Matlab: Funzioni

Informatica ICA AA 17/18

Giacomo Boracchi

19 Ottobre 2017

giacomo.boracchi@polimi.it

Le Regole del Gioco

- Esami
  - Primo compitino: 14 Novembre 2017 (data definitiva solo quando appare sul sito)
  - Secondo compitino: ?
  - Recupero compitini: ?
  - 3 appelli regolari dopo Febbraio
- Nei compitini: valutazione 0-32. Si passa l'esame se si passano entrambi con un voto maggiore o uguale a 18. Il voto finale è la media dei due voti.
- Il primo compitino riguarda esclusivamente materiali del primo emisemestre. Il secondo compitino riguarda esclusivamente materiali del secondo emisemestre.

Le Regole del Gioco (cnt.)

- Recupero compitini: è possibile recuperare uno o due compitini presentandosi all'appello di recupero. E' possibile recuperare anche un compitino non insufficiente (per migliorare il voto). Quando uno studente si presenta all'appello di recupero vede automaticamente cancellato il voto che intende recuperare.
- L'esame orale non è previsto se non a discrezione del docente
- Studenti con OFA in matematica (o totale) non possono sostenere le prove prima di aver passato l'OFA

A cosa servono le funzioni?

```
x = input('inserisci x: ');
fx = 1
for ii = 1 : x
  fx = fx * ii;
end
if (fx > 220)
  y = input('inserisci y: ');
  fy = 1
  for ii = 1 : y
    fy = fy * ii;
  end
end
```

A cosa servono le funzioni?

```
x = input('inserisci x: ');

fx = 1
for ii = 1 : x
  fx = fx * ii;
end

if (fx > 220)
  y = input('inserisci y: ');
  fy = 1
  for ii = 1 : y
    fy = fy * ii;
  end
end
```

Entrambi i frammenti di codice eseguono il calcolo del fattoriale

A cosa servono le funzioni?

- **Riusabilità**
  - Scrivo una sola volta codice utilizzato spesso
  - Modifiche e correzioni sono gestibili facilmente
  - Lo stesso codice viene facilmente richiamato in diversi programmi
- **Leggibilità**
  - Incapsulo porzioni di codice complesso, il programmatore non deve entrare nei dettagli
  - Aumento il livello di astrazione dei miei programmi
- **Flessibilià**
  - Posso aggiungere funzionalità non presenti nelle funzioni di libreria

Usiamo uno script file?

Uno script file può essere usato per incapsulare porzioni di codice riusabili in futuro

```
x = input('inserisci x: ');
fx=1
for ii=1:x
  fx = fx*ii
end
if (fx>220)
  y = input('inserisci y: ');
  fy=1
  for ii=1:y
    fy = fy*ii
  end
end
```

```
f=1
for ii=1:n
  f = f*ii
end
```

fattoriale.m

Limiti degli script-files

Problemi:

- Come fornisco l'input allo script?
- Dove recupero l'output?

Gli script utilizzano le variabili del workspace:

```
x = input('inserisci x: ');
n=x
fattoriale
fx=f
if (fx>220)
  y = input('inserisci y: ');
  n=y
  fattoriale
  fy=f
end
```

```
f=1
for ii=1:n
  f = f*ii
end
```

fattoriale.m

Limiti degli script-files

Problemi:

- Come fornisco l'input allo script?
- Dove recupero l'output?

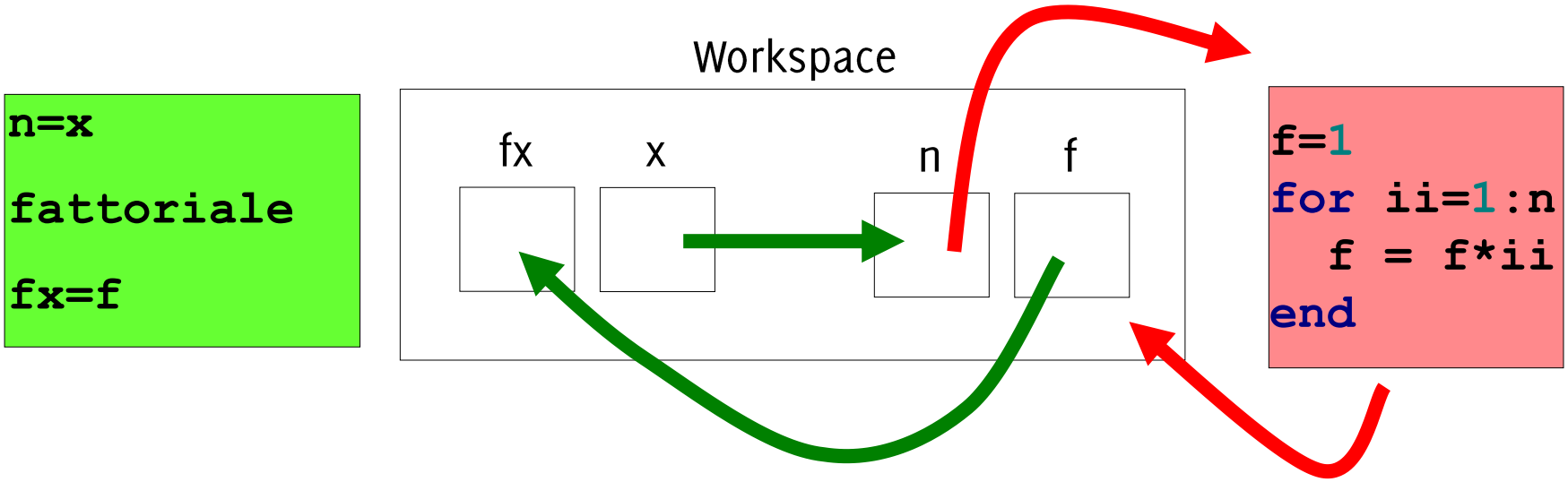
Gli script utilizzano le variabili del workspace:

```
x = input('inserisci x: ');
n=x          <- Prepara l'input in n
fattoriale   <- chiama lo script
fx=f         <- Salva il risultato in f
if (fx>220)
  y = input('inserisci y: ');
  n=y        <- Prepara l'input
  fattoriale <- chiama lo script
  fy=f       <- Salva il risultato in f
end
```

```
f=1
for ii=1:n
  f = f*ii
end
```

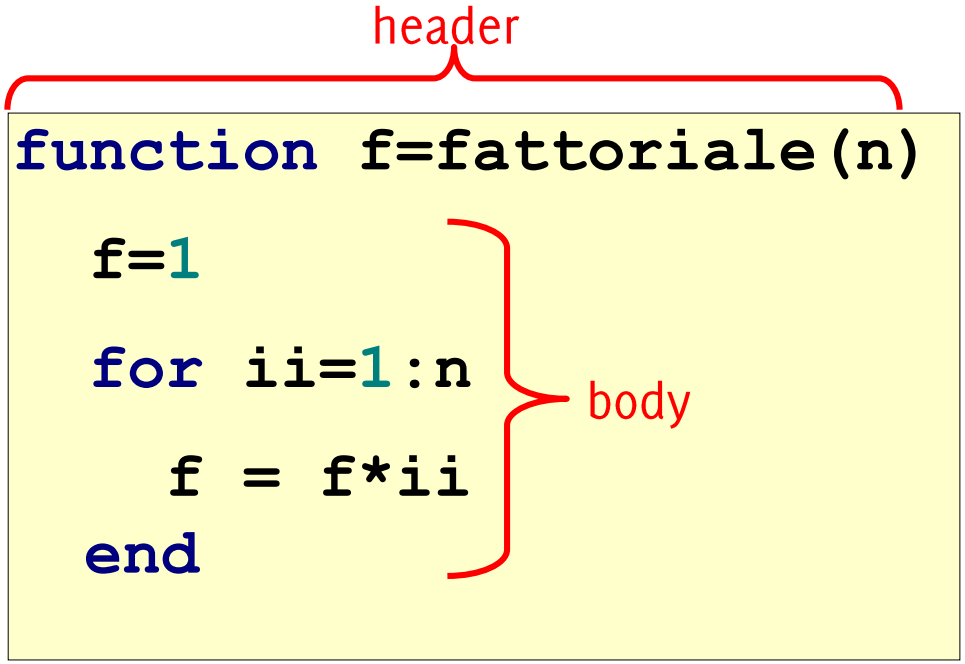
fattoriale.m

Limiti degli script-files (2)



- Questo meccanismo ha molti svantaggi:
  - poco leggibile
  - richiede molte istruzioni
  - poco sicuro
- Tutte le variabili sono nello stesso workspace (fattoriale.m può modificare tutte le variabili del workspace)
- Le funzioni non hanno questi problemi

