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*

(3): Input-Output dei dati

- Logica binaria e rappresentazione delle informazioni
- Rappresentazione binaria dei numeri
  - Rappresentazione degli interi
  - Rappresentazione di un numero reale
- Rappresentazione esadecimale
- Rappresentazione Binaria dei Caratteri
- Lettura e Scrittura di un file:

`fopen, fread, fscanf, fprintf`

Tipi di dati (o classi)

```
double, single, int, uint
logical
char
struct, cell
function handle
user classes
```

Per conoscere il tipo di dato, si usa la funzione: `class`

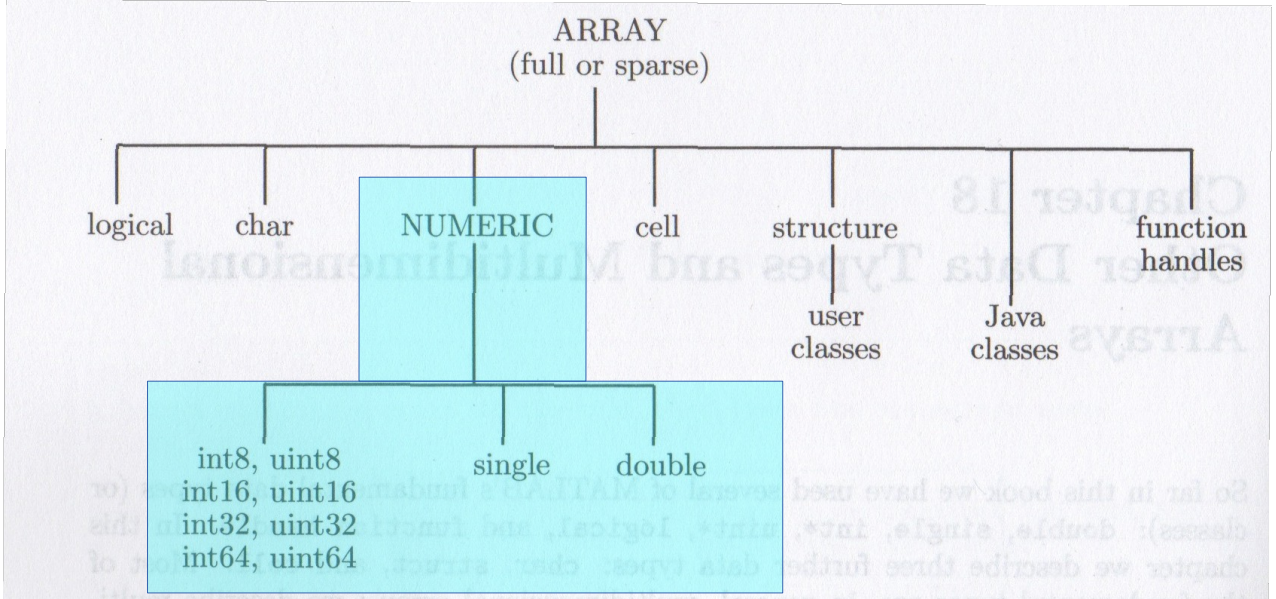


Figure 18.1. *MATLAB data class hierarchy.*

Logica binaria e rappresentazione delle informazioni

La rappresentazione dei dati nelle tecnologie digitali è in logica binaria, in cui l'elemento di base è il bit, variabile a due valori (per convenzione, 0 oppure 1)

Ogni informazione utilizzata da un calcolatore è rappresentata da una sequenza (*string*) di caratteri binari

Per dati con più di due valori, servono più bit:

byte -> gruppo di 8 bit

word -> gruppi di 2 e 4 bytes

Rappresentazione binaria dei numeri

Un numero N in base b è rappresentato come:

$$N=c_0 b^0 + c_1 b^1 + \ldots + c_k b^k$$

$(0 \le c \le b-1)$; per $b=2 \rightarrow$ c: 0, 1

Con k cifre $\rightarrow 2^k$ sequenze diverse

1 byte: codifica per $2^8 = 256$ numeri diversi: 0-255

Rappresentazione binaria dei numeri interi positivi

In un computer il numero di bit dipende dalla capacità di un registro base di memoria.

Generalmente i computer usano parole di 64 bit.

Il massimo numero intero positivo:

$$Imax = 2^{64} - 1 = 1.84467440737095e+019$$

Affinchè i 64 bit siano interpretati come un numero intero, ci vuole un'ulteriore informazione sul tipo di dato:

uint64

(unsigned integer 64 bit)

Rappresentazione binaria dei numeri interi positivi

In Matlab esiste una funzione *built-in*, che trasforma un numero in rappresentazione decimale non negativo N in un numero in rappresentazione binaria di k bit:

```
dec2bin(N,k)
```

Se si vuole implementare un programma che faccia questo, si possono usare le due funzioni di Matlab:

```
floor e mod(x,y)
```

Decimale --> binaria

N	floor(N/2)	mod(N,2)
41	20	1
20	10	0
10	5	0
5	2	1
2	1	0
1	0	1

41 -> 0010 1001

Rappresentazione binaria degli interi negativi

Rappresentazione del **complemento a due**:

Se si vuole rappresentare –M, si prende la rappresentazione di M, si negano i singoli bit (dove c'è 0 si mette 1 e dove c'è 1 -> 0) e si aggiunge 1.

