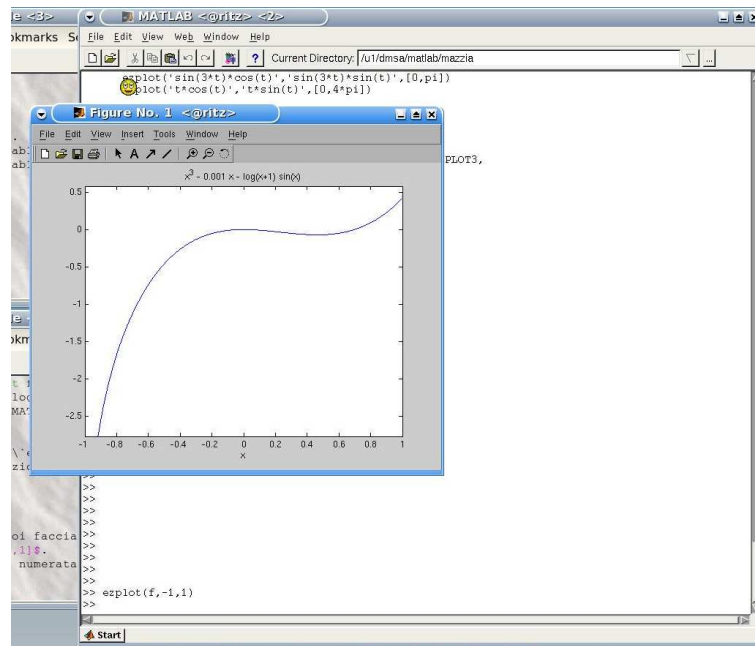
Osserviamo il messaggio di errore che è comparso quando abbiamo voluto calcolare

$f(-1)$, in cui la funzione non è definita (vedasi la funzione logaritmo presente nella espressione della f) e a cui il MATLAB dà il valore $-\text{Inf}$.

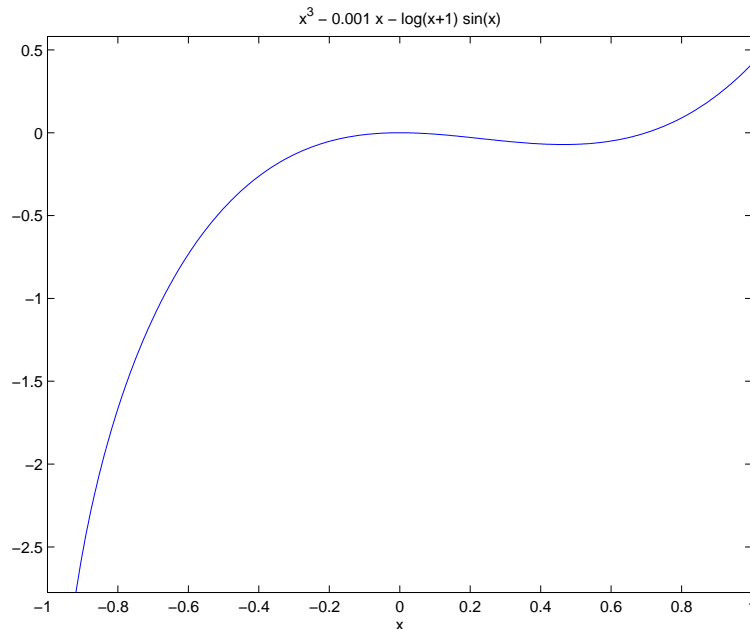
Un'utile istruzione è relativa alla possibilità di creare il grafico della funzione in un intervallo scelto dall'utente.

```
>> ezplot(f,-1,1)
```

Con questo comando noi facciamo il grafico della f nell'intervallo $[-1, 1]$.



Il grafico può essere salvato in vari formati (per esempio, in EPS o JPEG o PDF): basta andare, sulla finestra della figura, su **File**, quindi su **Export** e, di qui, scegliere il tipo del formato e il nome del file.



Mediante `help inline` e `help ezplot`, si approfondisca tutto ciò che può essere fatto mediante queste functions del MATLAB.