Le funzioni



n è **l'argomento** della funzione (serve a fornire l'input)

f è il **valore di ritorno** della funzione (serve a fornire l'output)

- La testata (header) inizia con la parola chiave function e definisce:
  - nome della funzione
  - argomenti (input)
  - valore di ritorno (output)
- Il corpo definisce le istruzioni da eseguire quando la funzione viene chiamata
  - Utilizza gli argomenti e assegna il valore di ritorno

Le funzioni (2)

- Una funzione può avere più argomenti separati da virgola:

```
function f(x,y)
```

- Nel caso sia necessario ritornare più valori, definiamo l'header affiancando più variabili in output usando la stessa notazione degli array (l'output non deve necessariamente essere omogeneo):

```
function  [v1,v2,…] = f(x,y)
```

- Esempio:

```
function [s, p] = sumProd(a, b)
  s = a + b;
  p = a * b;
```

Definizione dell'header di una funzione

- La sintassi per definire l'header di funzione è

```
function [out1, .., outM] = nomeFunzione(in1, .., inN)
```

- **Gli argomenti (parametri in ingresso) `in1,..,inN`** vanno elencate tra parentesi tonde e seguono il nome della funzione
- **I valori ritornati (parametri in uscita) `out1,..,outN`** vanno elencate tra parentesi quadre e seguono la keyword **`function`**.

- **NB:** la notazione **`[out1, .., outM]`** per le variabili in uscita di una funzione è la stessa dell'operatore CAT orizzontale. Però qui ha un altro significato perché **`out1, .., outM`** possono avere dimensioni e tipi non consistenti!

Invocazione

- Una funzione può essere invocata in un programma attraverso il suo nome, seguito dagli argomenti fra parentesi rotonde
- La funzione viene quindi eseguita e il suo valore di ritorno viene calcolato.
- Esempio

```
x = input('inserisci x:');
fx = fattoriale(x);
if (fx>220)
   y = input('inserisci y: ');
   fy = fattoriale(y);
end
```

Invocazione

Invocazione

```
function f=fattoriale(n)
  f=1
  for ii=1:n
    f = f*ii
  end
```

I Parametri

- **Definizioni:**
  - I **parametri formali** sono le variabili usate come **argomenti** e **valori** di **ritorno** **nella definizione** della funzione
  - I **parametri attuali** sono i valori (o le variabili) usati come **argomenti** e come **valori** di **ritorno** **nell'invocazione** della funzione

```
function f=fattoriale(n)
  f = 1;
  for ii=1:n
    f = f*ii;
  end
>> fat5 = fattoriale(5) %Invocazione
fat5 =
   120
```

f ed n sono parametri formali

fx e 5 sono parametri attuali

I Parametri (2)

- **Qualsiasi tipo di parametri** è ammesso (scalari, vettori, matrici, strutture, ecc.)
- I **parametri attuali** vengono **associati a quelli formali** in **base alla posizione**: il primo parametro attuale viene associato al primo formale, il secondo parametro attuale al secondo parametro formale, ecc.
- Esempio

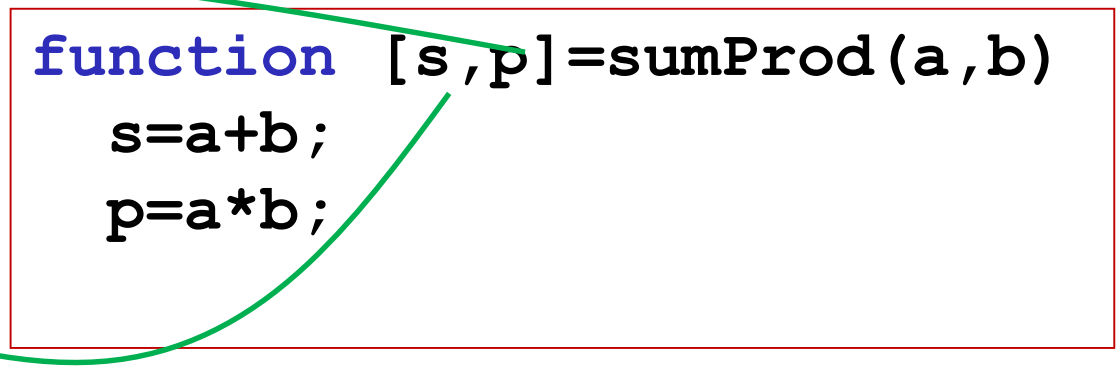
```
>> [x,y]=sumProd(4,5)
```

```
function [s,p]=sumProd(a,b)
  s=a+b;
  p=a*b;
```

I Parametri (2)

- **Qualsiasi tipo di parametri** è ammesso (scalari, vettori, matrici, strutture, ecc.)
- I **parametri attuali** vengono **associati a quelli formali** in **base alla posizione**: il primo parametro attuale viene associato al primo formale, il secondo parametro attuale al secondo parametro formale, ecc.
- Esempio

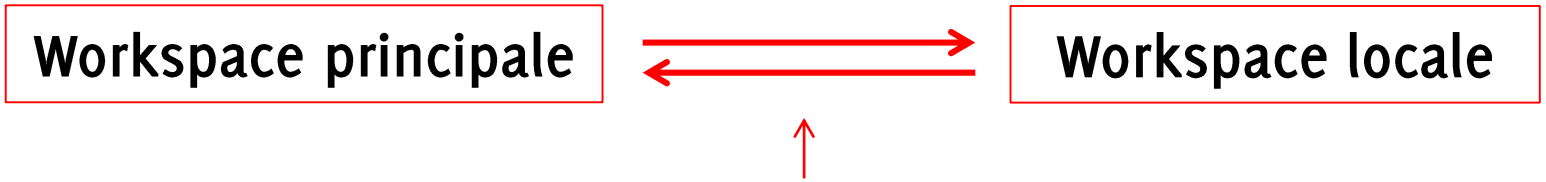
```
>> [x,y]=sumProd(4,5)
```



Esecuzione di una funzione

Quando una funzione viene eseguita, viene creato un workspace "locale" in cui vengono memorizzate tutte le variabili usate nella funzioni inclusi i parametri formali.

- All'interno delle funzioni non si può accedere al workspace "principale" (nessun conflitto coi nomi delle variabili)
- Al termine dell'esecuzione della funzione, il workspace "locale" viene distrutto!



Le comunicazioni tra i workspace avvengono solamente mediante copia dei valori dei parametri in ingresso ed in uscita

Riepilogando: Esecuzione di una funzione (2)

Quando viene invocata una funzione:

1. Vengono **calcolati** i valori dei **parametri attuali** di ingresso
2. Viene **creato un workspace "locale"** per la funzione
3. I **valori** dei **parametri attuali** di ingresso vengono **copiati** nei **parametri formali** all'interno del **workspace "locale"**
   - Il workspace locale ora contiene solamente i parametri formali con assegnati i valori dei parametri attuali
4. Viene **eseguito il corpo** della **funzione**
5. Vengono **copiati i valori di ritorno dai parametri formali nel workspace "locale" al workspace "principale"** nei corrispondenti parametri attuali
6. Il workspace "locale" viene **distrutto**

Esecuzione di una funzione: esempio

```
(1) >> x=3;
(2) >> w=2;
(3) >> r = funz(4);
```

```
function y = funz(x)
  y = 2*x;      %(1')
  x = 0;        %(2')
  z = 4;        %(3')
```

W "principale" dopo (2)

x=3
w=2

W "locale" dopo(1')

x=4
y=8

W "principale" dopo (3)

x=3
w=2
r= 8

W "locale" dopo(3')

x=0
y=8
z=4

~~W "locale" dopo (3)~~

Esecuzione di una funzione: esempio

```
(1) >> x=3;
(2) >> w=2;
(3) >> r = funz(4);
```

```
function y = funz(x)
  y = 2*x;      %(1')
  x = 0;        %(2')
  z = 4;        %(3')
  x = w - 1;    %(4')
```

W "principale" dopo (2)

x=3
w=2

W "locale" dopo(1')

x=4
y=8

W "locale" prima (4')

x=0
y=8
z=4
w=? → errore

W "principale" dopo (3)

x=3
w=2

W "locale" dopo(3')

x=0
y=8
z=4

~~W "locale" dopo (3)~~

I Parametri (3)

- In linea di massima, **il numero di parametri attuali** all'invocazione della funzione deve essere identico al numero di **parametri formali in ingresso**
- Il **vincolo vale per i parametri in ingresso**, anche se è possibile trattare i parametri formali nella funzione per gestire questi casi

I Parametri (3)

- In linea di massima, **il numero di parametri attuali** all'invocazione della funzione deve essere identico al numero di **parametri formali in ingresso**
- Il **vincolo vale per i parametri in ingresso**, anche se è possibile trattare i parametri formali nella funzione per gestire questi casi
- Il **vincolo non vale per i parametri in uscita**: verranno assegnati solamente i parametri attuali specificati.
  - Ad esempio `s = sommaProd(5,2)` il valore della somma viene assegnato a `s` ma non il valore del prodotto (anche se la funzione lo calcola)

Esempio

Scrivere una funzione che prende in ingresso tre numeri e resituisce il massimo ed il minimo.