Esempio (con 8 bit): 10: 00001010; -10: 11110101+

1

= 11110110

Rappresentazione binaria degli interi negativi

Ci sono delle limitazioni; Ad esempio, con k bit:

il numero negativo più piccolo è dato da -2^{k-1}

e il valore piu' grande positivo è $2^{k-1}-1$

(Si provi, con k=8, a fare il negativo di 128)

Quindi, ad esempio, con 12 bit, si possono rappresentare numeri tra:

-2048 e 2047

Un tipo di dato con segno, di 32 bit, si indica:

int32

(signed integer 32 bit)

Rappresentazione esadecimale

Base 16: 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F

Ogni cifra esadecimale è rappresentabile con 4 bit. E' immediato passare da questa rappresentazione a quella binaria:

si considera ogni cifra rappresentata dai propri 4 bit e si allinea con le altre cifre (es: 11 → 00010001)

In Matlab esistono funzioni *built-in*:

`dec2hex`

`hex2dec`

`dec2bin`

Rappresentazione binaria di un numero reale

- I numeri reali in un calcolatore si rappresentano in maniera approssimata e simile alla notazione esponenziale in base 10, dove un numero si esprime come:

$$N=(\text{segno}) \times m \times 10^{E}$$

con m: 1<=m<10.

$$\text{Es: } 0.00078 = +\ 7.8 \times 10^{-4}$$

In notazione binaria:

$$x \rightarrow \pm M_2.F \times 2^{E} \quad (1 \leq M_2 < 2)$$

(La mantissa M_2 essendo sempre 1, non si indica)

Es: x = 7.5 :

Come si determina la parte frazionaria F?

Rappresentazione binaria di un numero reale

Es: $x = 7.5 \rightarrow 1.111 \times 2^2$

Come si determina la parte frazionaria F?

Notare che per rappresentare la parte frazionaria di un numero che in base 10 ha un numero finito di cifre, possono essere necessarie un numero infinito di cifre in base 2:

Es: $0.2 \rightarrow 1.1001100011\ldots \times 2^{-2}$

Rappresentazione binaria di un numero reale

Anche per i reali, la precisione con cui si può approssimare un numero dipende dalle dimensioni dei registri. In Matlab si hanno a disposizione 8 byte (64 bit). Un dato di questo tipo si dice di tipo *double* (doppia precisione) ed ha la seguente struttura (*standard IEEE-754*):

s	e1….e11	f1…f52

1 bit -> segno (s=0 → + s=1 → -)

11 bit -> esponente (0-2047)

52 bit -> mantissa (parte frazionaria)

Rappresentazione binaria di un numero reale

I valori 0 e 2047 dell'esponente sono riservati.

Quindi i valori variano tra 1 e 2046.

Per rappresentare anche gli esponenti negativi, si sottrae dall'esponente il numero 1023.

Quindi, gli esponenti variano tra:

emin = 1- 1023 = -1022 ed emax = 2046 – 1023 = +1023

Rappresentazione binaria di un numero reale

Quindi si ha: $x = (-1)^s \times 2^{e-1023} \times 1.f$

Numero reale più grande rappresentabile in questo modo:

$$x\_max\ (\texttt{realmax}) = (-1)^0 \times 2^{2046 - 1023} \times 1. \sum_{i=1}^{52} (1/2)^i =$$

$$8.988465674311580e+307 \times 1.999999999999998 =$$

$$1.797693134862316e+308$$

Numero reale più piccolo rappresentabile in questo modo:

$$x\_min\ (\texttt{realmin}) = (-1)^0 \times 2^{1-1023}$$

$$2.225073858507201e-308$$

Rappresentazione binaria di un numero reale

Casi in cui e vale 0 oppure 2047:

1) e=0, convenzionalmente rappresenta lo zero (anche la mantissa è zero)

2a) e=2047 e mantissa nulla, vengono rappresentati $\pm\infty$

2b) e=2047 e mantissa diversa da zero, viene rappresentato NaN

Rappresentazione binaria di un numero reale

In Matlab, un numero reale può essere visualizzato sia in forma binaria (che esadecimale) e viceversa:

Si usano l'istruzione `num2bin` e `bin2num`

Prima bisogna definire un oggetto detto `quantizer` la cui sintassi è:

```
q=quantizer('double',[64 11])
```

Quindi:

```
num2bin(q,x)
```

```
bin2num(q,stringa_binaria)
```

Rappresentazione binaria di un numero reale

Cifre significative e massima precisione di un `double`:

Un numero che ha k cifre significative ha una precisione dell'ordine di 10^{-k}: una parte su 10^{k}.

