Un linguaggio per il calcolo numerico

Vittorio Zaccaria

November 23, 2018

Outline

Introduzione al linguaggio

Scripts

Astrazione mediante i tipi di dato

Meccanismi strutturati per il controllo della sequenza di esecuzione

Il concetto di sottoprogramma: funzioni e procedure come astrazioni.

Parametri, modalità di passaggio dei parametri, effetto di un sottoprogramma.

Gestione dei file. Gestione delle matrici.

Tecniche di visualizzazione grafica.

Introduzione alla ricorsione.

Introduzione al linguaggio

Cosa è Matlab

- Facilita lo sviluppo di programmi che eseguono complesse elaborazioni di calcolo numerico grazie a:
  1. un ambiente di sviluppo integrato ed uno specifico linguaggio di programmazione
  2. una ricca libreria di funzioni matematiche
- E' uno strumento commerciale ma ne esiste una alternativa gratuita di nome Octave molto simile a Matlab in molti aspetti
  `http://www.gnu.org/software/octave`
- Libro di testo: Introduzione alla programmazione in MATLAB. Campi, Di Nitto, Loiacono, Morzenti, Spoletini. Esculapio Editrice.

Linea comando

E’ un linguaggio interpretato, non richiede quindi la fase di traduzione in codice macchina. Il sorgente viene analizzato da un programma interprete che esegue direttamente tutti i comandi richiesti

Esempio di interazione:

```
> 3 + 4

ans =  7
```

Variabili

- Una variabile Matlab è un nome alfanumerico assegnato ad una o più celle della memoria centrale.
- Possiamo scrivere all'interno di queste variabili inizializzandole per:
  1. assegnamento (=)
  2. lettura da tastiera
  3. lettura da file
- Esempio di inizializzazione per assegnamento:

```
1  > a = 1 + 1
2  a =  2
```

Comportamento delle variabili

Una volta assegnato il valore ad una variabile, possiamo usare il suo nome per accedere a tale valore.

```
1  a = 1 + 1
2  b = 3 + a
3  a = a + b
4  b = 3 + a
```

```
a =  2
b =  5
a =  7
b =  10
```

Variabili predefinite

Nome	Descrizione
I, i, J, j	immaginari puri
Inf, inf	Infinito., 1/0, overflow
NaN, nan	Non un numero, ‘0/0’
eps	Precisione macchina
pi	π

Scripts

Cosa è uno script

- Uno script è un file di testo contenente una sequenza di comandi MATLAB
    1. non deve contenere caratteri di formattazione (solo testo puro)
    2. viene salvato con estensione `.m`
- I comandi all'interno di uno script sono eseguiti sequenzialmente, come se fossero scritti nella finestra dei comandi
    1. Per eseguire Il file si digita il suo nome sulla linea comando dell'interprete (senza `.m`)
    2. I risultati appaiono nella finestra dei comandi (se non usiamo il punto e virgola `;`)

Vantaggi e svantaggi

- Uno script può
  - essere ri-eseguito
  - essere facilmente modificato
  - essere spedito a qualcuno
- Uno script NON
  - accetta variabili di input
  - genera variabili di output
- Uno script opera sulle variabili del workspace che può arricchire introducendone di nuove

Come creare uno script

- Può essere creato utilizzando un qualsiasi editor di testo
- Ricordarsi di salvare il file come “solo testo” e di dare l’estensione `.m`
- Il file di script deve essere presente nella directory corrente in cui lanciate l’interprete.
- Matlab include un editor dove creare o modificare script

Astrazione mediante i tipi di dato

Caratteri e stringhe

Una stringa è un array di valori particolari; ognuno di essi è un *carattere*.

```
1  s = 'stringa semplice'
2  s(2)
3  s(1:7)
4  s(9:end)
```

```
s = stringa semplice
ans = t
ans = stringa
ans = semplice
```

Numeri complessi

Un numero complesso è la somma di un numero reale e un numero immaginario:

```
h = 3.1 + 10i
p = 1 + 1i
h*p
```

```
h =   3.1000 + 10.0000i
p =  1 + 1i
ans =  -6.9000 + 13.1000i
```

Array

Una variabile array si distingue da una variabile normale (o scalare) poiché contiene una sequenza di valori. Per crearla utilizziamo le parentesi quadre `[]`.

```
1  a = [ 5 8 (9+9+1) ]
```

```
a =

    5    8   19
```

Array - Accesso agli elementi

Possiamo accedere agli elementi di un array utilizzando le parentesi tonde `()`. All'interno specifichiamo la posizione, partendo da 1.

```
1  a = [ 7 3 8  ]
2  a(2)
```

```
a =

   7   3   8

ans =  3
```

Array - Assegnamento

Possiamo assegnare un elemento singolo dell'array

```
1  a = [ 1 2 3 ]
2  a(2) = 5
```

```
a =

   1    2    3

a =

   1    5    3
```

Array - Generazione

- Non tutti gli elementi devono essere specificati alla creazione dell'array; in questo caso, `c` non esiste ma lo creiamo direttamente assegnando uno dei suoi elementi.
- Possiamo anche estendere un array successivamente.

```
1  c(3) = 1
2  c(5) = 7
```

```
c =

   0   0   1


c =

   0   0   1   0   7
```