Esempio

```
function [minore, maggiore] = minmax(a,b,c)
maggiore = a;
if maggiore < b
    maggiore = b;
end
if maggiore < c
    maggiore = c;
end

minore = a;
if minore > b
    minore = b;
end
if minore > c
    minore = c;
end
```

Note sui Parametri in Uscita

- **I parametri formali** dei valori **di ritorno** devono essere **sempre definiti** (eventualmente possono essere vuoti)
- Questa funzione da errori quando il vettore inserito contiene solamente elementi negativi

```
function [positivi, media] = mediaPositivi(vett)
somma = 0; cnt = 0;
positivi = [];
for ii = 1 : length(vett)
    if vett(ii) > 0
        positivi = [positivi, vett(ii)];
        somma = somma + vett(ii);
        cnt = cnt + 1;
    end
end
if cnt > 0
    media = somma / cnt;
end
```

Note sui Parametri in Uscita

- **I parametri formali** dei valori **di ritorno** devono essere **sempre definiti** (eventualmente possono essere vuoti)
- Questa funzione da errori quando il vettore inserito contiene solamente elementi negativi

```
function [positivi, media] = mediaPositivi(vett)
somma = 0; cnt = 0;
positivi = [];
for ii = 1 : length(vett)
    if vett(ii) > 0
        positivi = [positivi, vett(ii)];
        somma = somma + vett(ii);
        cnt = cnt + 1;
    end
end
if cnt > 0
    media = somma / cnt;
end
```

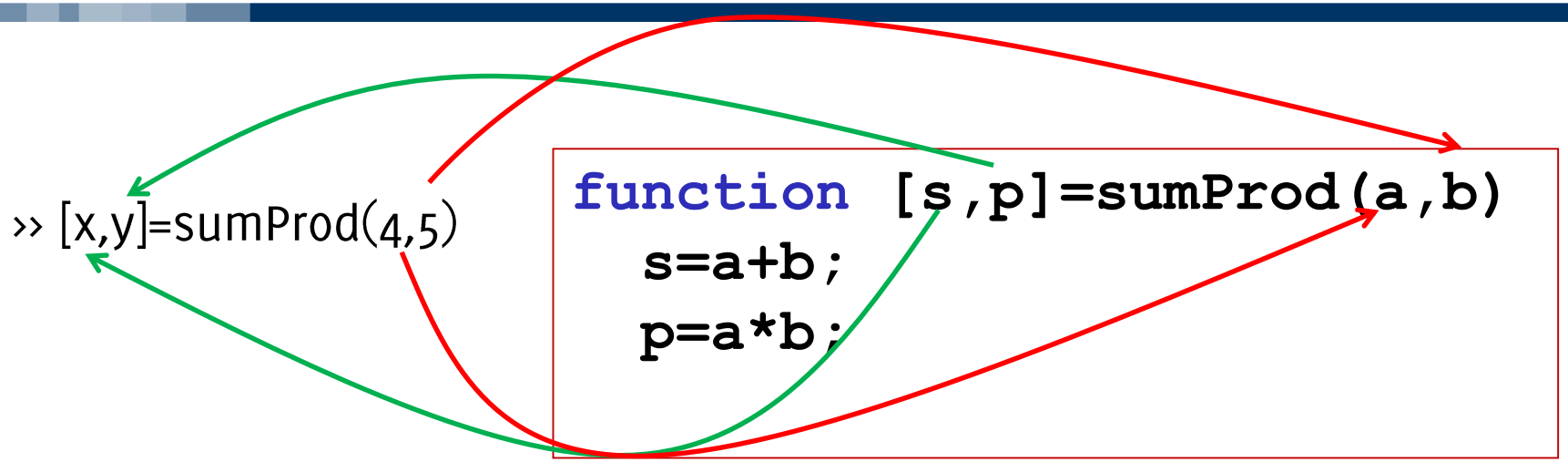
```
>> [a,b] = mediaPositivi(-[1 : 10])
Error in mediaPositivi

Output argument "media" (and maybe
others) not assigned during call to
mediaPositivi
```

Note sui Parametri in Uscita

- **I parametri formali** dei valori **di ritorno** devono essere **sempre definiti** (eventualmente possono essere vuoti)
- Questa funzione da errori quando il vettore inserito contiene solamente elementi negativi

```
function [positivi, media] = mediaPositivi(vett)
somma = 0; cnt = 0;
positivi = [];
for ii = 1 : length(vett)
    if vett(ii) > 0
        positivi = [positivi, vett(ii)];
        somma = somma + vett(ii);
        cnt = cnt + 1;
    end
end
if cnt > 0
    media = somma / cnt;
else
    media =  [];
end
```

Note sull'output



- È però possibile invocare la funzione senza specificare due parametri in uscita,
  - es x = sumProd(4,5). In tal caso solamente il primo output viene assegnato ad x
- L'invocazione sumProd(4,5) associa alla variabile ans il primo argomento restituito da sumProd
- Per riceve solo il secondo output uso ~ come se fosse una variabile da non considerare [~,y] = sumProd(4,5)

File funzione

- Come nel caso degli script le funzioni possono essere scritti in file di testo sorgenti
  - Devono avere estensione .m
  - Devono avere lo stesso nome della funzione
  - Devono iniziare con la parola chiave **function**
- Attenzione a non "ridefinire" funzioni esistenti
  - exist('nomeFunzione') → o se la funzione non esiste
- Se commentate, le prime righe della funzione rappresentano l'help e vengono visualizzate quando si scrive: help nomeFunzione

Esempio

Scrivere una funzione *contoAllaRovescia* che prende in ingresso un intero (che esprime i secondi) ed esegue il conto alla rovescia. Al termine viene emesso un suono e mandato un messaggio a schermo.

Esercizio

Implementare la funzione trasposizione per le matrici

Esercizio

Implementare la funzione trasposizione per le matrici

```
function [t]=trasposta(m)
    [r,c]=size(m);
    for ii=1:r
        for j=1:c
            t(j,ii)=m(ii,j);
        end;
    end
```

```
>> m =[1,2,3,4
       5,6,7,8
       9,10,11,12]
m =
     1     2     3     4
     5     6     7     8
     9    10    11    12

>> trasposta(m)
ans =
     1     5     9
     2     6    10
     3     7    11
     4     8    12
```

Esercizio

Scrivere una funzione che prende in ingresso due coefficienti m, q ed un vettore di punti xx e restituisce il vettore yy dei punti che stanno sulla retta $y = mx + q$ in corrispondenza a xx