Un `double` ha al più 53 cifre binarie significative:

precisione 2^{-53} -> circa 10^{-16}

`eps` rappresenta la più piccola differenza tra 1 e il numero reale immediatamente più grande rappresentabile in `double` e vale $2^{-52} = 2.220\ldots \times 10^{-16}$

Operazioni con i numeri *floating-point*

Valgono:

proprietà commutativa
della somma e del prodotto

Non valgono:

proprietà associativa,
distributiva,
unicità dello zero

Operazioni con i numeri *floating-point*

Non unicità dello zero:

a+b=a, quando b < `eps(a)`:

```
a=1; b=1;
count=0;
while a+b ~=a
   b=b/2;
   count=count+1;
  end

epsilon_half=b
    count
```

Operazioni con i numeri *floating-point*

Associatività violata in presenza di `overflow` e `underflow`: (cancellazione di cifre significative)

Es1: a=1.0e+308; b=1.1e+308; c=-1.001e+308

a+(b+c) = 1.0990e+308

(a+b)+c = `Inf`

Es2: x = 1.e-15;

((1+x)-1)/x

Es3:y=x.^7-7*x.^6+21*x.^5-35*x.^4+35*x.^3-21*x.^2+7*x-1;

(1-2*1.e-08,1+2*1.e-08,401)

Numeri Complessi

La parte immaginaria di un numero complesso si indica con i o j.

1) z= a+bi(j);
2) z=complex(x,y);

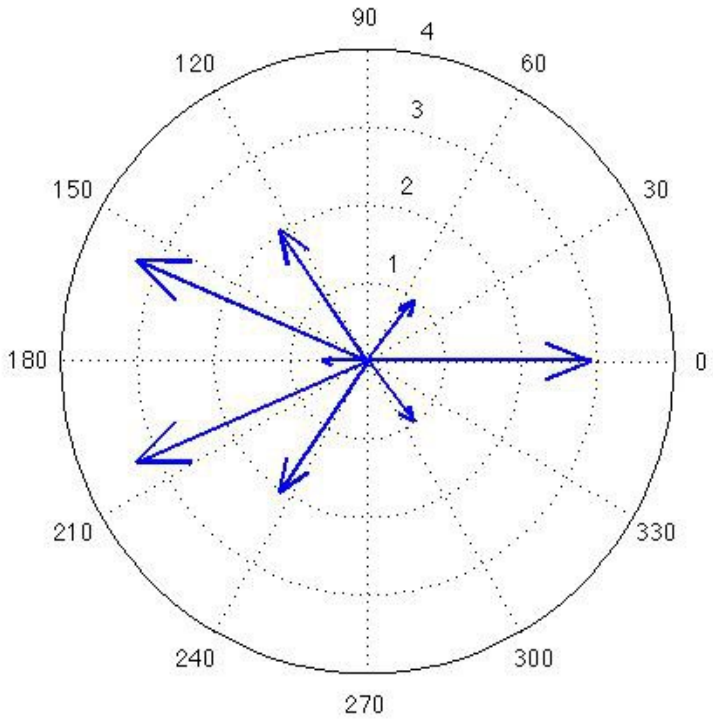
abs(z)=sqrt(a^2+b^2); angle(z);

3) z=abs(z)(cos(angle(z))+isin(angle(z)))

```
real(z)
imag(z)
conj(z)
compass(z)
```

Numeri Complessi

```
Z =
eig(randn(8));
compass(Z)
```



Tipi di dati (o classi): `logical`

`double, single, int, uint`
`logical`
`char`
`struct, cell`
`function handle`
`user classes`

