Matlab: Funzioni

Informatica B AA 17/18

Giacomo Boracchi

giacomo.boracchi@polimi.it

22 Novembre 2017

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

Prepara l'input in n

chiama lo script

Salva il risultato in f

```
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?

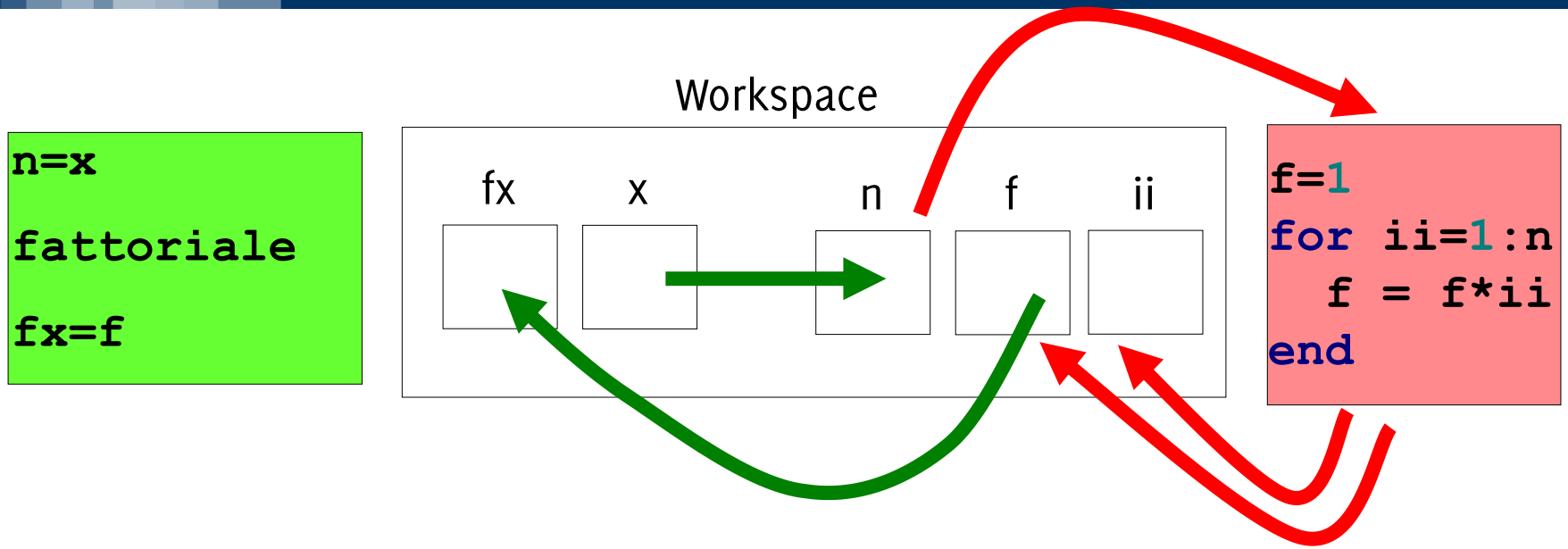
Gli script utilizzano le variabili del workspace:

```
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. Se ii fosse usata nel main questa sarebbe sovrascritta
- Le funzioni non hanno questi problemi