Se si vuole fare il grafico di una serie di dati - due vettori x e y di lunghezza n , per esempio $y_i = f(x_i)$ per una certa f - in MATLAB si può scrivere: `>> plot(x,y)`.

Si può poi giocare sul tipo di linea (solida, tratteggiata, a punti) oppure, se si vuole un grafico per punti, sul tipo di punto (cerchio, asterisco, etc), sui colori...

Si faccia `>> help plot` per conoscere tutte le potenzialità del `plot`.

Esempio.

Si provino le seguenti istruzioni e si veda cosa succede

```
>> x= [ 0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0];
>> y= exp(x);
>> plot(x,y)
>> plot(x,y,'o')
>> plot(x,y,'-.g')
>> plot(x,y,'-.g', x,y, '*r')
```

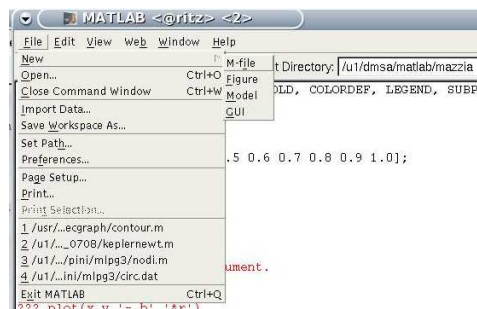
- ↘ Osserviamo che, il punto e virgola alla fine dell'istruzione per x e y **non** visualizza il contenuto dei vettori sulla Command Window.
- ↘ Il primo grafico viene cancellato dal secondo. Allo stesso modo il secondo grafico viene sostituito dal terzo, e così via. Se si vogliono sovrascrivere più grafici si deve scrivere il comando `>> hold on` dopo che è stato creato il primo grafico (si faccia `>> help hold` per capire meglio questa istruzione).

1.4 Functions

Una function, in MATLAB, può dipendere da una o più variabili di input e dare una o più variabili di output (corrisponde alle functions o alle subroutines del FORTRAN, ad esempio, a seconda che abbia un solo o più dati di output, rispettivamente).

L'importante è che queste functions vengano scritte nella stessa directory in cui si lavora nella Command Window o, alternativamente, che dalla Command Window ci si sposti nella directory in cui si trova la function con cui si vuole lavorare. Per vedere in quale directory ci si trova, basta osservare ciò che c'è scritto in alto a destra nella Command Window, là dove compare la scritta Current Directory.

La function che vogliamo scrivere noi, deve essere scritta in un file nella stessa directory e dovrà essere salvata con il nome che vogliamo dare alla function stessa e con l'estensione .m. Per scrivere una function si può anche utilizzare la finestra del MATLAB stessa preposta a tale scopo. Infatti, nella barra in alto a destra della finestra del MATLAB, da File si passa a New e di qui a M-file.



Per esempio, la funzione in linea vista precedentemente può essere scritta come una function. Scriviamo le seguenti righe in un file che chiameremo `funprova.m`:

```
function y= funprova(x)
% function y=funprova(x)
% funzione di prova per vedere come si scrivono
% functions in MATLAB
y= x.^3 - 0.001*x - log(x+1).*sin(x);
```

Adesso, dalla Command Window possiamo lavorare con questa function come abbiamo fatto in precedenza.

- ✎ Ciò che segue dopo `%` è una riga di commento.
- ✎ Le righe di commento scritte subito dopo la riga che definisce la function compaiono anche nella Command Window con l'`help`. Quindi, se noi scriviamo `>> help funprova` leggeremo quelle righe di commento sulla Command Window (fare una verifica).

- Nel definire la y abbiamo scritto dei \wedge e $.*$ per il caso in cui questa funzione voglia essere calcolata non solo su una variabile scalare, ma anche sulle componenti di un intero vettore - in tal modo viene creato un vettore y tale che $y(i) = f(x(i))$ e, quindi, le operazioni sono da fare componente per componente del vettore e non in forma vettoriale. Se mettessimo semplicemente \wedge e $*$ la function potrebbe essere usata solo per variabili scalari (e non si potrebbe usare `ezplot` perchè si avrebbero degli errori).