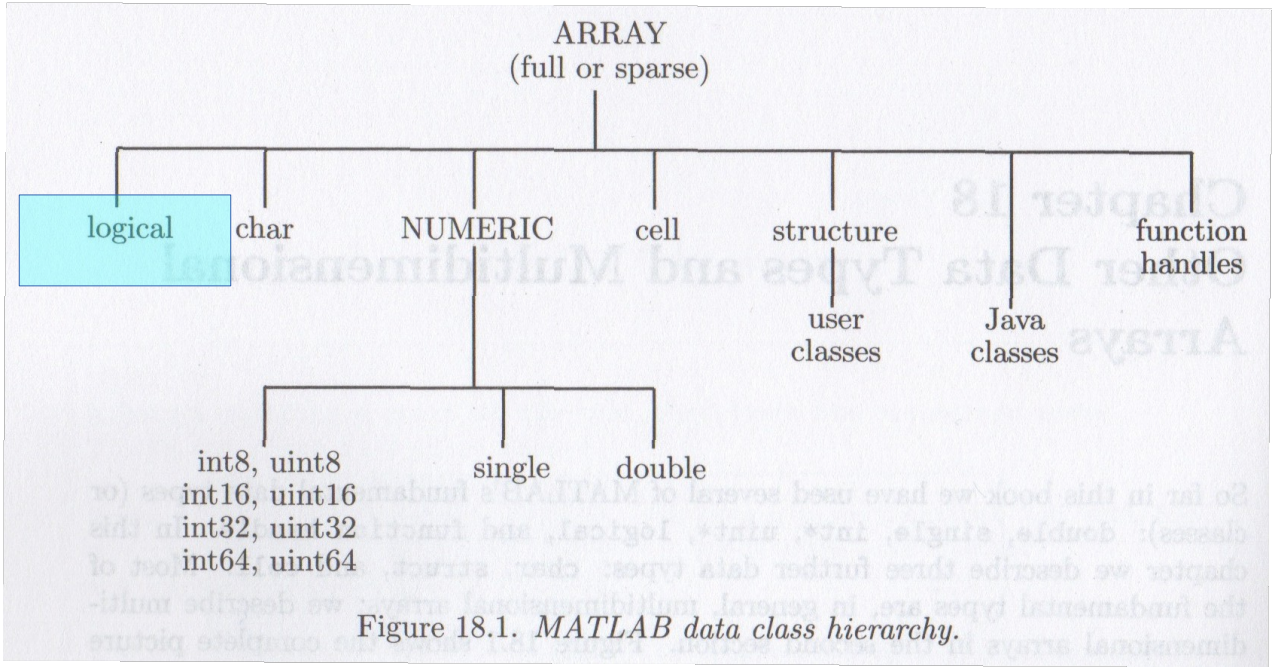


Figure 18.1. *MATLAB data class hierarchy.*

Tipi di dati (o classi): `logical`

I dati di tipo logico possono assume il valore 1 (TRUE), 0 (FALSE).

Dati logici sono prodotti da:

Operatori relazionali (**==**, ~=, <, >, <=, >=)
Funzioni logiche
Funzioni `true` e `false`

Essi occupano 1 byte

Confronti si possono fare tra matrici con le stesse dimensioni o tra matrici e scalari. L'applicazione di operatori e funzioni logiche produce ARRAY logici, i cui elementi sono

0 oppure 1.

Gli ARRAY logici possono essere creati anche applicando la funzione `logical` ad ARRAY numerici

Tipi di dati (o classi): `logical`

Esempi:

```
>> A=[1 -1 7; 7 5 9]

A =

    1    -1     7
    7     5     9

>> B=[0 8 2; 6 1 2]

B =

    0     8     2
    6     1     2
```

```
>> A>B

    1     0     1
    1     1     1

>> A==B

    0     0     0
    0     0     0

>> A~=B

    1     1     1
    1     1     1

>> A>5

    0     0     1
    1     0     1
```

Alcune funzioni di tipo logico:

Table 6.1. *Selected logical* `is*` *functions.*

`ischar`	Test for char array (string)
`isempty`	Test for empty array
`isequal`	Test if arrays are equal
`isequalwithequalnans`	Test if arrays are equal, treating NaNs as equal
`isfinite`	Detect finite array elements
`isfloat`	Test for floating point array (single or double)
`isinf`	Detect infinite array elements
`isinteger`	Test for integer array
`islogical`	Test for logical array
`isnan`	Detect NaN array elements
`isnumeric`	Test for numeric array (integer or floating point)
`isreal`	Test for real array
`isscalar`	Test for scalar array
`issorted`	Test for sorted vector
`isvector`	Test for vector array

Tipi di dati (o classi): char

```
double, single, int, uint
logical
char
struct, cell
function handle
user classes
```