Array - Generazione di sequenze

- Possiamo creare array con sequenze regolari di numeri, utilizzando l'operatore `:`.

```
1  c = 1:5
2  d = 1:2:5
```

```
c =

   1   2   3   4   5

d =

   1   3   5
```

Array - Operatori

L'aritmetica normale funziona anche per gli array, elemento per elemento, anteponendo un punto `.` all'operatore.

```
a = 1:3;
b = 2:4;
a .* b
```

```
ans =

    2    6   12
```

Matrici

Una matrice è essenzialmente una tabella di valori, ognuno dei quali ha una riga ed una colonna.

```
1  a = [ 1 2; 3 4]
2  a(2,1)
```

```
a =

   1   2
   3   4


ans =  3
```

Matrici e array

Un array è una matrice con una sola riga. Un array trasposto è una matrice con una sola colonna.

```
1  x = 1:2
2  y = x'
```

```
x =

    1    2

y =

    1
    2
```

Matrici - Creazione

Possiamo comporre matrici più grandi da matrici piccole.

```
m(1:2, 1:2) = 2
n = [m (m .+ 2) ]
```

```
m =

 2   2
 2   2

n =

 2   2   4   4
 2   2   4   4
```

Matrici - Generazione

Possiamo creare una matrice attraverso le funzioni contenute nell'interprete:

```
1  a = ones(2,2)
```

```
a =

   1   1
   1   1
```

Per altre funzioni di questo tipo, si veda (questo link).

Valore logico

E’ un tipo di dato che può avere solo due valori

- 0
- 1

I valori di questo tipo possono essere generati

- dagli operatori relazionali
- dagli operatori logici

I valori logici occupano un solo byte di memoria (i numeri ne occupano 8)

Operatori relazionali

Gli operatori relazionali operano su tipi numerici o stringhe e ritornano un valore logico.

```
3 > 4
3 >= 4
3 < 4
1 + 1 == 2
3 ~= 4
```

```
ans = 0
ans = 0
ans =  1
ans =  1
ans =  1
```

Operatori relazionali applicati ad array

```
[ 3 1 ] > [ 0 2 ]
'abc' > 'ABC'
```

```
ans =

   1   0

ans =

   1   1   1
```

Operatori logici per array

```
a = [ 0 0 1 1 ]';
b = [ 0 1 0 1 ]';
[ a b (a & b) (a | b) ]
```

```
ans =

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

Funzioni logiche

Le più importanti sono `all` and `any`:

```
a = [ 0 0 0 0 ];
b = [ 0 0 1 0 ];
c = [ 1 1 1 1 ];
[ all(a) all(b) all(c) ]
[ any(a) any(b) any(c) ]
```

