Argomenti

(1): Introduzione a Matlab

(2): *Scripts* – Funzioni – Operatori

(3): *Input-Output* dei dati

(4): Grafici e visualizzazione dei dati

(5): Analisi dei dati

(6): Analisi nonlineare ed approssimazioni di funzioni e di dati

(7): Differenziazione ed integrazione numerica – calcolo simbolico

(8): Sistemi lineari

(9): Soluzioni *ODEs*

(10): Cenni a soluzioni *PDEs*

(2): Scripts – *Functions* - Operatori

- *Scripts*
- *Functions*
- Istruzioni di Controllo
  - Cicli
  - Scelte Condizionate
- Operatori di Relazione
- Operatori Logici
- Esempi

Scripts

Uno ***script***, insieme di procedure in Matlab, raggruppa una serie di comandi senza eseguirli

Le procedure, o comandi, sono contenute in un file avente estensione '*.m' (M-file). Gli M-files equivalgono a programmi, funzioni, *subroutines* e procedure degli altri linguaggi di programmazione

Gli M-files, eseguibili al *prompt* dei comandi, si possono scrivere e salvare con l'apposito *editor* (o con altri *editor* di testo).
Sono eseguibili anche dall' *editor* (Debug → Run, oppure f5)

Gli M-files possono chiamare altri M-files

Per accedere al testo di un M-file: edit *nomefile*

Per eseguire un M-file: >> *nomefile*

Attenzione al *path* dell'M-file

Scripts

Quando si usano gli ***scripts***?

- Per automatizzare ripetizioni si sequenze di operazioni
- Per realizzare progetti complessi

Gli *scripts* interagiscono con le variabili del *workspace*

Le variabili prodotte durante l'esecuzione di uno *script* vengono scritte nel *workspace*

Scripts

Esempio1

```
% SPIRALE    Questo script traccia il grafico orario tridimensionale della
% legge  x=R*cos(2*pi*nu*t); y=R*sin(2*pi*nu*t);  z=b*t

nu=10;
R=5;
b=1;
ti=0;
tf=5;
N=10000;
t=linspace(ti,tf,N);
x=R*cos(2*pi*nu*t);
y=R*sin(2*pi*nu*t);
z=b*t;
plot3(x,y,z)
```

Scripts

Esempio2

```
%ROULDIST      Empirical distribution of number
%of real eigenvalues.

k = 100;
wheel = zeros(k,1);
for i = 1:k
    A = randn(8);
    % Count number of eigenvalues with imag.
part < tolerance.
    wheel(i) = sum(abs(imag(eig(A)))<.0001);
end
hist(wheel,[0 2 4 6 8]);
```

Functions

Una *function* in Matlab è come un *black-box*: riceve dati (variabili di ingresso o ***input***) e produce dei risultati come variabili di uscita o ***output***)

Le funzioni hanno la seguenti strutture:

M-files

anonymous

inline

Functions: M-files

1) La prima linea di una *function* è detta ***function declaration line*** ed ha la struttura:

function [output1, output2,..] = **nomefunction**(input1, input2, …)

2) Il nome del M-file ed il nome della *function* devono essere identici
3) I nomi delle *functions* seguono le stesse regole delle variabili
4) Subito dopo la prima linea, ci sono linee di commento: importanti (per il comando *help* e *lookfor*)
5) Sotto i commenti c'è il corpo della *function*
6) Una M-file *function* può contenere chiamate a *script file*, che vengono valutati nello spazio di lavoro della funzione

Functions

Esempio1:

```
function [t,x,y,z]=spirale_fun(nu,N)
% SPIRALE_FUN Questa function traccia il grafico orario tridimensionale della legge
% x=R*cos(2*pi*nu*t);  y=R*sin(2*pi*nu*t); z=b*t
% Accetta come dati in ingresso la frequenza nu e il numero di punti utilizzati N

R=5;
b=1;
ti=0;
tf=5;
t=linspace(ti,tf,N);
x=R*cos(2*pi*nu*t);
y=R*sin(2*pi*nu*t);
z=b*t;
plot3(x,y,z)
```