Le funzioni

header

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

body

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;
```

Sintassi per la Definizione 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. Questa è diversa perché **`out1, .., outM`** possono avere dimensioni e tipi non consistenti!

NB: se la funzione non ha parametri in ingresso/uscita le parentesi tonde/quadre rimangono vuote

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.

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

Esempi

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

Invocazione

Invocazione

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

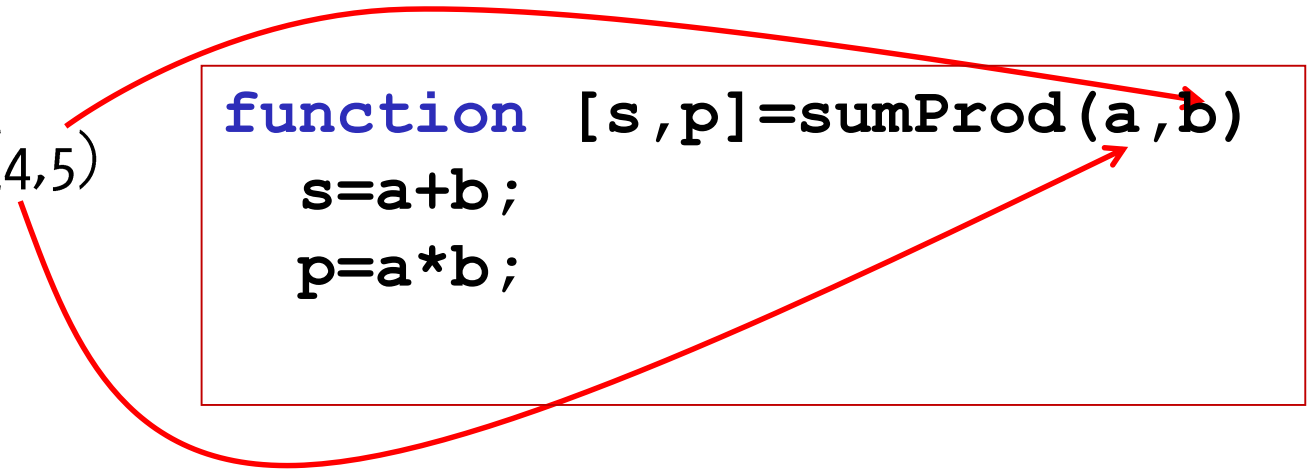
I Parametri (2)

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)
```



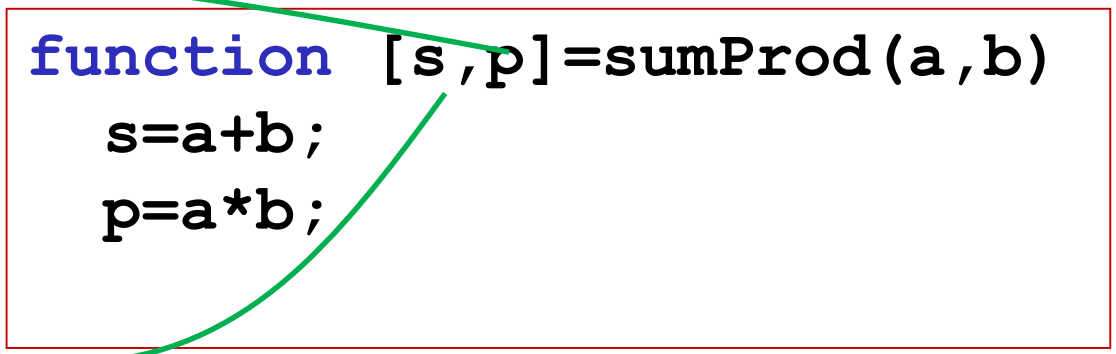
I Parametri (2)

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

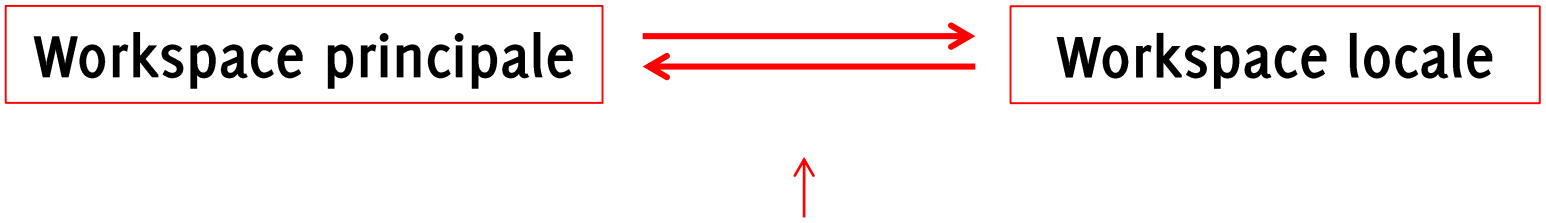
```
>> [x,y]=sumProd(4,5)
```



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 di nomi)
- 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
```

Esempio