```
ans =

   0   0   1

ans =

   0   1   1
```

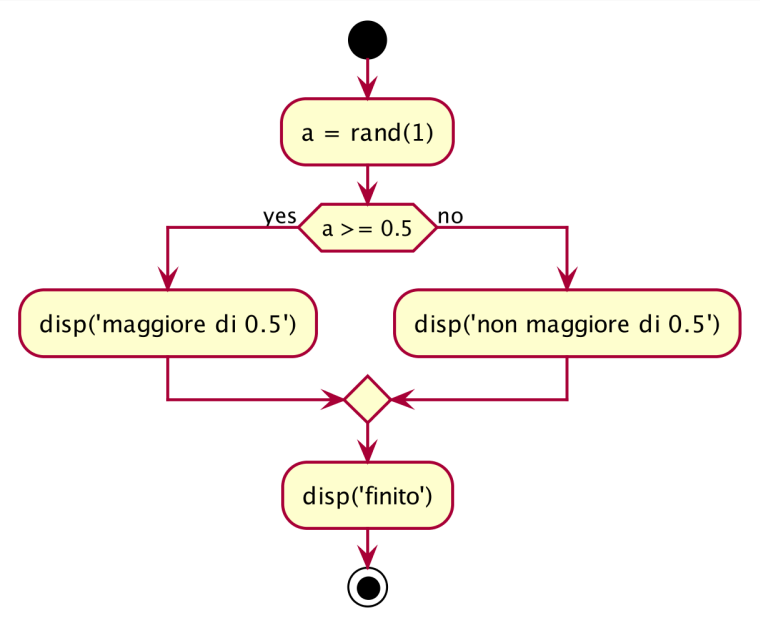
Funzione `find`

```
c = [ 5 8 9 ];
d = [ 10 2 1 ];
find(c>d)
c(find(c>d))
```

```
ans =

   2   3

ans =

   8   9
```

Meccanismi strutturati per il controllo della sequenza di esecuzione

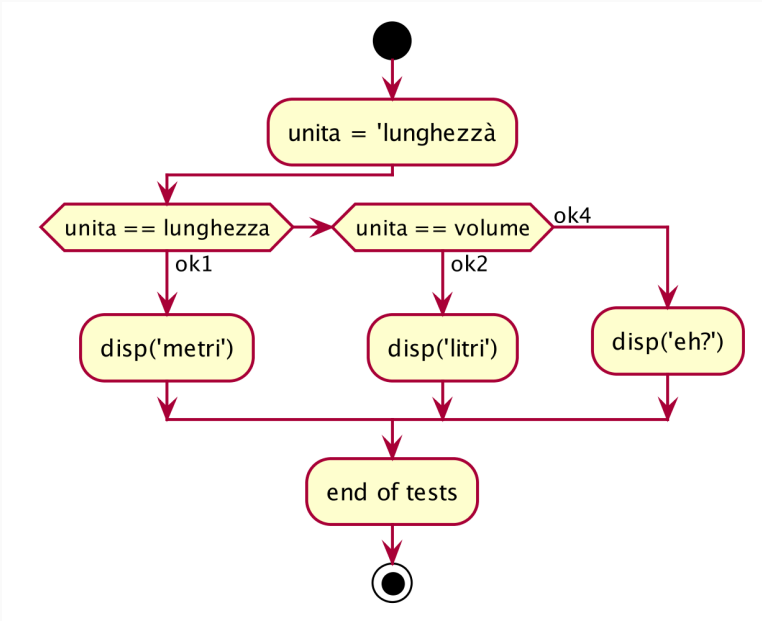
Costrutto if




```
1  a = rand(1)
2  if a >= 0.5
3    disp('maggiore di 0.5')
4  else
5    disp('non maggiore di 0.5')
6  end
7  disp('finito')
```

```
a =  0.41638
non maggiore di 0.5
finito
```

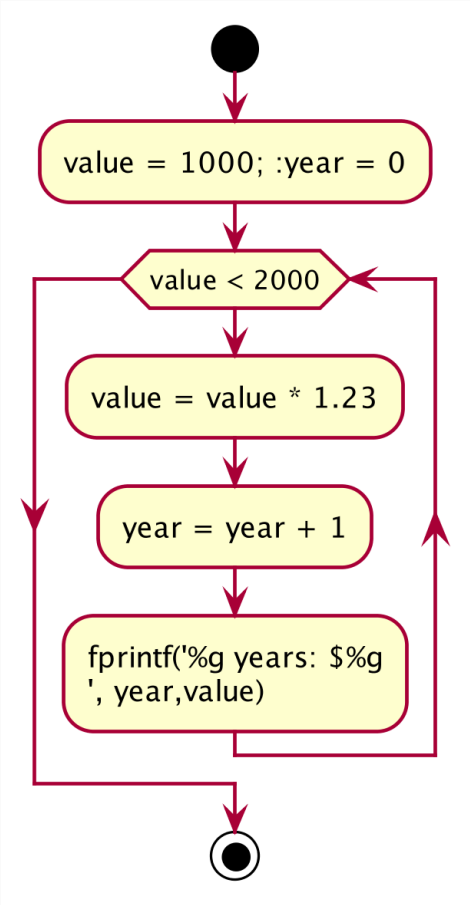
Costrutto switch/case




```
1  unita = 'lunghezzà
2  switch unita
3    case 'lunghezzà
4       disp('metri')
5    case 'volume'
6       disp('litri')
7    otherwise
8       disp('eh?')
9  end
```

```
unita = lunghezza
metri
```

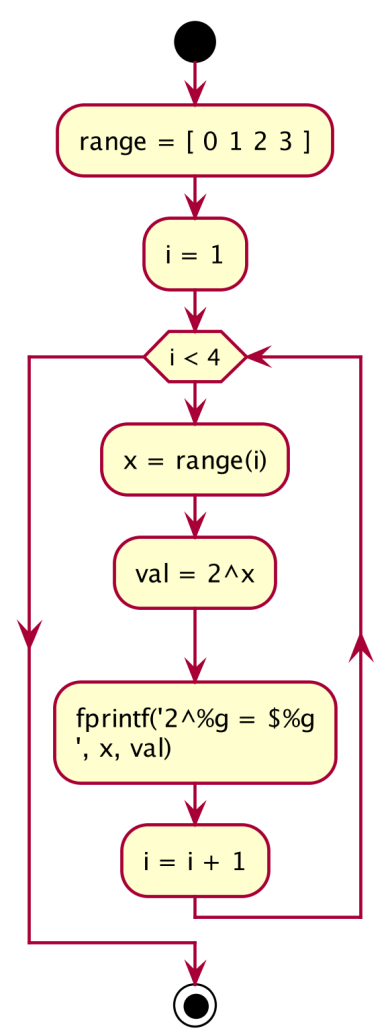
Costrutto while




```
1  value = 1000;
2  year = 0;
3  while value < 2000
4    value = value * 1.23;
5    year = year + 1;
6    fprintf('%g years: $%g\n', year,value
7  end
```

```
1 years: $1230
2 years: $1512.9
3 years: $1860.87
4 years: $2288.87
```

Costrutto for




```
1   for x = 0:1:3
2       val = 2^x;
3     fprintf('2^%g = $%g\n', x, val)
4   end
```