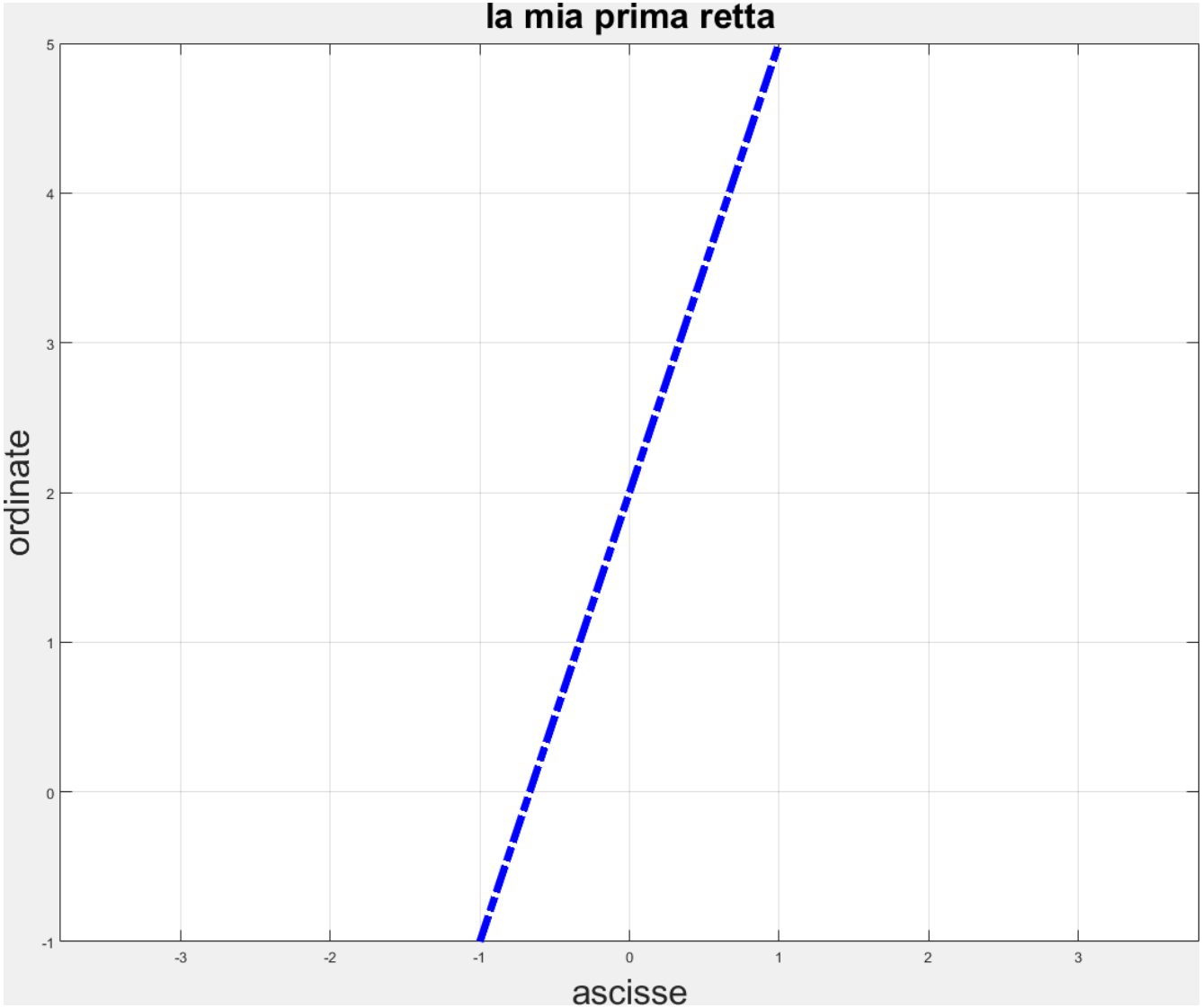
Esercizio

Scrivere una funzione che prende in ingresso due coefficienti m, q ed un vettore di punti xx e restituisce il vettore yy dei punti che stanno sulla retta $y = mx + q$ in corrispondenza a xx

```
function [yy] = retta(m, q, xx)
    yy = m * xx + q;

% for ii = 1 : length(xx)
%     yy(ii) = m * xx(ii) + q;
% end
```

Esempi di script per invocare la funzione

```
x = [-1 : 0.1 : 1];
% invoco la funzione per plottare y = 3x +2
y = retta(3,2,x)
figure
plot(x,y, 'b*') % disegno con le stelline
axis equal % assi della stessa dimensione
plot(x,y, 'b-'), %disegno con una retta
grid on % aggingo aggiungo la griglia
plot(x,y, 'b-', 'LineWidth', 3), axis equal, grid on
plot(x,y, 'b--', 'LineWidth', 5), axis equal, grid on
plot(x,y, 'b-.', 'LineWidth', 5), axis equal, grid on
title('la mia prima retta', 'FontSize', 24)
xlabel('ascisse', 'FontSize', 24)
ylabel('ordinate', 'FontSize', 24)
```


la mia prima retta
ordinate
ascisse
5
4
3
2
1
0
-1
-3
-2
-1
0
1
2
3

Esercizio

- Scrivere una funzione che calcola la sequenza di Fibonacci della lunghezza richiesta
- La successione di Fibonacci è definita così:
  - $F(0) = 0$
  - $F(1) = 1$
  - $F(n) = F(n-1) + F(n-2), n > 1$

Esercizio

- Scrivere una funzione che calcola la sequenza di Fibonacci della lunghezza richiesta
- La successione di Fibonacci è definita così:
  - $F(0) = 0$
  - $F(1) = 1$
  - $F(n) = F(n-1) + F(n-2), n > 1$

```
function F = fibonacci(n)
% function F = fibonacci(n)
%
% resituisce un vettore (F) contenente i primi n numeri di fibonacci
F = [0 , 1];
for indx  = [3 : 1 : n]
   F(indx) = F(indx - 1) + F(indx - 2);
end
```

Esercizio

```
%% scrivere un programma che  chiede all'utente di inserire un
% numero positivo  (nel caso in cui il numero non è positivo
ripetere  inserimento)
%
% verificare se il numero è perfetto
% in caso contrario dice se è  abbondante o difettivo.
% Dopo di che richiede un altro numero e controlla se
% i due numeri sono amici

% un numero è perfetto se corrisponde alla somma
% dei suoi divisori, escluso se stesso
% abbondante se è › della somma dei suoi divisori
% altrimenti difettivo
%
% a,b sono amici se la somma dei divisori di a= b e viceversa
```

Header delle funzioni

```
function [n] = inserisciInteroPositivo()
% richiede all'utente di inserire un numero e
contiuna finchè questo non è intero e positivo
restituisce l'intero positivo inserito


function [perf, abb] = calcolaSePerfetto(n)
% perf = 1 se n è perfetto, abb = 1 se n NON perfetto
e abbondante, abb = 0 se n NON perfetto e difettivo


function [s] = calcolaSommaDivisori(n)
% s somma dei divisori di n (escluso n)


function [res] = calcolaSeAmici(a, b)
% res = 1 se a e b sono amici, 0 altrimenti
```

Funzioni Built In

Alcune funzioni built in

Funzione	Significato
zeros (n)	Restituisce una matrice n×n di zeri
zeros (m,n)	Restituisce una matrice m×n di zeri
zeros (size(arr))	Restituisce una matrice di zeri della stessa dimensione di arr
ones(n)	Restituisce una matrice n×n di uno
ones(m,n)	Restituisce una matrice m×n di uno
ones(size(arr))	Restituisce una matrice di uno della stessa dimensione di arr
eye(n)	Restituisce la matrice identità n×n
eye(m,n)	Restituisce la matrice identità m×n
length(arr)	Ritorna la dimensione più lunga del vettore
size(arr)	Ritorna un vettore [r c] con il numero r di righe e c di colonne della matrice; se **`arr`** ha più dimensioni ritorna array con numero elementi per ogni dimensione

Funzioni predefinite

- Esempi
  - a = zeros(2); $\longrightarrow \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$
  - b = zeros(2,3); $\longrightarrow \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$
  - c = [1 2; 3 4];
  - d = zeros(size(c)); $\longrightarrow \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$

Funzioni Aritmetiche

Funzione	Scopo
ceil(x)	approssima x all'intero immediatamente maggiore
floor(x)	approssima x all'intero immediatamente minore
fix(x)	approssima x all'intero più vicino verso lo zero
[m,pos] = max(x)	se x è un vettore, ritorna il valore massimo in x e, opzionalmente, la collocazione di questo valore in x; se x è matrice, ritorna il vettore dei massimi delle sue colonne
[m,pos] = min(x)	se x è un vettore, ritorna il valore minimo nel vettore x e, opzionalmente, la collocazione di questo valore nel vettore; se x è matrice, ritorna il vettore dei minimi delle sue colonne
mean(x)	se x è un vettore ritorna uno scalare uguale alla media dei valori di x; se x è una matrice, ritorna il vettore contentente le medie dei vettori colonna di x;
mod(m,n)	mod(m,n) è m – q .* n dove q = floor(m ./ n) se n ~= 0
round(x)	approssima x all'intero più vicino
rand(N)	Restituisce una matrice di NxN numeri casuali con distribuzione uniforme tra 0,1

Quindi questa funzione….

```
function [minore, maggiore] = minmax(a,b,c)
minore = a;
maggiore = a;
if minore < b
    minore = b;
end
if maggiore > b
    maggiore = b;
end

if minore < c
    minore = c;
end
if maggiore > c
    maggiore = c;
end
```

Si poteva scrivere così

```
function [minore, maggiore] = minmax(a,b,c)
  minore = min ([a,b,c]);
  maggiore = max([a,b,c]);
```

funzioni `min` (e anche `max`) applicate a vettori e matrici

```
>> b = [4 7 2 6 5]
b =  4     7     2     6     5
>> min(b)
ans = 2
>> [x y]=min(b)
x =  2
y =  3
>>
```