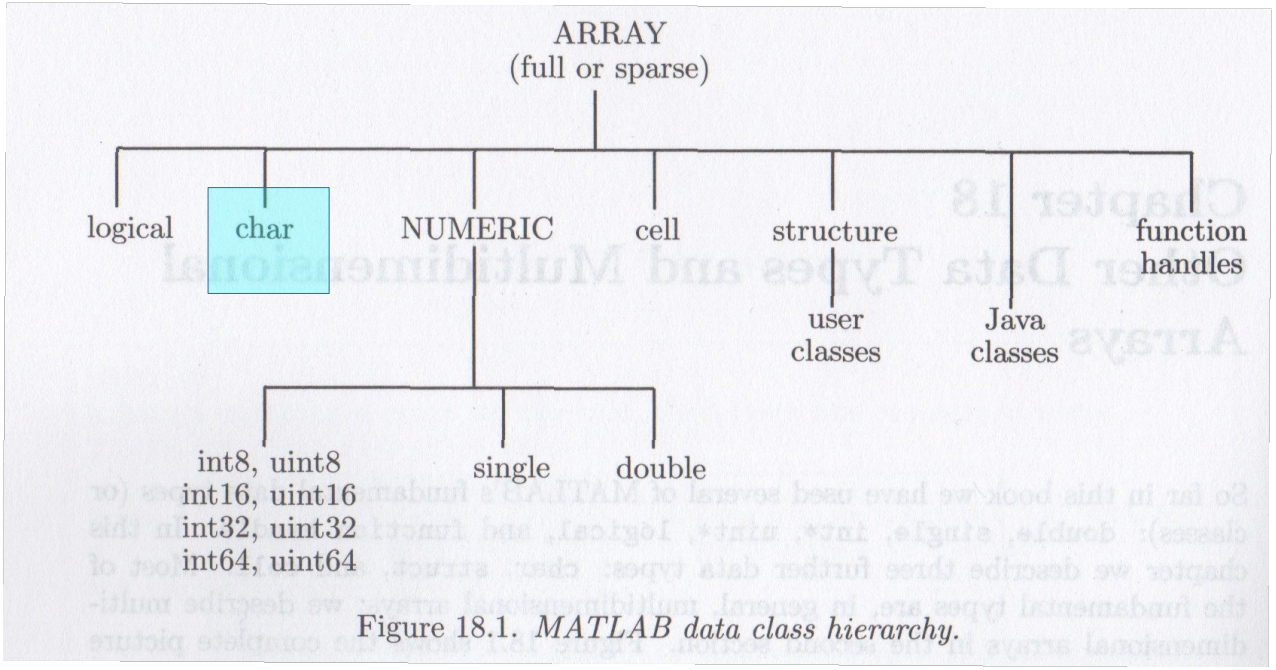


Figure 18.1. *MATLAB data class hierarchy.*

Tipi di dati (o classi): char

Per elaborare testi (tipo di dato: carattere): sempre in logica binaria.

Essendo piccolo il numero di caratteri diversi, è sufficiente un byte (2^8 bit) -> è possibile codificare 256 simboli.

codice ASCII: American Standard Code for Information Interchange -> e' limitata a caratteri codificati con 7 bit

Ad ogni carattere di un testo -> 1 byte (gli altri 128 simboli appartengono a tabelle del codice ASCII estese)

Una pagina di testo è pari a circa quanti byte?

Il tipo char occupa 2 byte in Matlab, ma solo 1 byte e' utilizzato

La funzione char accetta un vettore di interi come input e restituisce un vettore di caratteri

Tipi di dati (o classi): char

Un array di caratteri (char array)
e' una stringa:

```
>> stringa='12srtA'
stringa =
12srtA

>> double(stringa)
ans =
    49    50   115   114   116    65

>> stringa2=char(ans)
stringa2 =
12srtA
```

```
>> whos
Name      Size         Bytes  Class     Attributes

  ans         1x6        48  double
  stringa     1x6        12  char
  stringa2    1x6        12  char
```

Alcune operazioni sulle stringhe

Le stringhe sono indicizzate come gli altri array.

Gli operatori `int2str`, `num2str`, `sprintf` permettono di trasformare array numerici in array di caratteri

La funzione `strcat` concatena stringhe:

```
>> strcat('Lezioni',' Matlab')
ans =
Lezioni Matlab
```

Per confrontare stringhe: `strcmp`, `strcmpi`, `findstr`

Alcune operazioni sulle stringhe: funzione eval

Esempio

```
>> A=magic(3)

A =

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

```
>> for n=1:3
eval(['A', int2str(n), '= A-n*ones(3)'])
end
```

```
A1 =

     7     0     5
     2     4     6
     3     8     1

A2 =

     6    -1     4
     1     3     5
     2     7     0

A3 =

     5    -2     3
     0     2     4
     1     6    -1
```

Tipi di dati (o classi): Cell Array e Structure