```
2^0 = $1
2^1 = $2
2^2 = $4
2^3 = $8
```

Il concetto di sottoprogramma: funzioni e procedure come astrazioni.

A cosa servono le funzioni?

DRY (dont repeat yourself)

- nello stesso programma
- in più programmi

Forzano una miglior struttura del codice

Testabilità

- Posso verificare il comportamento dei singoli blocchi senza guardare il resto del programma

Problema

```
X = [ 5, 8, 10, 22, 14 ];
somma = 0;
for x = X
  somma = somma + x;
end
size(X,2);
media = somma/size(X,2)
```

```
media =  11.800
```

* Se devo calcolare la media di un altro vettore Y, cosa faccio? riscrivo tutto? E' possibile descrivere il metodo in maniera generica e richiamarlo quando opportuno?

Soluzione

Diamo un nome al metodo (`calcolaMedia`) e scriviamo in un file `calcolaMedia.m` la sua descrizione:

```
function media = calcolaMedia(V)
  somma = 0;
  for v = V
    somma = somma + v;
  end
  media = somma/size(V,2);
end
```

Il metodo si basa su una informazione nota a priori (v, ovvero il parametro in ingresso) e ne genera una in uscita (`media`, ovvero parametro in uscita).

Soluzione (2)

```
% io salvo le funzioni in una cartella 'functions'
addpath('./functions');

X = [ 5, 8, 10, 22, 14 ];
mediax = calcolaMedia(X)
Y = [ 10, 9, 11, 21 ];
mediay = calcolaMedia(Y)
```

```
mediax =  11.800
mediay =  12.750
```

Invocazione

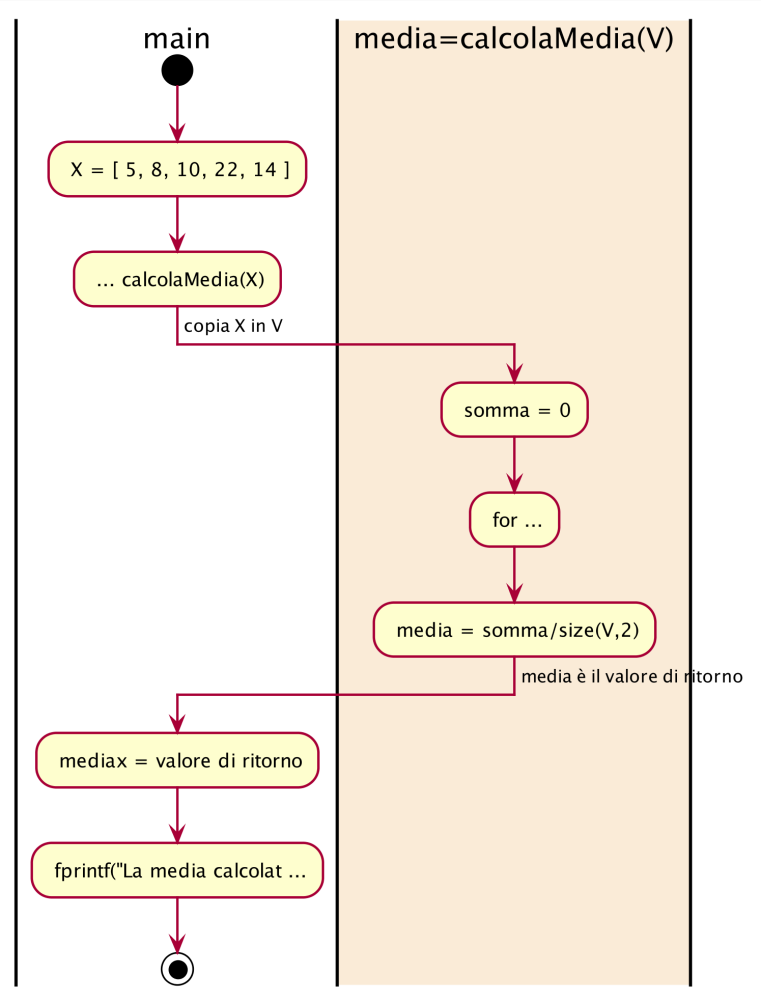
Consideriamo questo esempio:

```
addpath('./functions');
X = [ 5, 8, 10, 22, 14 ];
mediax = calcolaMedia(X)
fprintf('La media calcolata è %g', mediax)
```

Parametri, modalità di passaggio dei parametri, effetto di un sottoprogramma.

Invocazione (2)

L'invocazione avviene utilizzando le parentesi () dopo il nome della funzione stessa. x è chiamato, in questo caso, parametro attuale della funzione `calcolaMedia`. v invece è chiamato parametro formale.



Funzioni con più parametri

Supponiamo di avere questa funzione