(con un solo risultato) dà il valore del minimo

con due risultati dà anche la posizione del minimo

```
>> a = [24 28 21; 32 25 27; 30 33 31; 22 29 26]
a = 24    28    21
    32    25    27
    30    33    31
    22    29    26
>> min(a)
ans = 22    25    21

>> [x y]=min(a)
x =  22    25    21
y =  4     2     1
>>
```

per una matrice dà vettore dei minimi nelle colonne

per una matrice, con due risultati dà due vettori dei valori minimi nelle colonne e della loro posizione (riga)

Funzioni Aritmetiche

- Prod(vettore) calcola il prodotto di tutti gli elementi di vettore
- Esempio: alternativa «alla Matlab» per il calcolo del fattoriale

```
function k = fattoriale2(n)
k = prod([n : -1 : 1]);
```

Altre funzioni importanti

- **`length(v)`**, restituisce la lunghezza del vettore
- **`size(A)`** restituisce un vettore contenente le dimensioni dell'array A (come si vedono da whos)
- **`size(A, dim)`** restituisce il numero di elementi di A lungo la dimensione dim
- ATTENZIONE che **`length`** su matrici restituisce la dimensione maggiore! O meglio restituisce **`max(size(A))`**

Funzioni Logiche

Nome	Elemento restituito
all(x)	un vettore riga, con lo stesso numero di colonne della matrice x, che contiene 1, se la corrispondente colonna di x contiene tutti elementi non nulli, o 0 altrimenti. Se x è un vettore restituisce 0 o 1 con lo stesso criterio.
any(x)	un vettore riga, con lo stesso numero di colonne della matarice x, che contiene 1, se la corrispondente colonna di x contiene almeno un elemento non nullo, o 0 altrimenti. Se x è un vettore restituisce 0 o 1 con lo stesso criterio.
isinf(x)	un array delle stesse dimensioni di x con 1 dove gli elementi di x sono 'inf', 0 altrove
isempty(x)	1 se x è vuoto, 0 altrimenti
isnan(x)	un array delle stesse dimensioni di x con 1 dove gli elementi di x sono 'NaN', 0 altrove
finite(x)	un array delle stesse dimensioni di x, con 1 dove gli elementi di x sono finiti, 0 altrove
ischar(x)	1 se x è di tipo char, 0 altrimenti
isnumeric(x)	1 se x è di tipo double, 0 altrimenti
isreal(x)	1 se x ha solo elementi con parte immaginaria nulla, 0 altrimenti

Esempio

Scrivere una funzione *cerca* che controlla se un elemento **x** appartiene ad un vettore **vett** e, in caso affermativo, ne restituisce la posizione

Esercizio

Scrivere una funzione *closestVal* che prende in ingresso un vettore **vett** ed uno scalare **x** e restituisce il valore di **vett** più vicino ad **x**

Esercizio

Scrivere un programma che richiede in ingresso due parole e determina se l'una è l'anagramma dell'altra

- Uno script si occupa dell'acquisizione delle parole.
- Implementare una funzione per creare l'istogramma delle parole.
- Eseguire nello script il confronto tra istogrammi e visualizzare a schermo il risultato.

Funzioni Built in per Visualizzazione

Funzioni Grafiche

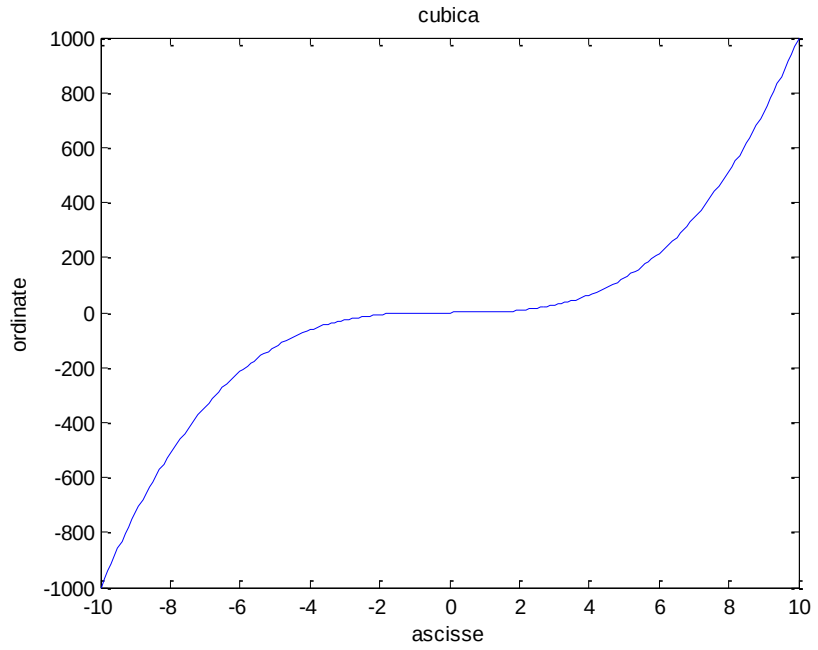
Funzione	Scopo
figure(figNumber)	apre una figura identificata dall'handle figNumber . Se non presente definisce l'handle in maniera incrementale
hold	+ on / off definisce se tenere o cancellare il grafico attualmente presente nella figura alla prossima operazione di visualizzazione sulla figura.
plot(x,y)	disegna in un riferimento cartesiano 2D le coppie di punti identificati da (x(1),y(1))… (x(end) , y(end)). x ed y devono avere la stessa lunghezza
plot3(x,y,z)	disegna in un riferimento cartesiano 3D le coppie di punti identificati da (x(1),y(1),z(1))… (x(end) , y(end), z(end). x,y e z devono avere la stessa lunghezza
plot(x,y, frmStr)	frmStr specifica il marker ed il colore usato nella visualizzazione dei punti
imagesc(A)	Visualizza la matrice A come un'immagine in colormap di default. Ogni pixel viene ridimensionato per migliorare la visualizzazione
imshow(A)	visualizza un'immagine A in scale di grigio (se A è di dimensione 2) o a colori nello spazio RGB (se A è di dimensione 3)
legend(titles)	Visualizza la legenda, usando le stringhe in titles

Diagrammi a due dimensioni

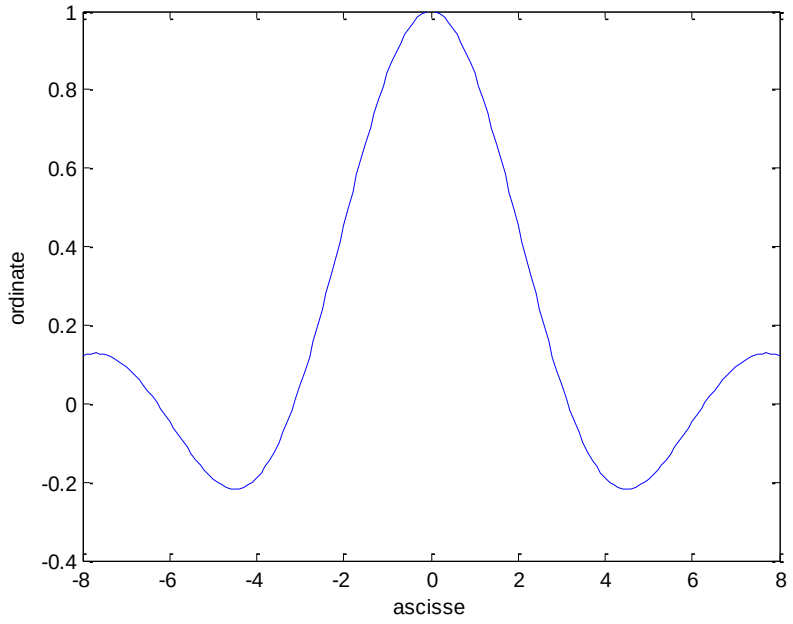
- La funzione `plot(x,y)` disegna il **diagramma** cartesiano dei punti che hanno valori delle ascisse nel vettore x, delle ordinate nel vettore y
- Il diagramma è l'insieme di **coppie di punti** [x(1), y(1)],…, [x(end),y(end)] rappresentanti le coordinate dei punti del piano cartesiano
  - La funzione `plot` congiunge i punti con una linea, per dare continuità al grafico.
- In `plot(x,y)`, `x` e `y` devono essere **due vettori aventi le stesse dimensioni**
- E' possibile specificare diversi elementi grafici (help plot per una lista delle opzioni)
- Le funzioni `xlabel` visualizzano una stringa come nome asse ascisse, `ylabel` per ordinate, `title` per il titolo