```
function [minore, maggiore] = minmax(a,b,c)
% alternativa che utilizza le funzioni
% built in di Matlab
maggiore = max([a, b, c]);
minore = min([a, b, c]);
```

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 (line 2)
positivi = vett(vett >0);

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

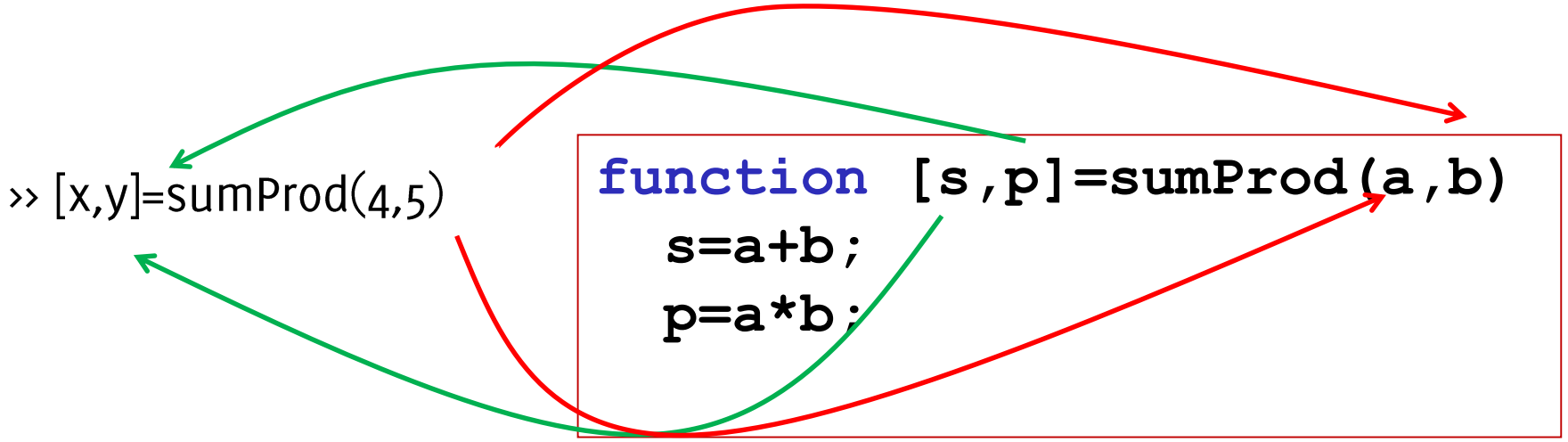
Note sui Parametri in Uscita

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
- La prima riga del file deve contenere l'header della funzione e di fatto 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

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

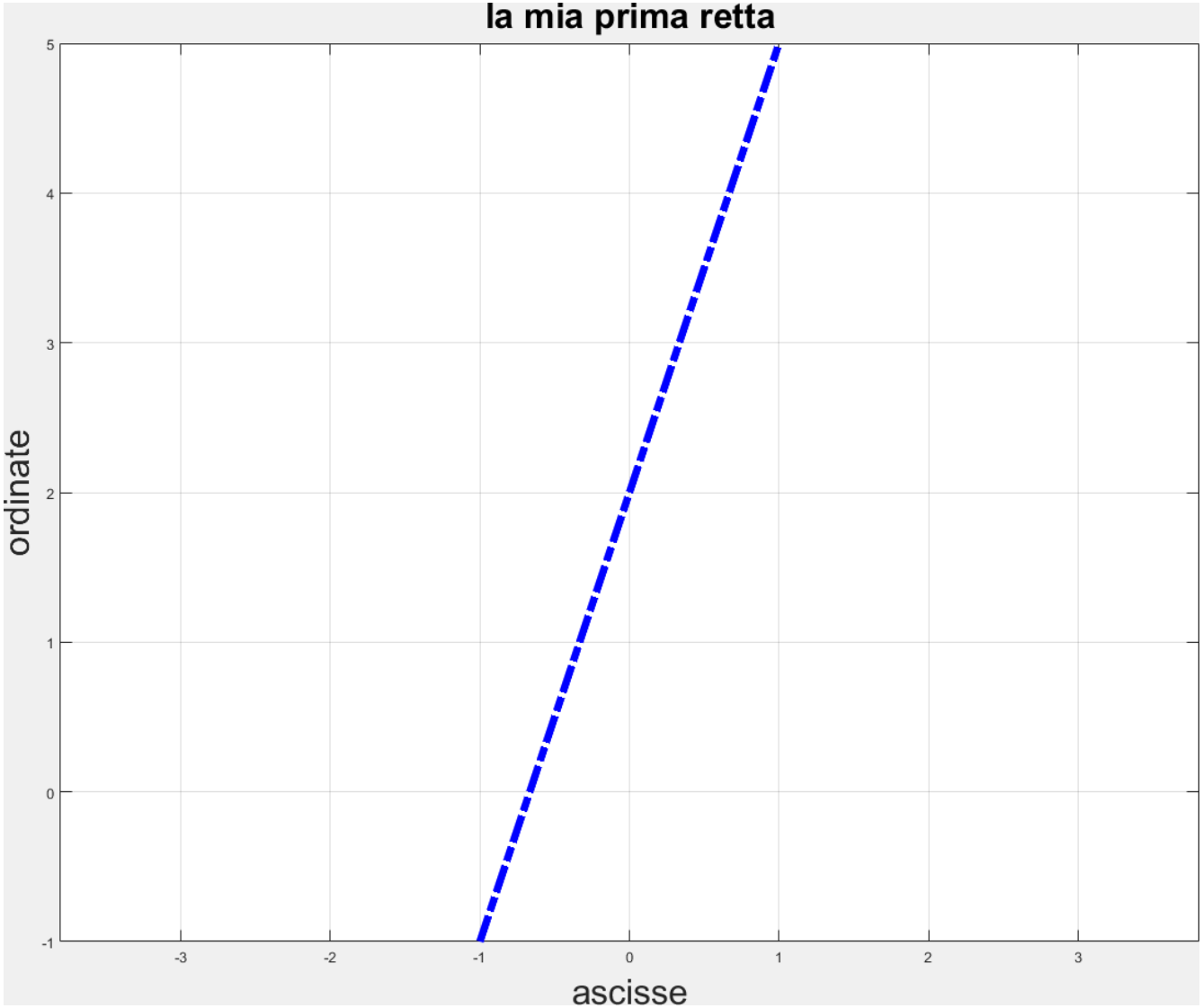
Scrivere una funzione che prende in ingresso due coefficienti m, q ed un vettore di punti xx e restituisce il vettore yy dei punti che stanno sulla retta $y = mx + q$ in corrispondenza a xx