```
1  function [minore, maggiore] = calcolaMinMax(a, b, c)
2    minore = min([a,b,c]);
3    maggiore = max([a,b,c]);
4  end
```

Funzioni con più parametri (2)

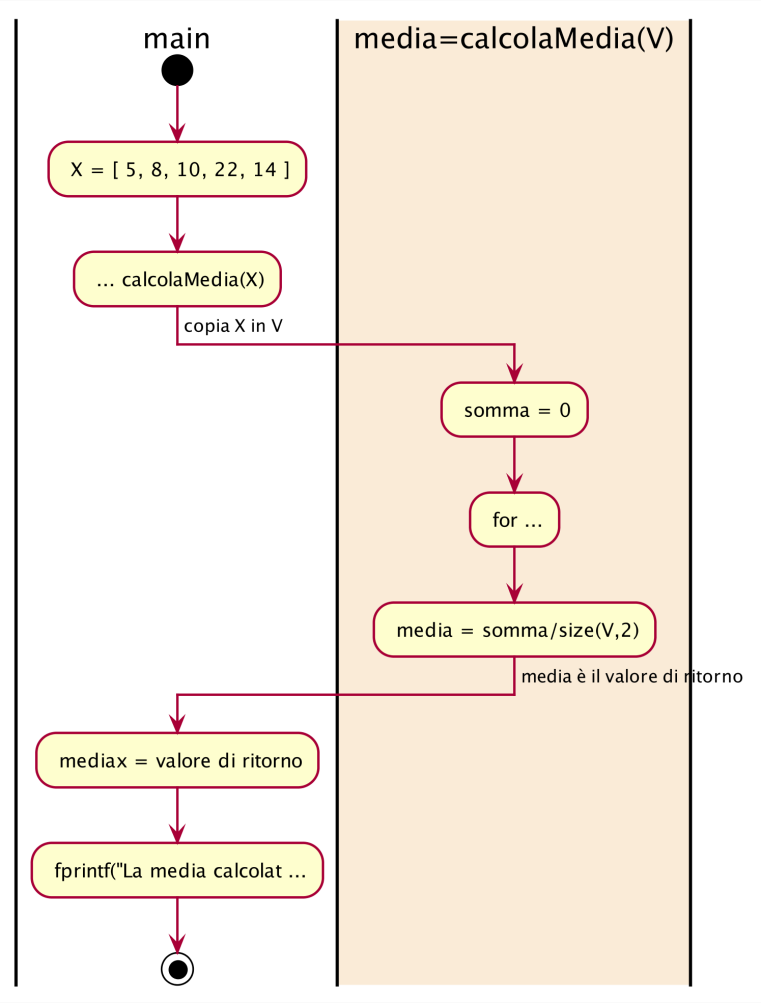
Quando la invochiamo, dovremo passare tre parametri in ingresso e salvare esplicitamente i valori di ritorno in due variabili differenti:

```
addpath('./functions');
X = [ 5, 8, 10, 22, 14 ]; Y = [ 2, 5 ]; Z = [ 21, 3, 5 ];
disp('cosi perdo il secondo valore di ritorno')
calcolaMinMax(X, Y, Z)
disp('cosi salvo entrambe i valori di ritrono in s e z:')
[s, t] = calcolaMinMax(X, Y, Z)
```