Diagrammi a due dimensioni: esempi

```
>> x = -10:0.1:10;
>> y=x.^3;
>> plot(x,y);
>> xlabel('ascisse');
>> ylabel('ordinate');
>> title('cubica');
```




```
>> x=[-8:0.1:8];
>> y= sin (x) ./ x;
>> plot(x, y);
>> xlabel('ascisse');
>> ylabel('ordinate');
```



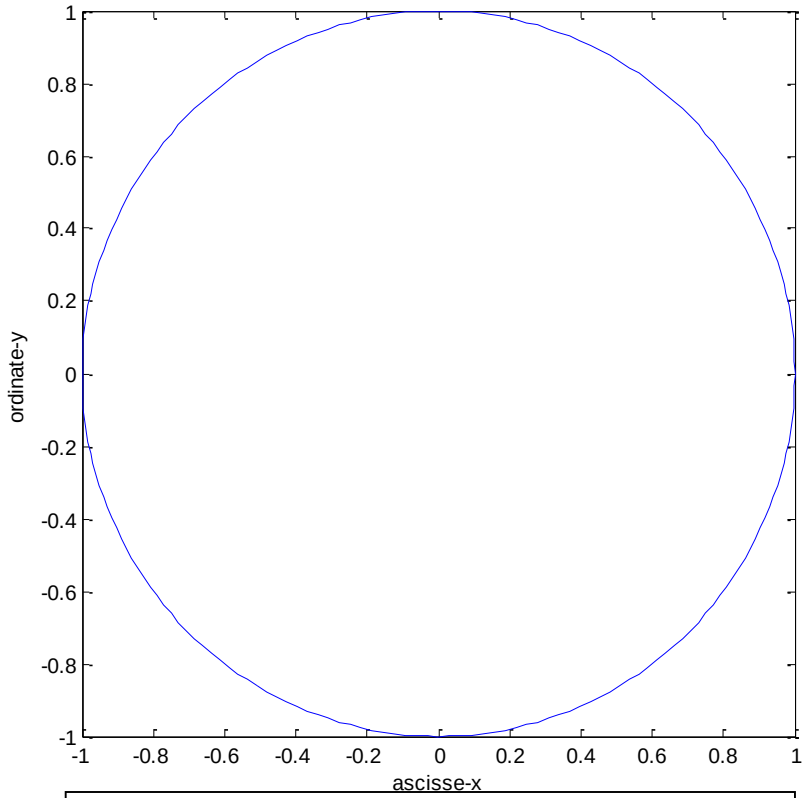
Diagrammi a due dimensioni: ancora esempi

- Un diagramma è semplicemente una sequenza ordinata di punti, di coppie di coordinate cartesiane
- In `plot(x,y)` non necessariamente `x` contiene valori equispaziati e `y` non è necessariamente funzione di `x`. Sia `x` che `y` possono essere, ad esempio, funzioni di qualche altro parametro.
- Che diagrammi disegnano i seguenti esempi?

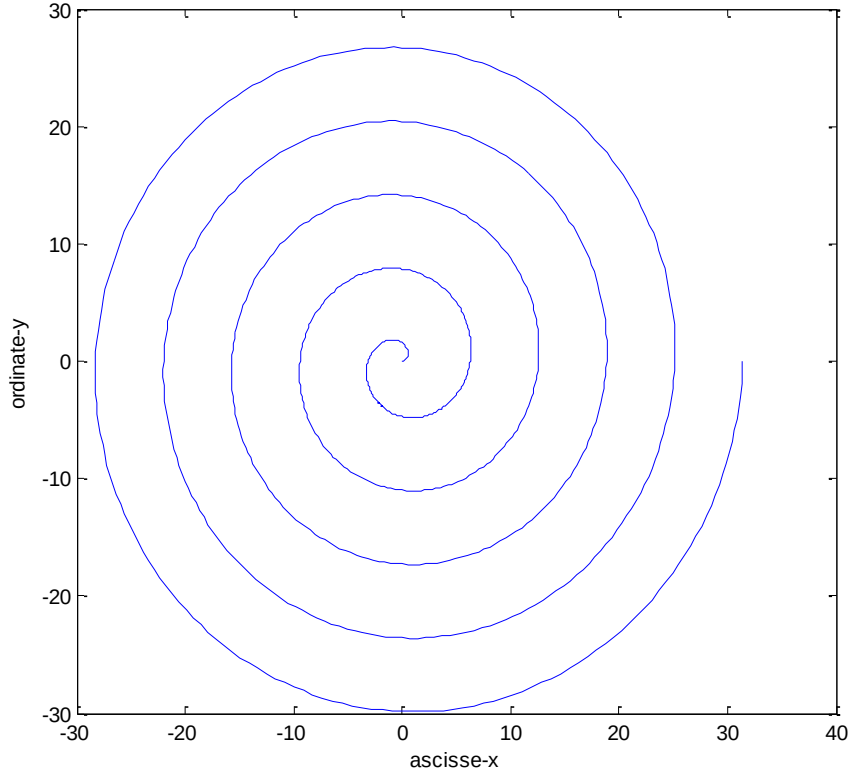
```
>> t=[0:pi/100:2*pi];
>> x=cos(t);
>> y=sin(t);
>> plot(x,y);
>> xlabel('ascisse-x');
>> ylabel('ordinate-y');
```

```
>> t=[0:pi/100:10*pi];
>> x=t .* cos(t);
>> y=t .* sin(t);
>> plot(x,y);
>> xlabel('ascisse-x');
>> ylabel('ordinate-y');
```

Esempi




```
>> t=[0:pi/100:2*pi];
>> x=cos(t);
>> y=sin(t);
>> plot(x,y);
>> xlabel('ascisse-x');
>> ylabel('ordinate-y');
```




```
>> t=[0:pi/100:10*pi];
>> x=t .* cos(t);
>> y=t .* sin(t);
>> plot(x,y);
>> xlabel('ascisse-x');
>> ylabel('ordinate-y');
```

Esempi

- Definire una funzione *samplePolynomial* che prende in ingresso
  - un vettore di coefficienti C
  - un vettore che definisce un intervallo [a,b]

  e restituisce un due vettori di 100 punti xx ed yy contenente i punti della del polinomio

  $$y = C(1)x^{n-1} + C(2)x^{n-2} + \ldots + C(n-1)x^1 + C(n)$$

- Utilizzare *samplePolynomial* per calcolare i punti delle seguenti curve (in un intervallo [-10 10]) e visualizzarlo:

  $$y = x - 1;$$
  $$y = 2x^2 + x - 12;$$
  $$y = -0.1x^3 + 2x^2 - 10x - 12$$

- visualizzare, per ogni valore di x, la curva maggiore

Soluzione

```
function [xx, yy] = samplePolynomial(polyCoeff, interval)
% determina 100 nell'intervallo interval
% appartenenti al polinomio avente coefficienti polyCoeff

% per essere certi che a <= b
a = min(interval);
b = max(interval);

xx = [a  : (b-a) / 100 : b];
% oppure xx = linspace(a , b, 100)
yy  = zeros(size(xx));

for ii = 1 :  1 : length(polyCoeff)
    yy = yy + polyCoeff(ii) * xx.^(length(polyCoeff) - ii);
end
```

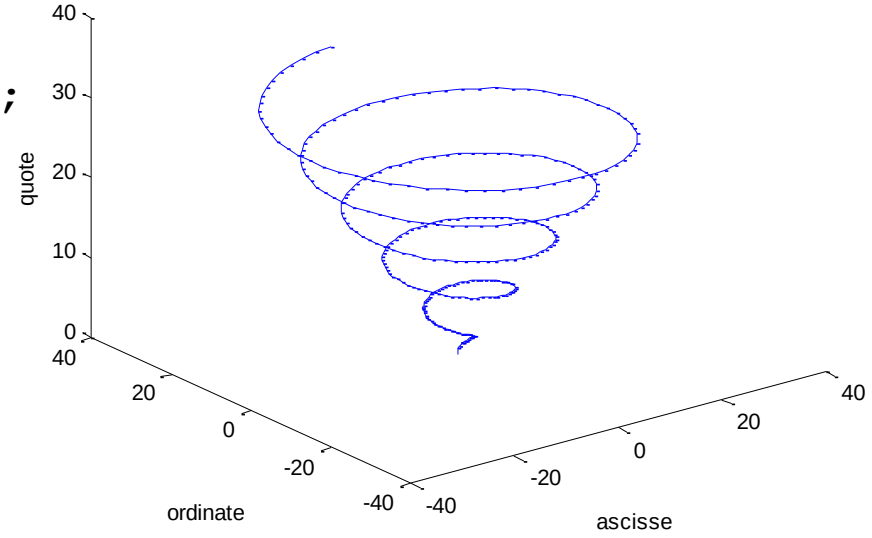
```
interval = [-10 , 10];
rettaCoeffs = [1 ,-1];
parabolaCoeffs = [ 2 ,  1 , -12] ;
cubicaCoeffs = [-0.1 , 2 ,-10 , -12];

% calcola i valori dei polinomi
[rx,ry] = samplePolynomial(rettaCoeffs , interval);
[px,py] = samplePolynomial(parabolaCoeffs , interval);
[cx,cy] = samplePolynomial(cubicaCoeffs, interval);


% determina i punti ad ordinata maggiore per ogni ascissa
indx_r = find(ry > py & ry > cy);
indx_p = find(py > ry & py > cy);
indx_c = find(cy > py & cy > ry);
```

```
%plot polynomials
figure(2),
plot(rx, ry, 'k-');
hold on
plot(px, py , 'b-')
plot(cx, cy, 'r-')

% disegna il punto ad ordinata maggiore per ogni curva
plot(rx(indx_r), ry(indx_r) , 'ro' , 'LineWidth' , 2);
legend('retta' , 'parabola' , 'cubica' , 'maggiore')
plot(px(indx_p), py(indx_p) , 'ro' , 'LineWidth' , 2);
plot(cx(indx_c), cy(indx_c) , 'ro' , 'LineWidth' , 2);
hold off
```