Una function può avere più parametri di output. Facciamo un'esempio semplicissimo con la function `funprova2.m` in cui modifichiamo l'output della precedente:

```
function [y,i]= funprova2(x)
% function [y,i]=funprova2(x)
% funzione di prova con due variabili di output
% y = valore della funzione f(x)= x^3 -0.001x -log(x+1)*sin(x)
% i = fornisce il segno della y (se positivo da' 1 altrimenti -1)
y= x.^3 - 0.001*x - log(x+1).*sin(x);
if y>= 0
    i=1;
else
    i=-1
end
```

- Adesso nella Command Window dobbiamo scrivere `>> [y,i] = funprova2(1)` se vogliamo il giusto output della `funprova2` per $x=1$:

```
>> [y,i]=funprova2(1)
```

```
y =
```

```
    4.1574e-01
```

```
i =
```

```
    1
```

- Un'altra osservazione è che in questa function abbiamo usato il ciclo `if`, esattamente come in un linguaggio di programmazione.

1.5 Cicli di controllo

Ricordiamo le varie strutture che possono essere utilizzate (e per ciascuna di esse basta fare un `help` per approfondire il loro utilizzo in MATLAB):

- ↘ il ciclo `for`
- ↘ il ciclo `while`
- ↘ la costruzione `if-else-end`
- ↘ la costruzione `switch-case`

1.5.1 Ciclo `for`

Lo schema è

```
for k = array
    comandi
end
```

I comandi che si trovano tra `for` e `end` sono eseguiti per tutti i valori di `k` che sono nell'array.

Supponiamo che vogliamo creare un vettore che valuta la funzione seno su undici punti equispaziati da 0 a π . Osserviamo che in MATLAB `pi` indica π . Possiamo scrivere:

```
for n=0:10
    x(n+1)= sin(pi*n/10);
end
```

Possiamo usare più cicli `for` l'uno concatenato all'altro.

1.5.2 Ciclo `while`

```
while espressione logica
    comandi
end
```

Questo ciclo è usato quando i comandi devono essere ripetuti fino a quando rimane vera una certa espressione logica.

Facciamo un esempio: supponiamo di voler dividere π per 2 e di continuare a dividere il risultato del quoziente fino a che non diventa minore o uguale a 0.01. Qual è l'ultimo quoziente più grande di 0.01?

```
>> q=pi;
>> while q > 0.01
    q=q/2;
end
>> q*2
```

1.5.3 Il ciclo if-else-end

```
if espressione logica
    comandi
end
```

```
if espressione logica
    comandi % (eseguiti se l'espressione
logica è vera)
else
    comandi % (eseguiti se l'espressione
logica è falsa)
end
```

```
if espressione logica 1
    comandi % (eseguiti se espressione
logica 1 è vera)
elseif espressione logica 2
    comandi % (eseguiti se espressione
logica 2 è vera)
elseif ...
:
else
    comandi % (eseguiti se tutte le
espressione logiche precedenti sono false)
end
```

1.5.4 La costruzione switch-case

```
switch espressione (scalare o stringa)
    case valore1 % (eseguita se
l'espressione è valutata al valore1)
        comandi
    case valore2 % (eseguita se
l'espressione è valutata al valore2)
        comandi
    :
        otherwise
        comandi
end
```

Il ciclo con `switch` confronta i valori dati nell'espressione di input (subito dopo `switch`) con ciascun valore assegnato a `case` ed esegue le istruzioni relative al `case` in cui valore ed espressione coincidono.

Facciamo l'esempio sulla scelta di un caso test.

```
scelta='test1';
switch scelta
case {'test1'}
a= -1;
b= 1;
case {'test2'}
a= 0;
b= 4;
otherwise
disp('nessun caso test scelto')
end
```

- ✎ Osserviamo che abbiamo introdotto la variabile `scelta` che è data da una stringa di caratteri (il nome del caso test).
- ✎ Dopo `case` abbiamo usato le parentesi graffe.
- ✎ Per visualizzare un messaggio sulla Command Window, abbiamo usato l'istruzione `disp`.