```
function yy = retta(m, q, xx)
% function yy = retta(m, q, xx)
% m, q  sono i coefficienti e xx un vettore di punti
% la funzione restituisce il vettore yy dei punti che
stanno sulla
% retta y = mx + q in corrispondenza a xx

yy = m * xx + q;
```

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

Implementare la funzione trasposizione per le matrici

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.

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.

```
function [] = contoAllaRovescia(n)
disp(['... ', num2str(n)])
for ii = [n - 1: -1 : 0]
    pause(1)
    disp(['... ', num2str(ii)])
end
disp('BOOOM!')
beep
```

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.

Se la funzione non restituisce nulla si può omettere [] =

```
function contoAllaRovescia(n)
disp(['... ', num2str(n)])
for ii = [n - 1: -1 : 0]
    pause(1)
    disp(['... ', num2str(ii)])
end
disp('BOOOM!')
beep
```

Esercizio

%% scirivere un programma che chiede all'utente di inseririe 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

Implemento diverse funzioni che richiamo

```
function n = inserisciInteroPositivo()
% function n = inserisciInteroPositivo()
%
% richiede all'utente di inserire un intero positivo
% e lo restituisce

function somma = calcolaSommaDivisori(n)
%function somma = calcolaSommaDivisori(n)
%
% calcola la somma di tutti i divisori di n escluso n

function [res, abb] = controllaSePerfetto(n)
% function [res, abb] = controllaSePerfetto(n)
%
% res = true se n è perfetto (uguale alla somma dei suoi
divisori escluso se stesso)
%  se res = false e abb = true/false se è abbondante o
difettivo

function res = controllaSeAmici(a,b)
% function res = controllaSeAmici(a,b)
%
% res = 1 se a è amico di b, 0 altrimenti
```