Diagrammi lineari a tre dimensioni

- Generalizzazione del diagramma a due dimensione: insieme di terne di coordinate
- `plot3(x,y,z)` disegna un diagramma cartesiano con `x` come ascisse, `y` come ordinate e `z` come quote
- funzioni `xlabel, ylabel, zlabel, title`
- Esempio

```
>> t = 0:0.1:10*pi;
>> plot3 (t.*sin(t), t.*cos(t), t);
>> xlabel('ascisse');
>> ylabel('ordinate');
>> zlabel('quote');
```



Superfici

- Come si disegna una superficie che rappresenta una funzione a due variabili $z = f(x,y)$?
- La funzione `mesh(xx,yy,zz)` genera superficie, a partire da tre argomenti
  - `xx` contiene le ascisse
  - `yy` contiene le ordinate
  - `zz` contiene le quote
- `xx` e `yy` identificano una griglia in corrispondenza del quale per `zz` rappresenta il valore della funzione in corrispondenza di quell'ascissa e di quell'ordinata

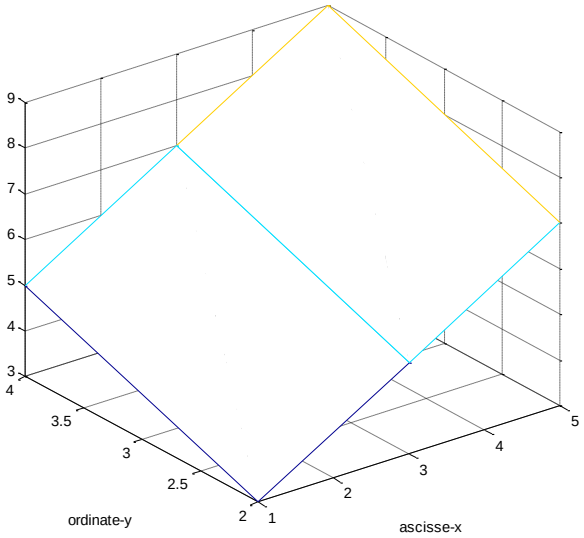
Funzione meshgrid

- Le due matrici, xx, e yy, si possono costruire, mediante la funzione `meshgrid(x,y)`
- `[xx,yy]=meshgrid(x,y)`
  - `x` e `y` sono due vettori
  - `xx` e `yy` sono due matrici entrambe di legth(y) righe e length(x) colonne
  - la prima, `xx`, contiene, ripetuti in ogni riga, i valori di `x`
  - la seconda, `yy` , contiene, ripetuti in ogni colonna, i valori di `y` trasposto

Superfici: esempi

Disegnamo z=x+y

```
>> x=[1, 3, 5];
>> y=[2, 4];
>> [xx,yy]=meshgrid(x,y);
>> zz=xx+yy;
>> mesh(xx,yy,zz);
>> xlabel('ascisse-x');
>> ylabel('ordinate-y');
```




```
>> xx
xx =
     1     3     5
     1     3     5
```

```
>> yy
yy =
     2     2     2
     4     4     4
```

Punti di coordinate (x,y)...

(1,2) (3,2) (5,2)
(1,4) (3,4) (5,4)

```
>> zz
zz =
     3     5     7
     5     7     9
```

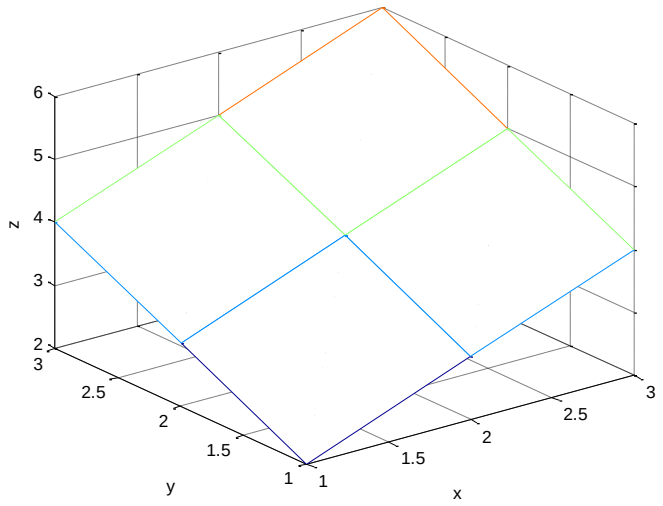
...hanno coordinate (x,y,z)

(1,2,3) (3,2,5) (5,2,7)
(1,4,5) (3,4,7) (5,4,9)

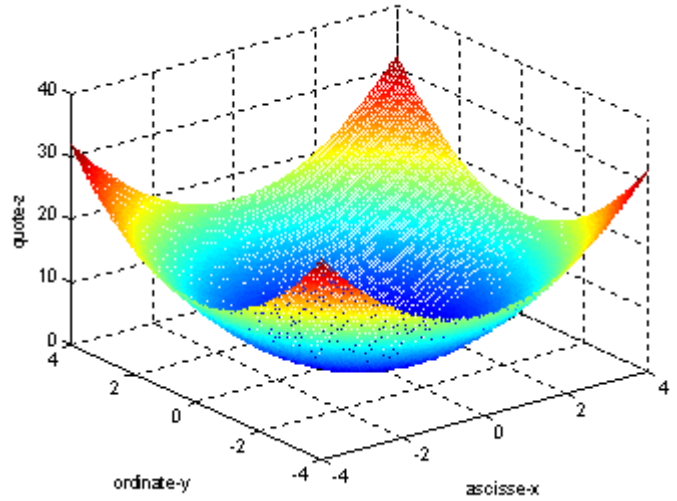
(NB: z=x+y)

Superfici: esempi (2)

```
>> x=[1:1:3];
>> y=x;
>> [xx,yy]=meshgrid(x,y);
>> zz=xx+yy;
>> mesh(xx,yy,zz);
>> xlabel('x');
>> ylabel('y');
>> zlabel('z');
```




```
>> x=[-4:0.05:4];
>> y=x;
>> [xx,yy]=meshgrid(x,y);
>> zz=xx .^ 2 + yy .^ 2;
>> mesh(xx,yy,zz);
>> xlabel('ascisse-x');
>> ylabel('ordinate-y');
>> zlabel('quote-z');
```



Hold on

- Le superfici vengono visualizzate su un grafico 3D.
- È quindi possibile aggiungere degli elementi in sovraimpressione utilizzando la funzione
  - plot3(), mesh(),altre funzioni grafiche quali surf() etc..
- Per sovrascrivere ad un grafico usare la funzione hold on e hold off quando si ha terminato
- Esempio, disegnare in sovraimpressione al paraboloide precedente la curva $\begin{cases} z = x^2 \\ y = 0 \end{cases}$ dove `x=[-4:0.05:4];`

```
figure, mesh(xx, yy, zz)
hold on
% aggiunge una linea rossa con uno spessore di 5
plot3(x, zeros(size(x)), x.^2, 'r-', 'LineWidth', 5);
hold off
```