```
cosi perdo il secondo valore di ritorno
ans =  2
cosi salvo entrambe i valori di ritrono in s e z:
s =  2
t =  22
```

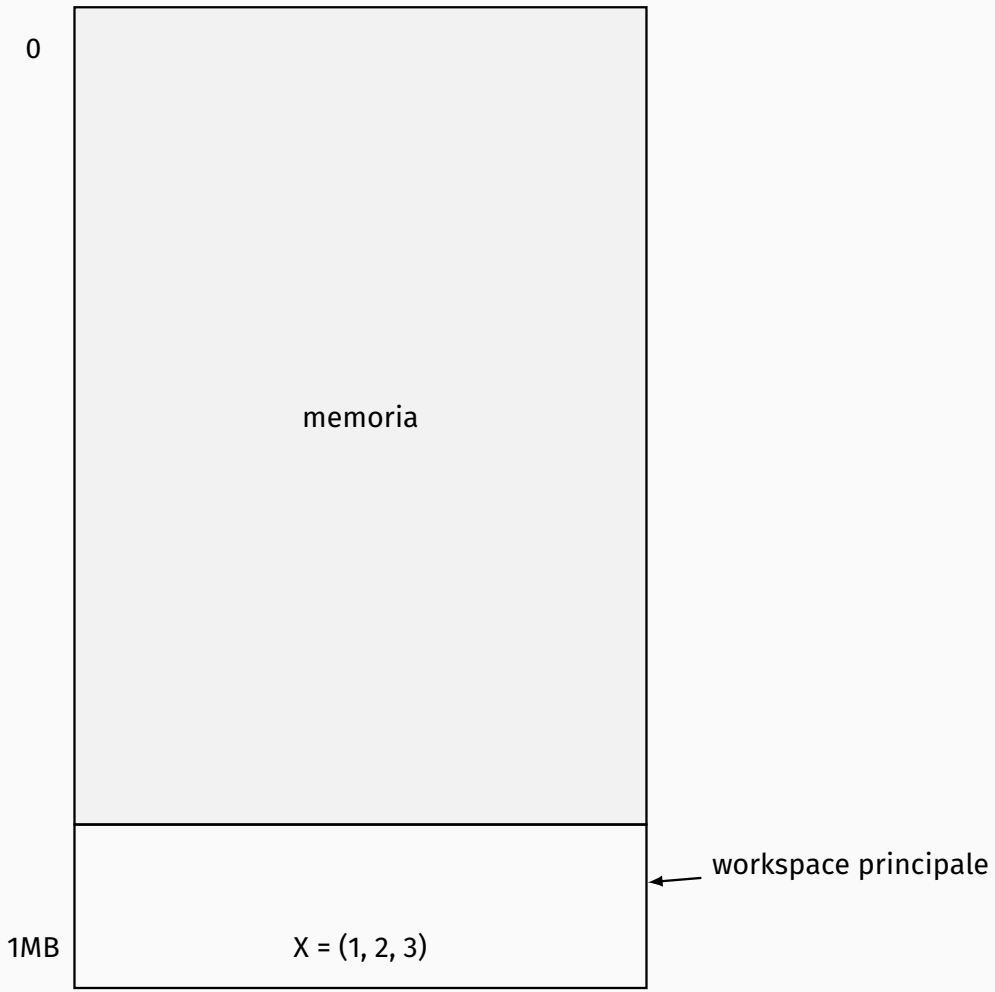
Workspaces

Abbiamo visto all'inizio che le variabili non sono altro che celle di memoria con un nome. Dove si trovano allora, in memoria, x, v, `media` e `mediax`?



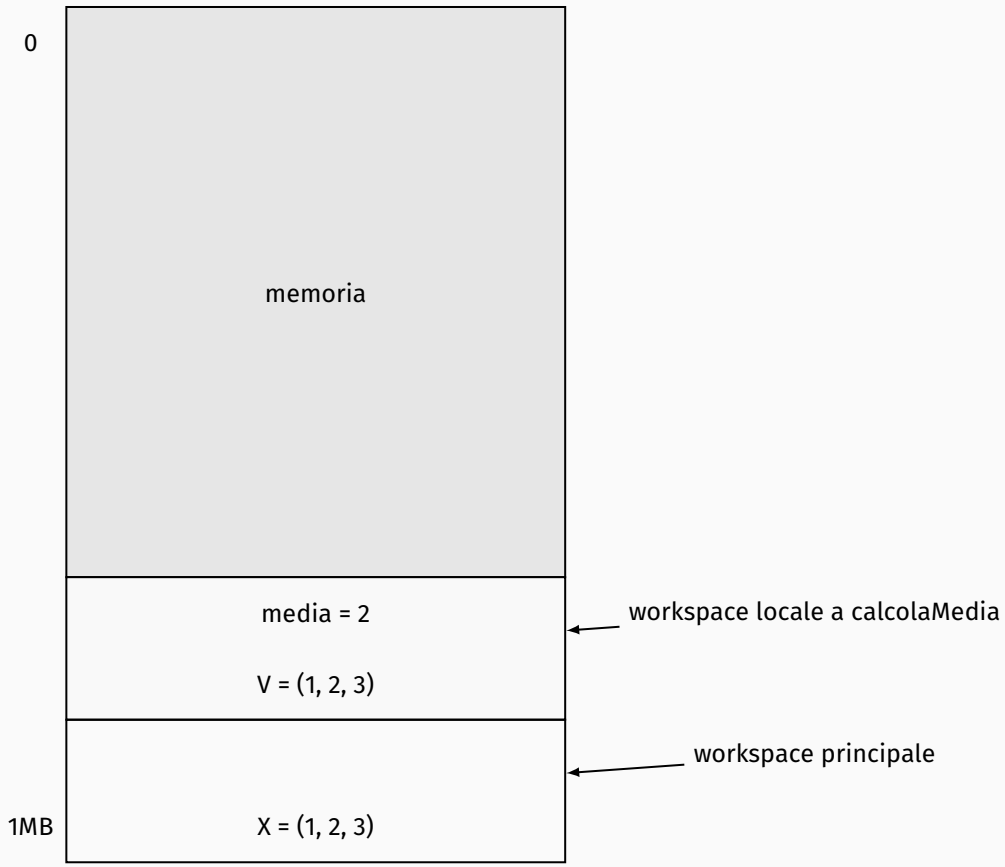
Stack

Prima
dell'esecuzione di
`calcolaMedia`



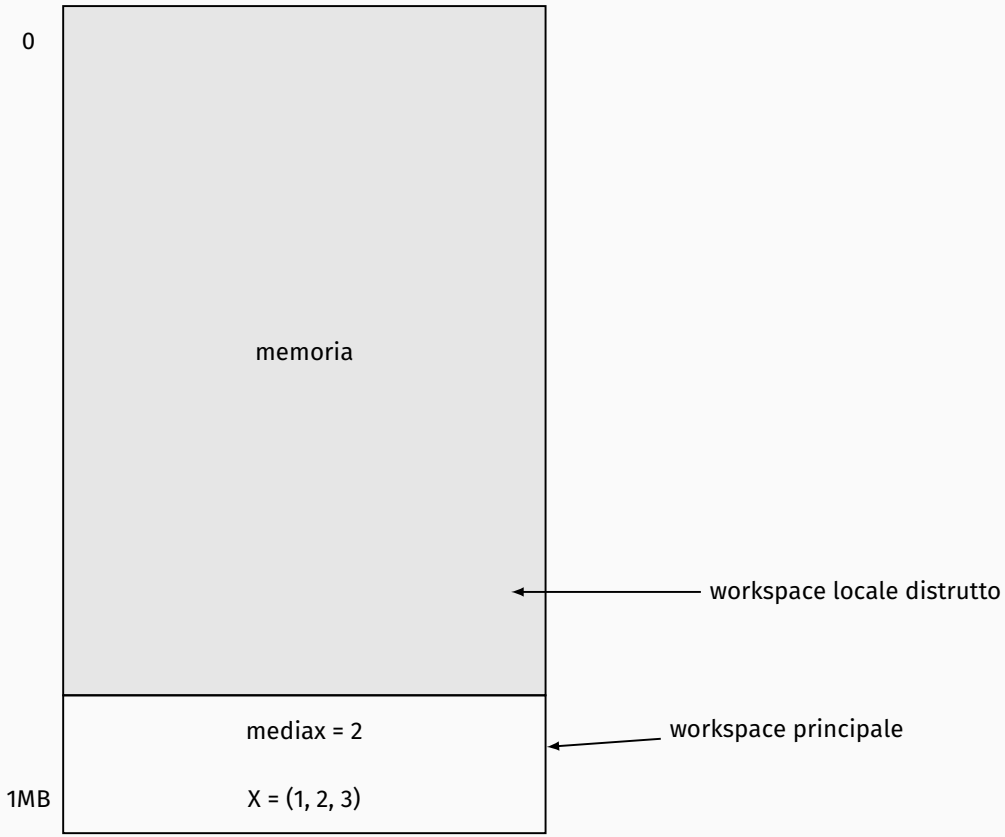
Stack (2)

Durante l'esecuzione di `calcolaMedia`



Stack (3)

Dopo l'esecuzione di `calcolaMedia`



Gestione dei file. Gestione delle matrici.

Files

- Contenitori di informazione permanenti
- Sono memorizzati su memoria di massa
- Possono continuare ad esistere indipendentemente dalla vita del programma che li ha creati
- Possono essere acceduti da più programmi

Files (2)