```
double, single, int, uint
logical
char
struct, cell
function handle
user classes
```

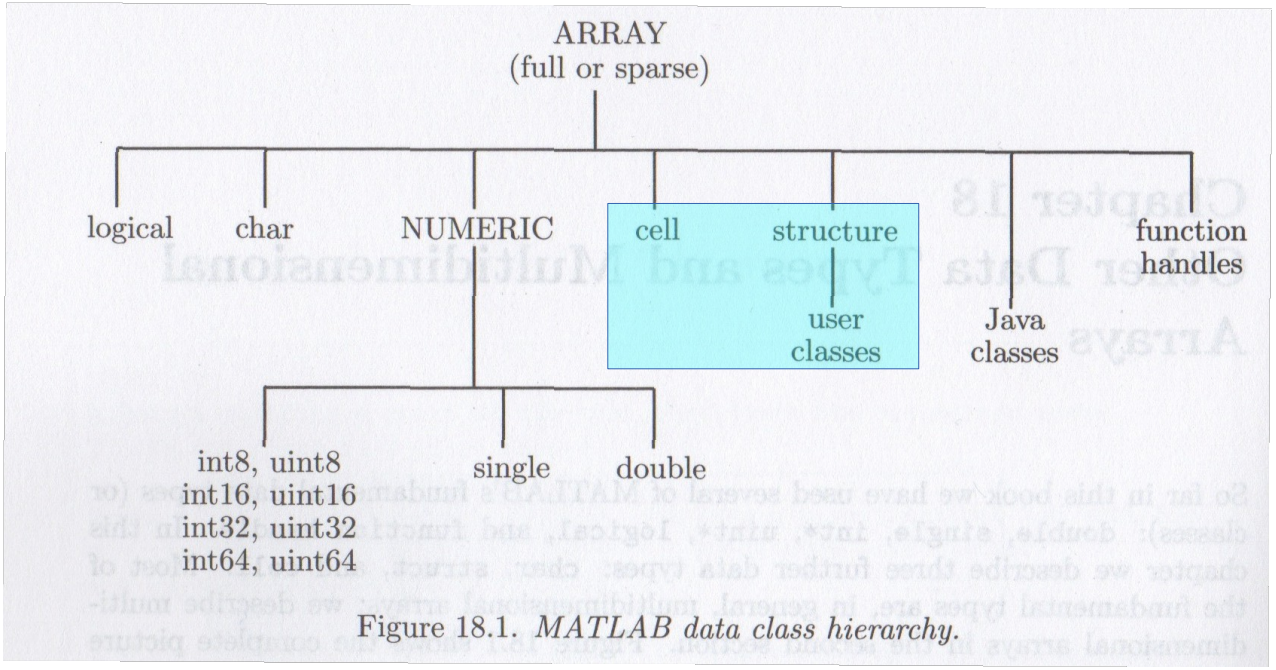


Figure 18.1. *MATLAB data class hierarchy.*

Tipi di dati (o classi): *Cell Arrays* e *Structures*

Questi tipi di dati permettono di raggruppare `arrays` diversi in una singola variabile.

`Cell Arrays` e `Structures` differiscono principalmente per il fatto che una singola cell è identificata da un numero mentre una singola struttura da un nome.

I cell arrays sono arrays i cui elemeni sono cells.

```
>>help struct
```

```
>>help cell
```

Tipi di dati (o classi): Cell Arrays

Esempio di creazione di una Cell Array:

```
>> clear
>> A(1,1)={ones(3)};

>> A(1,2)={8+10i};

>> A(2,1)={'Cell Array & Structures'};


>> A(2,2)={(-2:2)};

>>A

A =

                [3x3 double]    [8.0000 +10.0000i]
    'Cell Array & Structures'           [1x5 double]
```

Tipi di dati (o classi): Stuctures

Esempio di creazione di una Structure:

```
>> cerchio.raggio=2
>> cerchio.centro=[2 1];
>> cerchio.colore='yellow';
>> cerchio.stilelinea='--';

>> cerchio
    cerchio=
        raggio: 2
        centro: [2 1]
        colore: 'yellow'
    stilelinea: '--'


>> whos
  Name          Size                Bytes  Class     Attributes

  cerchio       1x1                   744  struct
```

Input-Output

In Matlab ci sono vari modi per importare dati nell'area di lavoro ed esportare dati dall'area di lavoro su *files*

Importazione files .txt, .xls:

```
A=load('nomefile.txt')
B=xlsread('nomefile.xls')   ([a,b]=xlsread('nomefile.xls') )
```

Uso di `Import Data Wizard`

Esportazione dati in files .txt, .xls:

```
save data.out   A  -ASCII

dlmwrite('data.out',A,';')
dlmwrite('data.xlsx',A,';')
```

Input-Output

```
>> help iofun
```

File input and output.

File import/export functions.

dlmread	- Read ASCII delimited file.
dlmwrite	- Write ASCII delimited file.
importdata	- Load data from a file into MATLAB.
daqread	- Read Data Acquisition Toolbox (.daq) data file.
matfinfo	- Text description of MAT-file contents.

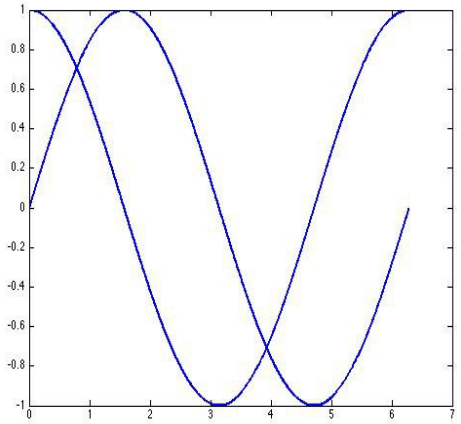
Spreadsheet support.

xlsread	- Get data and text from a spreadsheet in an Excel workbook.
xlswrite	- Stores numeric array or cell array in Excel workbook.

.....

Input dall'utente

```
>> x=input('punto iniziale=')
punto iniziale=10
x =
    10
>> x=input('punto iniziale=','s')
punto iniziale=zero
x =
zero
```