Functions

Esempio2:

```
function y = maxentry(A)
%MAXENTRY   Largest absolute value of matrix
%entries.
%MAXENTRY(A) is the maximum of the absolute
%values of the entries of A.

y = max(max(abs(A)));
```

Functions

Esempio3:

```
function [f,fprime] = flogist(x,a)
%FLOGIST   Logistic function and its derivative
%[F,FPRIME] = FLOGIST(X,A) evaluates the
%logistic function F(X) = X.*(1-A*X) and its
%derivative FPRIME at the matrix argument X,
%where A is a scalar parameter.

f = x.*(1-a*x);
fprime = 1-2*a*x;
```

Functions: M-files

7) In un singolo *M-file function* si possono avere piu' *functions*
8) Le *subfunctions* possono essere chiamate solo all'interno del M-file
   E' bene chiamare le *subfunctions* col prefisso *local*
9) Le *functions* possono avere un numero variabile di argomenti di input e di output, gestite da *nargin* e *nargout*:

```
function [f1,f2] = square(x1,x2)
    If nargin==1
      f1=x1^2;
      f2=NaN;
    elseif nargin==2
      f1=x1^2;
      f2=x2^2;
    end
```

10) Le istruzioni *varargin* e *varargout* permettono di costruire *functions* con argomenti opzionali

Functions

Ci sono vari modi per definire, valutare e rappresentare una *funzione matematica* in Matlab:

1a) fun= ('x./(1+3*x.^2)')

1b) y=eval('fun')

1c) fplot(fun,[xmin xmax])

2a) fun= inline('x./(1+3*x.^2)','x')

2b) y=feval(fun, 'x')

2c) fplot(fun,[xmin xmax])

Functions:

3a) fun= @(x)[x./(1+3*x.^2)]

3b) y=fun('x')

3c) fplot(fun,[xmin xmax])

4a) function y= fun(x)

    y= x./(1+3*x.^2);

    end

4b) y=fun(x) oppure y=feval(@fun,x)

4c) fplot(@fun,[xmin xmax])

Istruzioni di controllo

L'esecuzione di un programma (che in MatLab ha la forma di uno *script* o di una *function*), avviene con una sequenza di istruzioni eseguite una dopo l'altra in maniera sequenziale, a partire dalla prima riga.

Lo scorrimento della sequenza dei programmi può essere controllata da particolari strutture poste all'interno del programma.

Le due principali strutture sono:

Ripetizioni di un blocco di istruzioni (iterazione): *for, while*

Scelta condizionata tra sequenze alternative di istruzioni: *if-else, switch*

Ciclo *for*

Struttura del ciclo *for*:

for x=vettore

Sequenza di istruzioni

end

Sequenza di istruzioni -> corpo del ciclo

Esempio:

```
X = [-2   -1    0    1    2]
i=1

for b=x

espo(i)=exp(b)

i=i+1

end
```

```
espo =
    0.1353
i =
     2
espo =
    0.1353    0.3679
i =
     3
espo =
    0.1353    0.3679    1.0000
i =
     4
espo =
    0.1353    0.3679    1.0000    2.7183
i =
     5
espo =
    0.1353    0.3679    1.0000    2.7183    7.3891
i =
     6
```

Ciclo *for* indicizzato

Struttura del ciclo *for*:

```
for  k=1:n
Sequenza di istruzioni
end
```

n -> contatore

Piu' in generale:

k=inizio:incremento:fine

```
>> n=5
n =
     5
>> prod=1
for k=1:n
    prod=prod*k
end
nfattoriale=prod

prod =
1
prod =
     1
prod =
     2
prod =
     6
prod =
    24
prod =
   120
nfattoriale =
   120
```

Cicli *for* nidificati

Struttura di cicli *for* nidificati:

```
for  I=1:M
        for  k=1:N
Sequenza di istruzioni
        end
end
```

Il ciclo più interno viene eseguito ciclicamente M volte per effetto del ciclo più esterno.