```
file: sequenza di bytes con nome
┌──┬──┬──┬──┬──┬──┬──┐       ┌──┬──┬──┐
 00 2a 5f ff aa 00 11    ...    22 11 21
└──┴──┴──┴──┴──┴──┴──┘       └──┴──┴──┘

primo         pos                    ultimo
byte          corrente               byte
```

Directory/Folders e Cartelle

- sono la stessa cosa;
- sono contenitori di files con nome
- possono contenere altre cartelle

Salvataggio matrice

```
a=rand(1,1000);
filename='pluto.txt'
[fid msg]=fopen(filename, 'w');
if(fid>0)
    cont=fwrite(fid,a,'float64');
    disp([num2str(cont) ' valori scritti...']);
    fclose(fid);
else
    disp(msg);
end
```

Caricamento matrice

```
filename='pippo.txt'
[fid msg]=fopen(filename, 'r');
if(fid>0)
    [vett cont]=fread(fid,[1 1000],'float64');
    disp([num2str(cont) ' valori letti...']);
    fclose(fid);
else
    disp(msg);
end
```

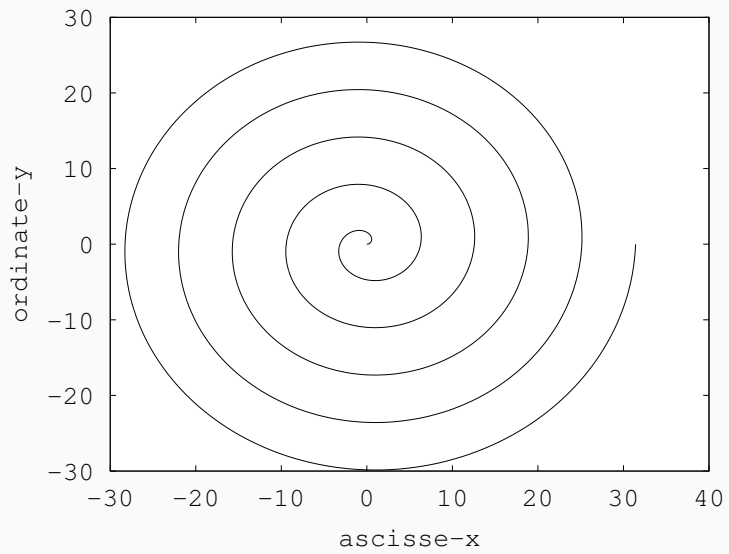
Tecniche di visualizzazione grafica.

Funzione `plot`