```
>> plot(t,cos(t),'LineWidth',2)
>> hold on
>> plot(t,sin(t),'LineWidth',2)
>> [x,y]=ginput(2)
x =
    0.7823
    3.9274
y =
    0.6988
   -0.7105
```

Output sullo schermo

Se non si usa il `;` tutte le operazioni vengono mostrate sullo schermo.

Alcune funzione controllano il formato dell'output:

```
>> disp('questa è una matrice 3x3 con tutti gli elementi pari a 1'), disp(ones(3))
questa è una matrice 3x3 con tutti gli elementi pari a 1
     1     1     1
     1     1     1
     1     1     1
```

Anche la funzione `fprintf` controlla l'output sullo schermo, quando viene omesso il `fid` (vedremo avanti)

Lettura e scrittura di dati da un file: `fopen - fread`

Archiviazione e recupero di dati

Si vuole elaborare un testo, in Matlab, contenuto in un *file* txt

Ricordiamo che tutti i *files*, che contengano testo, numeri, immagini, etc, hanno sempre la stessa struttura:
una sequenza di cifre binarie.

Quello che differenzia un *file* da un altro *file* è come vengono interpretati i dati.

`fopen`

Per avere l'accesso ad un *file* nella directory corrente, bisogna chiedere l'autorizzazione

Questa si chiede con l'istruzione `fopen`:

```
fid=fopen(nomefile,permission)
```

`fread`

Per leggere il contenuto (binario) di un file si usa la funzione `fread`:

```
A = fread(fid,size,precision)
```

`fread`

`A = fread(fid,size)` legge il numero di elementi specificato da size

Valori ammissibili per size sono:

N	legge N elementi e li mette in un vettore colonna
inf	legge fino alla fine del file
[M,N]	legge gli elementi fino a riempire una matrice M-per-N, secondo colonna

N può essere `inf`

M non può

fread

`A = fread(fid,size,precision)`
legge il *file* secondo il formato specificato da `precision`. L'input di `precision` contiene (se non specificato, la *default precision* è 'uint8'):

MATLAB	Description
'uchar'	unsigned integer, 8 bits.
'schar'	signed integer, 8 bits.
'int8'	integer, 8 bits.
'int16'	integer, 16 bits.
'int32'	integer, 32 bits.
'int64'	integer, 64 bits.
'uint8'	unsigned integer, 8 bits.
'uint16'	unsigned integer, 16 bits.
'uint32'	unsigned integer, 32 bits.
'uint64'	unsigned integer, 64 bits.
'single'	floating point, 32 bits.
'float32'	floating point, 32 bits.
'double'	floating point, 64 bits.
'float64'	floating point, 64 bits.

Lettura binaria di un file di testo

Esempio:

```
fid = fopen('text_sample','r')
a=fread(fid,8,'ubit8') % legge per 8 volte un dato rappresentato con 8
                       % bit
fclose(fid)

fid = fopen('text_sample','r')
a=fread(fid,1,'double','s') % legge 1 dato double, interpreta gli 8
                            % byte come un double
fclose(fid)
```

(s -> *big endian byte ordering* a -> *little endian byte ordering*)

Lettura di un file formattato: `fscanf`

(`>>help fscanf`)

Esempio:

vogliamo leggere, per elaborarli, dati da un file (*area_macchiesolari.txt*) in cui vengono riportate informazioni sulle macchie solari su un lungo intervallo temporale:

```
Yr  Mn     Year     Areas,mph
182101     1821.04      305
182102     1821.13       39
.....................................
198911     1989.88     2649
198912     1989.96     2543
```

Vogliamo leggere il file in modo da avere la seconda e la terza colonna del file in due vettori di numeri floating point (la sintassi dei comandi è mutuata dal C)

La funzione di Matlab che consente di fare questo tipo di lettura è la `fscanf`

Lettura di un file formattato: `fscanf`

`fscanf` legge dati formattati da file identificati da `fid`, li converte secondo il `format` e li scrive in una matrice A:

```
fid = fopen('nome_file','r')
[A,COUNT] = fscanf(fid,format,size)
```

`COUNT` è opzionale e registra il numero di dati letti.

`SIZE` è opzionale e fissa il numero di elementi letti dal file. Quando non c'è, l'intero file viene letto. Quando c'è, può essere:

N	legge N elementi e li mette in un vettore colonna
inf	legge fino alla fine del file
[M,N]	legge gli elementi fino a riempire una matrice M-per-N, secondo colonna

Lettura di un file formattato: `fscanf`

`format:`

È una stringa (caratteri di conversione) che inizia col carattere % seguito da tre parametri:

`*`	(opzionale): indica che il dato va letto ma non restituito (ignorato)
`Numero`	(opzionale): numero di cifre (caratteri) che vanno letti di quel dato
`Tipo di dato`	cui deve essere convertito il dato letto (obbigatorio):

Esempi:

`%s`	leggi una stringa (legge una stringa dal primo all'ultimo carattere prima dello spazio, saltandola)
`%d`	leggi un intero
`%g (%f, %e)`	leggi un floating point
`%4d`	leggi un intero di 4 cifre
`%5s`	leggi una stringa di 5 caratteri
`%c`	leggi un carattere compresi gli spazi

Notare: la stringa di controllo viene applicata **sequenzialmente** fino alla lettura dei dati secondo `size`

Lettura di un file formattato: `fscanf`

% Lettura dati dal file *area_macchiesolari.txt*