Questa struttura viene chiamata nidificazione.

Ciclo *for*

```
% Esempio1
% Questo script calcola, col rapporto incrementale, la
% velocità e l'accelerazione approssimate, nel moto
% circolare uniforme: x=R*cos(2*pi*nu*t)
% y=R*sin(2*pi*nu*t)
N=10000;
n=4;      % numero di giri
nu=1;     % frequenza del moto circolare uniforme
R=5;      % raggio del moto
ti=0;     % tempo iniziale
tf=n/nu; % tempo finale
Dt=(tf-ti)/(N-1);
t=ti:Dt:tf;
t=linspace(ti,tf,N);
x=R*cos(2*pi*nu*t);
y=R*sin(2*pi*nu*t);
vx=zeros(1,N-1);
vy=zeros(1,N-1);

for k=1:N-1
   vx(k)=(x(k+1)-x(k))/(t(k+1)-t(k));
   vy(k)=(y(k+1)-y(k))/(t(k+1)-t(k));
end
```

```
for k=1:N-2
   ax(k)=(vx(k+1)-vx(k))/(t(k+1)-t(k));
   ay(k)=(vy(k+1)-vy(k))/(t(k+1)-t(k));
end

subplot(3,2,1)
plot(t,x)
subplot(3,2,2)
plot(t,y)
subplot(3,2,3)
plot(t(1:N-1),vx)
subplot(3,2,4)
plot(t(1:N-1),vy)
subplot(3,2,5)
plot(t(1:N-2),ax)
 subplot(3,2,6)
plot(t(1:N-2),ay)
```

Ciclo *for*

Esempio1

Possibili alternative al seguente ciclo for:

```
for k=1:N-1
vx(k)=(x(k+1)-x(k))/(t(k+1)-t(k));
vy(k)=(y(k+1)-y(k))/(t(k+1)-t(k));
end
```

1):
```
vx=(x(2:N)-x(1:N-1))./(t(2:N)-t(1:N-1));
vy(k)=(y(2:N)-y(1:N-1))./(t(2:N)-t(1:N-1));
```

Oppure

2) usando la funzione *diff*: (diff(x)->($x_2 - x_1$, $x_3 - x_2$,..., $x_N - x_{N-1}$)

```
vx=diff(x)./diff(t);
vy=diff(y)./diff(t);
```

Ciclo *for*

Esempio2 (vedi più avanti):

Soluzione numeriche di equazioni differenziali ordinarie col metodo di Eulero:

$$dy/dt = f(x,y)$$

$$y(0)=y0$$

Nel caso di moto di caduta di un proiettile soggetto a forza peso e resistenza dell'aria, si ha:

$$dx^2/dt^2 = g - (gamma/m)\ v^2$$

$$x(1) = 0,\ v(0) = 0$$

Ciclo *for*

Esempio3:

Stima del tempo necessario per eseguire la moltiplicazione di due numeri *floating-point*:

Computer performance

Name	FLOPS
yottaFLOPS	10^{24}
zettaFLOPS	10^{21}
exaFLOPS	10^{18}
petaFLOPS	10^{15}
teraFLOPS	10^{12}
gigaFLOPS	10^{9}
megaFLOPS	10^{6}
kiloFLOPS	10^{3}

In Matlab sono disponibili le funzioni:

tic, toc

tic fa partire un orologio

toc restituisce il tempo trascorso in secondi

Per avere il numero di operazioni n al secondo:

Flops=n/toc

http://www.top500.org/

List	Rank	System	Vendors	Cores	Rmax (GFlop/s)	Rpeak (GFlop/s)
06/2012	1		IBM	1572864	16324751	20132659.2
11/2011	17		IBM	65536	690197	838861
11/2011	22		IBM	212992	478200	596378
06/2011	14		IBM	212992	478200	596378
11/2010	12		IBM	212992	478200	596378
06/2010	8		IBM	212992	478200	596378
11/2009	7		IBM	212992	478200	596378
06/2009	5		IBM	212992	478200	596378
11/2008	4		IBM	212992	478200	596378
06/2008	2		IBM	212992	478200	596378
11/2007	1		IBM	212992	478200	596378

http://www.top500.org/

43rd List: The TOP10

#	Site	Manufacturer	Computer	Country	Cores	Rmax [Pflops]	Power [MW]
1	National University of Defense Technology	NUDT	Tianhe-2 NUDT TH-IVB-FEP, Xeon 12C 2.2GHz, IntelXeon Phi	China	3,120,000	33.9	17.8
2	Oak Ridge National Laboratory	Cray	Titan Cray XK7, Opteron 16C 2.2GHz, Gemini, NVIDIA K20x	USA	560,640	17.6	8.21
3	Lawrence Livermore National Laboratory	IBM	Sequoia BlueGene/Q, Power BQC 16C 1.6GHz, Custom	USA	1,572,864	17.2	7.89
4	RIKEN Advanced Institute for Computational Science	Fujitsu	K Computer SPARC64 VIIIfx 2.0GHz, Tofu Interconnect	Japan	795,024	10.5	12.7
5	Argonne National Laboratory	IBM	Mira BlueGene/Q, Power BQC 16C 1.6GHz, Custom	USA	786,432	8.59	3.95
6	Swiss National Supercomputing Centre (CSCS)	Cray	Piz Daint Cray XC30, Xeon E5 8C 2.6GHz, Aries, NVIDIA K20x	Switzer-land	115,984	6.27	2.33
7	Texas Advanced Computing Center/UT	Dell	Stampede PowerEdge C8220, Xeon E5 8C 2.7GHz, Intel Xeon Phi	USA	462,462	5.17	4.51
8	Forschungszentrum Juelich (FZJ)	IBM	JuQUEEN BlueGene/Q, Power BQC 16C 1.6GHz, Custom	Germany	458,752	5.01	2.30
9	Lawrence Livermore National Laboratory	IBM	Vulcan BlueGene/Q, Power BQC 16C 1.6GHz, Custom	USA	393,216	4.29	1.97
10	Government	Cray	Cray XC30, Xeon E5 12C 2.7GHz, Aries	USA	225,984	3.14	

TOP500 SUPERCOMPUTER SITES

Ciclo while

Struttura del ciclo *while*:

***while* espressione logica**
sequenza di istruzioni
end

La sequenza di istruzioni viene eseguita ripetutamente fintantochè la espressione logica è TRUE (1).

Esempio:

```
function numwhile()
n=1;
tic;
while toc<1
    n=n+1;
end
n
```

Scelta condizionata: Struttura if-else-end

```
if espressione logica
sequenza di istruzioni
end
```

L'espressione logica può assumere due valori (logici):

Vero(**TRUE**) Falso(**FALSE**)

Se il valore dell'espressione è TRUE, la sequenza viene eseguita fino all' end, altrimenti viene ignorata.

Caso più generale:

```
if espressione logica
sequenza1 di istruzioni
else
sequenza1 di istruzioni
end
```

Se l'espressione logica è TRUE, viene eseguita la sequenza1, altrimenti viene eseguita la sequenza2

Scelta condizionata: Struttura *switch*

Per la scelta tra più opzioni si può far uso di valori di espressioni piuttosto che di valori logici, tramite la struttura *switch*:

switch espressione scalare o stringa
case value 1
sequenza di istruzioni
case value 2
sequenza di istruzioni
case value 3
sequenza di istruzioni
…….
otherwise
sequenza di istruzioni
end