1.5.5 Relazioni e Operatori logici

In MATLAB si ha:

Operatore	Descrizione
<	strettamente minore
<=	minore o uguale
>	strettamente maggiore
>=	maggiore o uguale
==	uguale
~=	non uguale

Operatore logico	Descrizione
&	e
	o
~	non

1.6 Gli scripts .m

Oltre alle `functions` in MATLAB è possibile scrivere dei veri e propri programmi in cui poter dare dei dati in input e ottenere dei risultati, scrivendo tutto l'algoritmo in un file - che sarà il nostro programma di calcolo. Questo file sarà salvato con lo stesso nome con il quale sarà eseguito nella Command Window, ma con l'estensione `.m`. Se, ad esempio, scriviamo un programma dal nome `prova.m`, potremo eseguire questo programma dalla Command Window (dopo esserci assicurati che il programma si trova nella stessa directory su cui stiamo lavorando in MATLAB), scrivendo il comando `>> prova`.

1.7 Applicazioni al Calcolo Numerico

1.7.1 Sulla precisione di macchina

Sappiamo che i numeri reali a infinite cifre decimali non possono essere rappresentate esattamente da un calcolatore. La precisione di macchina `eps` è il più piccolo numero positivo di macchina tale che $float(1.0 + eps) > 1.0$.

In MATLAB esiste già una variabile chiamata `eps` che dà la precisione di macchina del MATLAB.

```
>> format long
>> eps
```

```
ans =
```

```
2.220446049250313e-16
```

```
>> help eps
```

EPS Floating point relative accuracy.
 EPS returns the distance from 1.0 to the next largest floating point number. EPS is used as a default tolerance by PINV and RANK, as well as several other MATLAB functions.

See also REALMAX, REALMIN.

Vediamo ora due vie per calcolare *eps*.

Il primo è dato dalle seguenti programma:

```
format long
a=4/3
b=a-1
c=3*b
e=1-c
```

Se i risultati fossero in aritmetica esatta, il valore di *e* dovrebbe essere 0. In realtà *e* è uguale alla precisione di macchina *eps*, a causa degli errori di arrotondamento.

Un secondo modo è di utilizzare la definizione che abbiamo data a *eps*, possiamo scrivere il seguente programma, che chiamiamo *calcoloeps.m*:

```
% calcolo di epsilon = precisione di macchina
nmax=100;
n=0;
epsilon=0.5;
eps1=epsilon+1;
while (eps1 > 1 & n < nmax)
    n=n+1;
    epsilon=0.5*epsilon;
    eps1=epsilon+1;
end
epsilon=2*epsilon; % questo e' il valore piu' piccolo che ha
                  % verificato la disequaglianza
fprintf(1, 'epsilon= %16.15e n= %6d\t',epsilon, n )
epn=2^(-n); % epsilon pu\o essere calcolato anche prendendo il valore
            % di n
fprintf(1, '\n epsilon di macchina = %16.15e\t',epn )
```

Osserviamo che la stampa dei risultati è stata fatta mediante il comando `fprintf` - molto simile a quanto viene fatto in linguaggio C.

Con `fprintf(1 ...` la stampa avviene sulla Command Window. Se all'inizio dello script si apre un file di dati mediante l'istruzione

```
fid=fopen('filedati.txt', 'w')
```

allora su quel file possiamo scrivere i risultati con `fprintf` associato a `fid`.


```
function s = powersin(x)
% powersin: calcolo di sin(x) mediante lo sviluppo in serie
%di potenze
% s = powersin(x) cerca di calcolare sin(x) dalla sua serie
% di potenze
% sin(x) = x - x^3/3! + x^5/5! - x^7/7! + ....
%
% s = valore approssimato per sin(x)
%
s = 0;
t = x;
n = 1;
while s+t ~= s;
    s = s + t;
    t = -x.^2/((n+1)*(n+2)).*t;
    n = n + 2;
end
```

- Quando si esce fuori dal ciclo `while`?
- Quali valori si ottengono per $x = \pi/2$, $11\pi/2$, e $x = 31\pi/2$? Confrontare i risultati con il valore esatto.
- Nella `function` dare come dati di output anche `n` e `t` oltre a `s`. Quanti termini sono richiesti per dare il risultato finale?
- Qual è il termine più grande della serie?
- Cosa si può concludere?