```
fid=fopen('area_macchiesolari.txt','r');
fscanf(fid,'%s',4); % legge una stringa per 4 volte
% senza assegnarla
a=fscanf(fid,'%4d %2d %g %g',[4, inf]);
a=a'; % trasposta
anni=a(:,3);
areamacchie=a(:,4);
plot(anni,areamacchie)
```

Un modo alternativo per saltare (leggere e ignorare) una riga di un file di testo fino al carattere newline e' con l'istruzione:

```
tline=fgetl(fid)
```

Lettura di un file formattato: `fscanf`

```
% Lettura dati dal file area_macchiesolari.txt
% Alternativa: si ignorano I primi due dati

fid=fopen('area_macchiesolari.txt','r');
fscanf(fid,'%s',4);
a=fscanf(fid,'%*4d %*2d %g %g',[2, inf]);
a=a';
anni=a(:,1);
areamacchie=a(:,2);
plot(anni,areamacchie)
```

Scrittura di dati su un file: `fprint`

`fid=fopen(nomefile,'w')` apre un file (se non esiste, lo crea) e inizia a scrivere dall'inizio

oppure

`fid=fopen(nomefile,'a')` apre un file (se non esiste, lo crea) e inizia a scrivere dalla fine

```
fid=fprintf(fid, format, A)
```

`A` è una matrice da cui vogliamo prendere i dati

`format` ha la stessa sintassi del format per fscanf

(tra i caratteri di controllo della stringa meritano di essere citati `\n` (new line) e `\t` (tabulazione orizzontale)

I dati vengono letti per colonna (in A) e scritti per riga, secondo il `format`

Scrittura di dati su un file: `fprint`

Supponiamo di avere un vettore `t`, lungo `N`, la cui i-esima componente rappresenta il tempo trascorso al rilevamento del dato i-esimo.

In una matrice `M(N,2)`, sono registrati i dati di due grandezze fisiche (ad es posizione e velocità)

Si vogliono scrivere i dati su un file in modo che nella colonna 1 ci sia il tempo, nella 2 la posizione e nella 3 la velocità:

1) Si deve prima preparare una matrice A opportunamente costruita.

2) Occorre scegliere il formato di scrittura.

Scrittura di dati su un file: `fprint`

Esempio:

```
function scrividati(nomefile,t,M)
% Scrive nel file di nome nomefile.txt i dati contenuti nel vettore t(Nx1)
% e nella matrice M (Nx2) disponendoli per tre colonne

nome=strcat(nomefile,'.txt');
A=[t(:),M];
fid=fopen(nome,'w');
fprintf(fid,'%11.7f\t %11.7f\t %11.7f\n',A.');
fine=fclose(fid);
if fine==0
disp('scrittura dei dati terminata')
end
```

Lettura di dati da un file:

Esempio: (le_graf .m)

```
clear x y fname
fname = input('Nome del file da leggere? ','s');
nski=0;
[x,y]=leggiv(fname,nski);
disp(sprintf('Trovati %d dati (coppie)',length(x)))
plot(x,y,'*');
grid on;
```

Lettura di dati da un file e scrittura su una matrice:

Esempio: (leggiv.m)

```
function [x,y]=leggiv(name,nski)
% [x,y]=leggiv(name,nski)
% salta NSKI righe
% poi legge due colonne reali
%separate dalla virgola

[fid, mess]=fopen(name,'r');
if fid<0
   disp(mess);
    return
end
if(nski>0)
    for i=1:nski
        fgetl(fid);
    end
end
a=fscanf(fid,'%g %g',[2 inf]);
fclose(fid);
a=a';
x=a(:,1);
y=a(:,2);
return
```

Lettura di dati da un file e scrittura su una matrice:

Esempio: (leggivar.m)

```
function a=leggivar(fname,nski,ncol)
% a=leggivar(fname,nski,ncol)
% salta NSKI righe
% poi legge NCOL colonne reali
% separate da spazi
% restituisce la matrice

[fid, mess]=fopen(fname,'r');
if fid<0
   disp(mess);
   return
end
```

```
if(nski>0)
   for i=1:nski
        fgetl(fid);
    end
end
formato='';
for i=1:ncol
    formato=[formato '%g '];
end
a=fscanf(fid,formato,[ncol inf]);
fclose(fid);
a=a';
return
```