Scelta condizionata: Struttura *switch*

Esempio:

```
function prova_switch()
a=round(6*rand+0.5)
switch a
          case 1
         disp('esce 1')
          case 2
         disp('esce 2')
          case 3
         disp('esce 3')
          case 4
         disp('esce 4')
          case 5
         disp('esce 5')
          case 6
         disp('esce 6')
     otherwise
         disp('il dado è strano')
end
```

Scelta condizionata: Operatori di Relazione

<	Minore di
<=	Minore o uguale a
>	Maggiore di
>=	Maggiore o uguale a
==	Uguale a
~=	Diverso da

Un operatore di relazione tra due variabili è una funzione che ha le due variabili (**array**) come argomenti e che restituisce un valore di tipo logico (**TRUE** o **FALSE**), che può anche essere assegnato ad una variabile.

In Matlab (e in altri linguaggi), i valori logici TRUE e FALSE vengono trattati come valori aritmetici:

TRUE = 1 FALSE = 0

Scelta condizionata: Operatori di Relazione

Esempi:

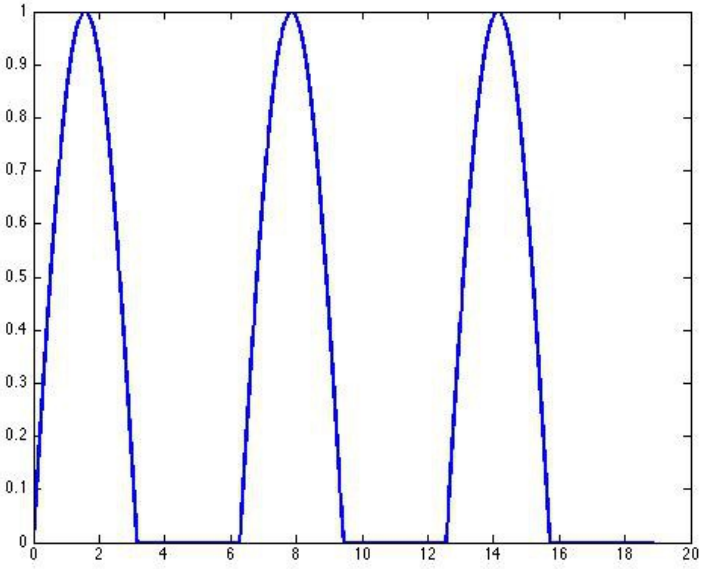
```
>> 4<7
ans =
    1

>> 6<3

ans =
    0

>> a=[2 5 1 7]
a =
    2    5    1    7

>> b=[1 0 7 1]
b =
    1    0    7    1

>> a<b
ans =
    0    0    1    0
```

```
>> a=rand(4,4)
a =
    0.2850    0.1624    0.3966    0.9514
    0.6011    0.0995    0.6163    0.5474
    0.3074    0.7614    0.6294    0.4708
    0.5129    0.9669    0.6881    0.8896

>> b=rand(4,4)
b =
    0.9467    0.6782    0.1403    0.1700
    0.2250    0.8292    0.2128    0.8267
    0.2147    0.4373    0.6823    0.2025
    0.0229    0.1054    0.9046    0.1330

>> t=a>b
t =
     0     0     1     1
     1     0     1     0
     1     1     0     1
     1     1     0     1
```