```
1  %% figure( 1, 'visible', 'off' );
2  x = 0:0.1:2*pi;
3  plot(x, sin(x))
4  title('Il mio primo grafico')
5  xlabel('etichetta asse x')
6  ylabel('etichetta asse y')
7  print -deps 'images/chart.eps' -F:20;
8  ans = 'images/chart.eps'
```

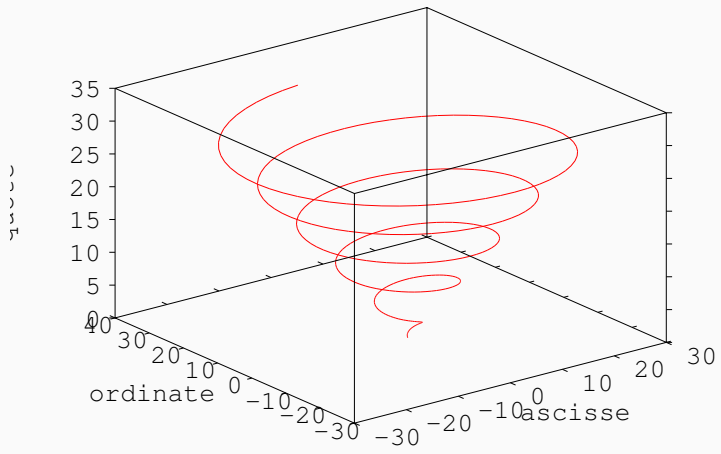
Funzione `plot` - diagrammi parametrici

```
1  %% figure( 1, 'visible', 'off' );
2  t=[0:pi/100:10*pi];
3  x=t .* cos(t);
4  y=t .* sin(t);
5  plot(x,y);
6  xlabel('ascisse-x');
7  ylabel('ordinate-y');
8  print -deps 'images/chartp.eps' -F:20;
9  ans = 'images/chartp.eps'
```



Funzione `plot3` - diagrammi 3d

```
1  %% figure( 1, 'visible', 'off' );
2  t = 0:0.1:10*pi;
3  plot3 (t.*sin(t), t.*cos(t), t);
4  xlabel('ascisse');
5  ylabel('ordinate');
6  zlabel('quote');
7  print -deps 'images/chart3d.eps' -F:20;
8  ans = 'images/chart3d.eps'
```



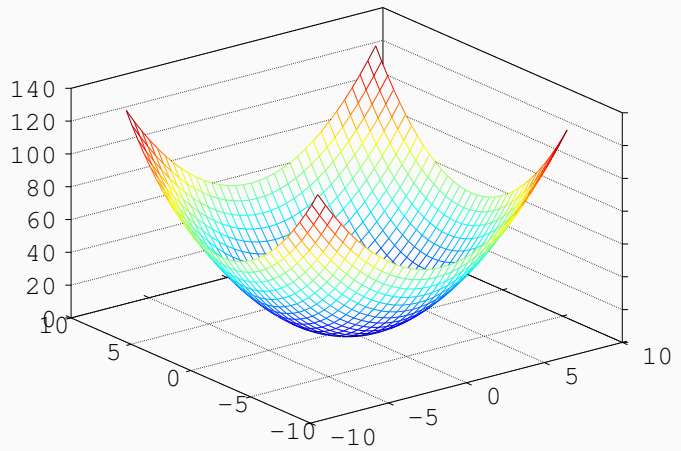
Superfici, costruzione range

```
1  x = 0:2;
2  y = 0:2;
3  [xx,yy] = meshgrid(x,y)
```

```
xx =

   0   1   2
   0   1   2
   0   1   2

yy =

   0   0   0
   1   1   1
   2   2   2
```

Superfici, plot

```
x = -8:0.5:8;
y = -8:0.5:8;
[xx,yy] = meshgrid(x,y)
zz = xx.^2 + yy.^2
mesh(xx,yy,zz)
print -deps 'images/chartsu.eps' -F:20;
ans = 'images/chartsu.eps'
```



Introduzione alla ricorsione.

Cosa è la ricorsione

- Un sottoprogramma P richiama se stesso (ricorsione diretta)
- Un sottoprogramma P richiama un'altro sottoprogramma Q che comporta un'altra chiamata a P (ricorsione indiretta)

A cosa serve

- E’ una tecnica di programmazione molto potente
- Permette di risolvere in maniera elegante problemi complessi

Esempio - Fattoriale

$$
\begin{aligned}
f(0) &= 1 \\
f(1) &= 1 \\
f(2) &= 2 * f(1) \\
f(3) &= 3 * f(2) \\
f(4) &= 4 * f(3)
\end{aligned}
$$

Esempio - Fattoriale

Definizione matematica

$$f(n) = \begin{cases} 1 & n = 0, 1 \\ n * f(n-1) & \text{altrimenti} \end{cases}$$

Possiamo descriverla in Matlab

```
1  function [f]=factRic(n)
2    if (n==0)
3        f = 1;
4    else
5        f = n * factRic(n-1);
6    end
7  end
```

Esempio - Sequenza di Fibonacci

Definizione matematica

$$f(n) = \begin{cases} 1 & n = 1, 2 \\ f(n-1) + f(n-2) & \text{altrimenti} \end{cases}$$

Versione Matlab

```
1  function [ fib ] = fibonacci(n)
2    if n==1 | n==2
3      fib = 1;
4    else
5      fib = fibonacci(n - 2) + fibonacci(n - 1);
6  end
```

Comportamento di una invocazione ricorsiva

Supponiamo di richiamare il fattoriale `f` di 3, e.g.:

```
1   x = factRic(3)
```