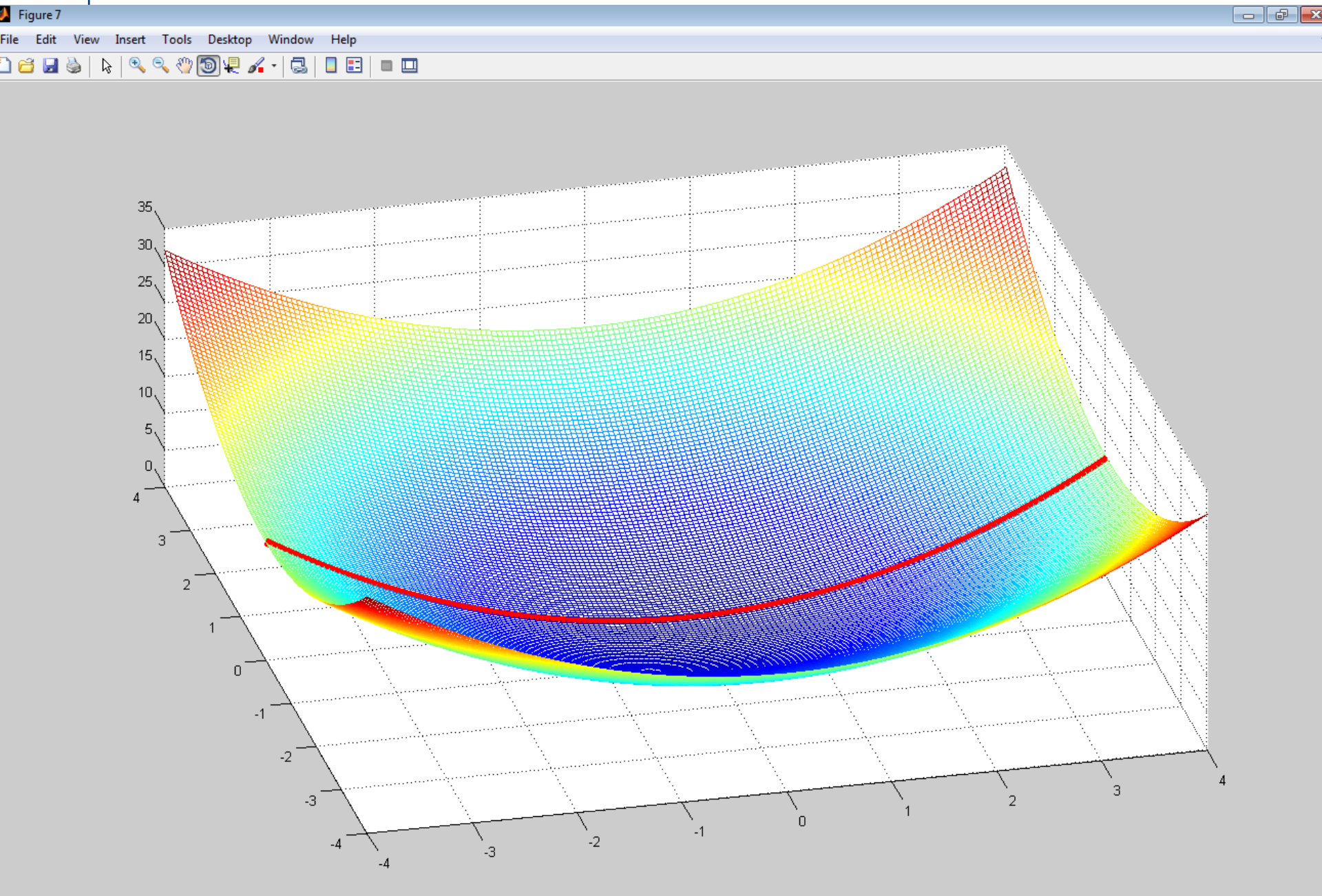
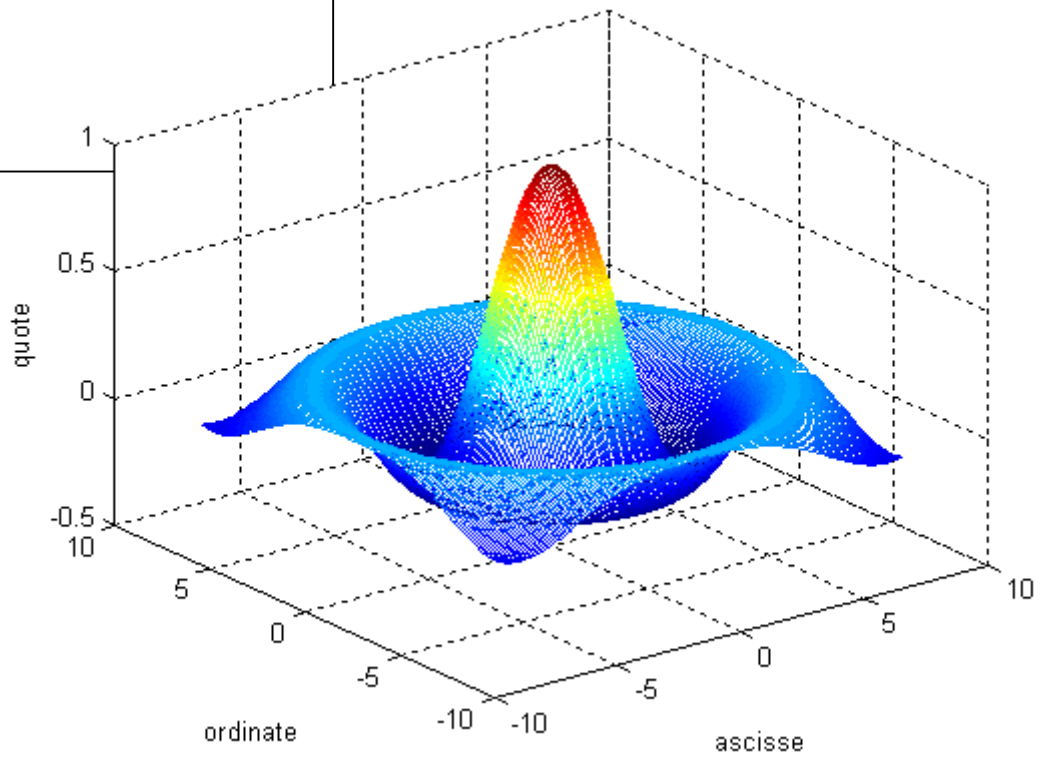


Figure 7
File Edit View Insert Tools Desktop Window Help
35
30
25
20
15
10
5
0
4
3
2
1
0
-1
-2
-3
-4
-4 -3 -2 -1 0 1 2 3 4

Superfici: esempi (3)

```
>> tx=[-8:0.1:8];
>> ty=tx;
>> [xx, yy] = meshgrid (tx, ty);
>> r = sqrt (xx .^ 2 + yy .^ 2);
>> tz = sin (r) ./ r;
>> mesh (xx, yy, tz);
>> xlabel('ascisse');
>> ylabel('ordinate');
>> zlabel('quote');
```



Funzioni per Stringhe e Return

Funzioni per Stringhe

- Esiste la funzione di **confronto**

  TF = strcmp(str1 , str2)

  - INPUT: str1,str2 stringhe da confrontare
  - OUTPUT: TF valore booleano o ,1 **(è diverso dal C)**
  - Similmente strcmpi(str1, str2) non fa differenze tra maiuscole e minuscole

- **NB:** in linea di principio è possibile confrontare le stringhe come due vettori, con l'operatore == . Questo però richiede che le **due stringhe abbiano le stesse dimensioni.** Altrimenti genera errori
  - La funzione strcmp permette di confrontare anche stringhe di dimensione diverse (restituendo false).

```
if('cane'== 'canguro')
 disp('uguali')
else
    disp('diverse')
End
›› Error using  ==

Matrix dimensions must agree.


if strcmp('cane','canguro')
    disp('uguali')
else
    disp('diverse')
end
›› diverse
```

Funzioni per Stringhe

- **Non** occorre strlen (si usa length o size)
- **Non** occorre strcpy (la copia tra stringhe è nativa in Matlab)
- Esiste la funzione di **ricerca**

K = strfind(TEXT,PATTERN)

  - INPUT: PATTERN stringa da ricercare in TEXT
  - OUTPUT: K indice di tutte le occorrenze (vuoto se non ce ne sono)

Return

- Non necessario in Matlab,
  - I valori ritornati sono definiti dall'header della funzione
- Tuttavia può essere usata per terminare l'esecuzione della funzione

```
function [p,m]=cercaMultiplo(v, a)
for k = 1 : length(a)
    if mod(a(k), v)==0
        p=k; m=a(k);
        return;  %si restituisce il primo multiplo incontrato
        % evita ulteriori inutili calcoli
    end;
end;
p=0; m=0;  %eseguite solo se non trovato alcun multiplo
```