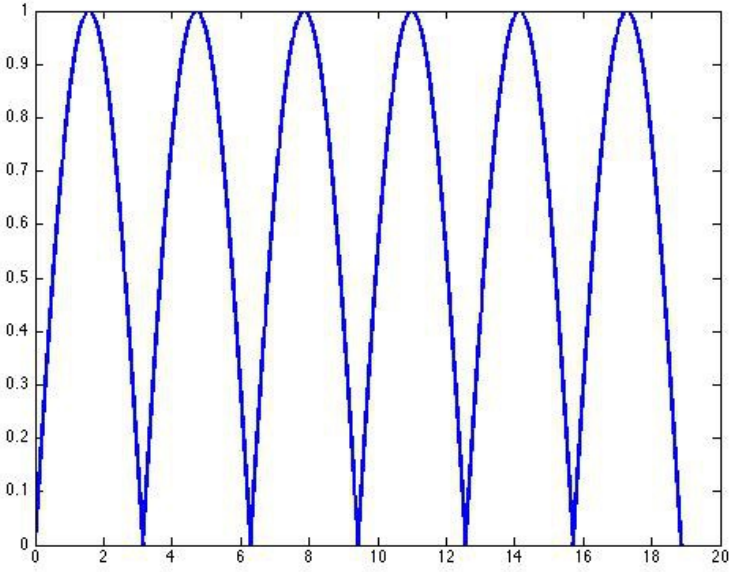
Scelta condizionata: Operatori di Relazione

Esempi:

```
x=linspace(0,6*pi,10000);
y=sin(x);
g=y.*(y>0);
plot(x,g)
```




```
x=linspace(0,6*pi,10000);
y=sin(x);
g=y.*(y>0)-y.*(y<0);
plot(x,g,'LineWidth',2)
```



Operatori logici

Gli operatori logici permettono di combinare le espressioni relazionali

Gli operatori logici agiscono sulle variabili di tipo logico, che possono essere quindi TRUE o FALSE e il risultato è ancora un valore logico

Ricordiamo che in Matlab, i valori TRUE e FALSE sono assegnati rispettivamente a variabili diverse da zero e uguali a zero

Nomi simboli e sintassi degli operatori logici:

Nome	Simbolo	Sintassi
AND	&	a&b
OR	|	a| b
NOT	~	~a
XOR		xor(a,b)
ANY		any(x1,...,xn)
ALL		all(x1,...,xn)

Operatori logici

Esempi:

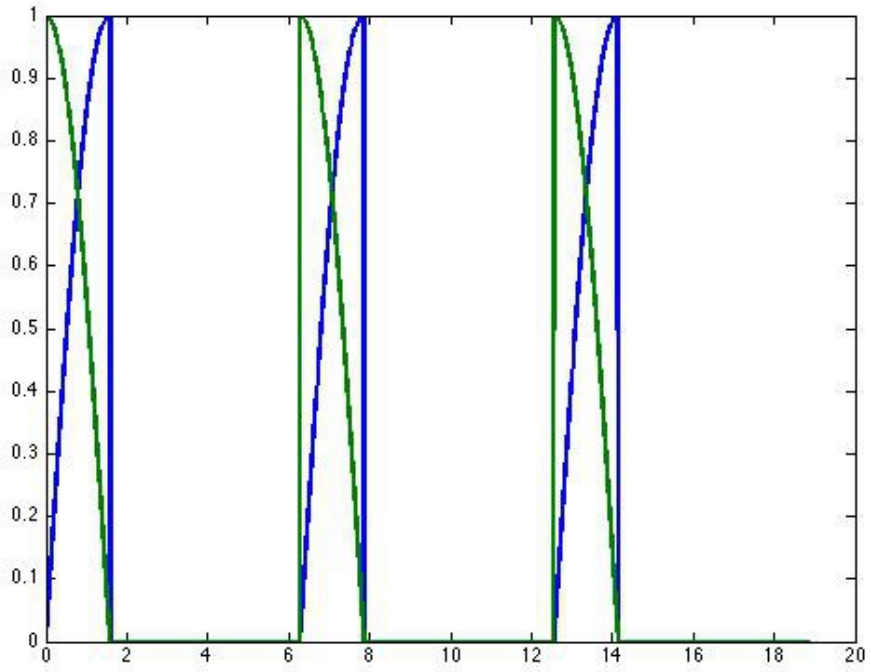
```
>> a=4
a =
     4

>> not(a)
ans =
     0

>> b=0
b =
     0

>> not(b)
ans =
     1
```

```
>> x=[1 -3 0 8]
x =
     1    -3     0     8

>> not(x)
ans =
     0     0     1     0

>> y=[3 1 7 2]
y =
     3     1     7     2

>> and(x,y)
ans =
     1     1     0     1
```

Operatori logici

Esempi:

```
x=linspace(0,6*pi,10000);
ys=sin(x);
yc=cos(x);
intersect=(ys>=0)&(yc>=0);
plot(x,intersect.*ys,x,intersect.*yc,'LineWidth',2)
```



Cicli

Esempio:

```
function [gitt,gittata_teo]=gittata(h,gamma,v0,teta)
%GITTATA in input chiede il passo temporale
% h, il coefficiente di attrito dell'aria gamma, la
% velocità iniziale v0 l'angolo teta in gradi.
% Restituisce nella variabile gittata la distanza
% percorsa sull'asse x da un proiettile
% soggetto alla  forza di gravità e alla forza di
% attrito dell'aria di modulo  gamma*v^2 e
% componenti:Fx= -gamma*v*vx
% Fy= -gamma*v*vy e nella variabile
% gittata_teo il valore calcolato in assenza  di
% attrito.
% L'iterazione della soluzione si interrompe
% quando la  coordinata y del proiettile
% diventa  negativa.
tetarad=teta*pi/180;  % Converte l'angolo in rad
G=9.81; m=1;
N=1e6;   % Numero massimo di iterazioni
%Condizioni iniziali
x=0; y=0; vx=v0*cos(tetarad); vy=v0*sin(tetarad);
 %%METODO DI EULERO
for k=1:N
    mod_v=sqrt(vx^2+vy^2);
    nvx=vx-gamma/m*mod_v*vx*h;
    nvy=vy-(g+gamma/m*mod_v*vy)*h;
    nx=x+vx*h;
    ny=y+vy*h;
    if ny<0
        break
    end
    x=nx; y=ny; vx=nvx; vy=nvy;
end
    gitt=x; gittata_teo=v0^2*sin(2*tetarad)/g;
    return
```

Cicli

Esempio:

```
function Y = cheby(x,p)
%CHEBY    Chebyshev polynomials.
%         Y = CHEBY(X,P) evaluates the first P Chebyshev polynomials
%         at the vector X.  The K'th column of Y contains the
%         Chebyshev polynomial of degree K-1 evaluated at X.

Y = ones(length(x),p);
x = x(:);  % Ensure x is a column vector.
if p == 1, return, end

Y(:,2) = x;
for k = 3:p
  Y(:,k) = 2*x.*Y(:,k-1) - Y(:,k-2);
end
```

Cicli

Esempio:

```
function [x,iter] = sqrtn(a,tol)
%SQRTN    Square root of a scalar by Newton's method.
%         X = SQRTN(A,TOL) computes the square root of the scalar
%         A by Newton's method (also known as Heron's method).
%         A is assumed to be >= 0.
%         TOL is a convergence tolerance (default EPS).
%         [X,ITER] = SQRTN(A,TOL) returns also the number of
%         iterations ITER for convergence.

if nargin < 2, tol = eps; end

x = a;
iter = 0;
xdiff = inf;
fprintf(' k              x_k           rel. change\n')

while xdiff > tol
    iter = iter + 1;
    xold = x;
    x = (x + a/x)/2;
    xdiff = abs(x-xold)/abs(x);
    fprintf('%2.0f:  %20.16e  %9.2e\n', iter, x, xdiff)
    if iter > 50
       error('Not converged after 50 iterations.')
    end
end
```

Cicli

Esempio:

```
function [x_sort,x_mean,x_med,x_std] = marks2(x)
%MARKS2  Statistical analysis of marks vector.
%        Given a vector of marks X,
%        [X_SORT,X_MEAN,X_MED,X_STD] = MARKS2(X) computes a
%        sorted marks list and the mean, median, and standard
deviation
%        of the marks.

x_sort = sort(x);
if nargout > 1, x_mean = mean(x); end
if nargout > 2, x_med  = median(x); end
if nargout > 3, x_std  = std(x); end
```

Functions

- Function Handles
- Anonymous Functions
- Inline Functions
- Subfunctions
- nargin – nargout – varargin – vararout
- Nested Functions
- Private Functions
- Recursive Functions
- Global Variables

Functions

Function Handles

Per fare di una funzione l'argomento di un'altra funzione, si usa una cosiddetta *function handle*.

Viene creata mettendo il carattere @ prima del nome della funzione.

Ad esempio: ezplot(@cos), traccia il grafico della funzione cos(x) tra -2pi e 2pi. Equivale a ezplot('cos')

@cos

equivale a

feval('cos', [-2*pi:h:2pi])

Functions

Anonymous Functions

Queste funzioni rappresentano un modo di scrivere una funzione cosiddetta 'one-line', senza scrivere una M-function.

```
>> f=@(x) exp(x.^2) - 1

f =

    @(x)exp(x.^2)-1
```

f è una *function handle* alla funzione anonima. Quindi può essere passata ad un'altra funzione. Ad esempio, per il plot si può usare ezplot:

```
ezplot(f)
```

Se x è un vettore:

```
>> x=0:2;
>> f(x)
      ans =
            0     1.718    53.5981
```

Functions

Inline Functions

Queste funzioni sono state sostituite dalle anonymous functions. Bisogna conoscerle perchè si possono ancora trovare in molti codici.

Esempio:

```
>> f=inline('exp(x.^2+y.^2)')

f =
     Inline function:
     f(x,y) = exp(x.^2+y.^2)

>> f(2,3)
ans =
   4.4241e+05
```

Functions

Subfunctions

Una *function* può contenere altre *functions*, chiamate *subfunctions i*ntrodotte nella *function* principale (primaria) in ordine qualsiasi.
Le *subfunctions* sono visibili solo alla funzione principale e alle altre *subfunctions*.
L'uso di *subfunctions* evita il proliferare di M-files. Esempio:

```
function max_err = poly1err(n)
%POLY1ERR   Error in linear interpolating polynomial.
%           POLY1ERR(N) is an approximation based on N sample points
%           to the maximum difference between subfunction F and its
%           linear interpolating polynomial at 0 and 1.

max_err = 0;
f0 = f(0); f1 = f(1);
for x = linspace(0,1,n)
    p = x*f1 + (x-1)*f0;
    err = abs(f(x)-p);
    max_err = max(max_err,err);
end

% Subfunction.
function y = f(x)
%F       Function to be interpolated, F(X).
y = sin(x);
```

Functions

Per approfondire:

nargin – nargout – varargin – vararout

Nested Functions

Private Functions

Recursive Functions

Global Variables

Uso delle espressioni logiche

Funzione *find*:

Se applicata ad un vettore x, restituisce un vettore le cui componenti sono gli indici per cui le componenti del vettore x sono diverse da zero:

Es:

```
>> x=[1 2 2 2 0 -2 0]
x =
    1    2    2    2    0   -2    0

>> ind=find(x)
ind =
    1    2    3    4    6
```

Uso delle espressioni logiche

Sia x=[x1 x2...xn] un vettore;

Verificare:

1) x > 0
2) diff (x>0)
3) find(diff(x>0))
4) xs = x(find(diff(x>0)))
5) xd = x(1+find(diff(x>0)))

Uso delle espressioni logiche

Esempi:

```
>> x=[1 5 -3 8 9 -1 10]
x =
    1    5   -3    8    9   -1   10

>> x>0
ans =
    1    1    0    1    1    0    1

>> diff(x>0)
ans =
    0   -1    1    0   -1    1

>> find(diff(x>0))
ans =
    2    3    5    6

>> x(ans)
ans =
    5   -3    9   -1
```