Struct e Ricorsione in Matlab

Informatica ICA (LC)

Giacomo Boracchi

giacomo.boracchi@polimi.it

2 Novembre 2017

Strutture in Matlab

Struct vs Array

Gli **array** permettono di aggregare variabili **omogenee** in una sequenza

Le `struct` permettono di aggregare variabili **eterogenee** in una sola variabile

- Le `struct` è una sorta di "contenitore" per variabili disomogenee di tipi più semplici.
- Le variabili aggregate nella struct sono dette **campi** della struct

Esempio: variabile per contenere anagrafica di impiegati

- *nome, cognome, codice fiscale, indirizzo, numero di telefono, stipendio, data di assunzione etc.*
- *Non posso metterli in un array, sono variabili diverse, è molto sconveniente metterle in variabili separate, specialmente se ho diversi impiegati*

Creazione di una struct

Creazione di una struttura :

Utilizzando la funzione struct()

```
studente = struct('nome', 'Giovanni', 'eta', 24)
```

Assegnamento dei valori ai campi (e contestuale definizione dei campi)

```
studente.nome = 'Giovanni';
studente.eta = 24;
```

Accedere ai campi di una `struct`

Per accedere ai campi si usa l'operatore ***dot***.
Sintassi:

```
nomeStruct.nomeCampo;
```

Quindi, **`nomeStruct.nomeCampo`** diventa, a tutti gli effetti, **una «normale» variabile** del tipo di **`nomeCampo`**.

- **Ai campi** di una struttura applicabili tutte le **operazioni caratteristiche** del tipo di appartenenza
- In questo senso, il *dot* è l'omologo di **`(indice)`** per gli array

Creazione di una struttura campo per campo

Esempio: creo una struttura studente

```
studente.nome = 'Giovanni Rossi';
studente.indirizzo = 'Via Roma 23';
studente.citta = 'Cosenza';
studente.eta = 25;
```

Accesso ai campi come nel C con l'operatore .

`nomeStruttura.nomeCampo`

Es

```
disp([studente.nome, ' (' , studente.citta ,') ha ' ,
num2str(studente.eta) , ' anni'])
```

Creazione di una struttura campo per campo

Esempio: la struttura studente

```
studente.nome = 'Giovanni Rossi';
studente.indirizzo = 'Via Roma 23';
studente.citta = 'Cosenza';
studente.media = 25;
```

É possibile far diventare **`studente`** un array di strutture, accodando un altro elemento in **`studente(2)`**.

```
studente(2).nome = 'Giulia Gatti';
studente(2).media = 30;
```

Tutte le strutture dell'array devono avere gli stessi campi (**l'array deve essere omogeneo, la struttura non necessariamente**).

É possibile assegnare solo alcuni campi a **`studente(2)`**: i campi non assegnati rimangono vuoti.

Aggiunta di campi

Aggiunta di un campo

```
%facciamo riferiemento alla definizione di studente delle slide precedenti

studente(2).esami = [20 25 30];
```

Il campo esami viene aggiunto a tutte le strutture che fanno parte di studente

- Avrà un valore iniziale per studente(2). Sarà vuoto per tutti gli altri elementi dell'array

Creazione di una struttura mediante la funzione struct

Consente di preallocare una struttura o un array di strutture

```
S = struct('campo1',val1,'campo2',val2, …)
```

Es:
```
rilieviAltimetrici =
struct('latitudine',30,'longitudine',60,
'altitudine', 1920)
```

Creazione di una struttura mediante la funzione struct

Consente di preallocare una struttura o un array di strutture

```
S = struct('campo1',val1,'campo2',val2, …)
```

Es: `rilieviAltimetrici = struct('latitudine',30,'longitudine',60, 'altitudine', 1920)`

Esempio array di strutture:

```
s(5) = struct('x',10,'y',3);
```

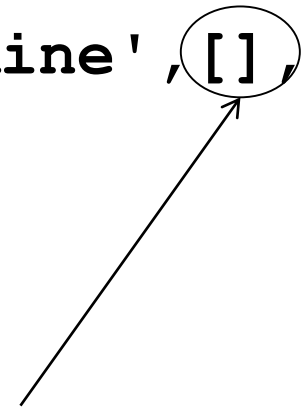
- s è un array 1x5 in cui ogni elemento ha attributi x e y
- solo il quinto elemento di s viene inizializzato con i valori x=10 e y=3
- gli altri elementi vengono inizializzato con il valore di default: [] (array vuoto)

Creazione di una struttura mediante la funzione struct

Consente di preallocare una struttura o un array di strutture

```
S = struct('campo1',val1,'campo2',val2, …)
```

Es: `rilieviAltimetrici(1000) = struct('latitudine',30,'longitudine',[], 'altitudine', 1920)`



Array vuoto. Attenzione: se si Inserisce un valore (es. 20), questo viene assunto dal campo longitudine dell'elemento 1000, ma non dallo stesso campo degli altri elementi dell'array

Array di strutture innestati

Un campo di una struttura di strutture può essere di qualsiasi tipo

E` quindi possibile avere un campo che è, di nuovo, una struttura o un array di strutture

Esempio

```
studente(1).corso(1).nome='InformaticaB';
studente(1).corso(1).docente='Von Neumann';
studente(1).corso(2).nome='Matematica';
studente(1).corso(2).docente='Eulero';
```

corso è un array di strutture

```
›› studente
studente =
    corso: [1x2 struct]
```

Esercizio

Si sviluppi uno script matlab che acquisisce da tastiera i dati relativi ad un numero arbitrario di rilievi altimetrici e che quindi stampa a video l'altitudine media di tutti i rilievi che si trovano nell'intervallo

- latitudine [30, 60]
- longitudine [10, 100]

Soluzione

```
% s = struct('altezza',[],'latitudine',[], 'longitudine',[])
n = input(['quanti rilievi? ']);
 % acquisizione dei rilievi
for ii = 1 : n
    s(ii).altezza = input(['altezza rilievo nr ', num2str(ii), ' ']);
    s(ii).latitudine= input(['latitudine rilievo nr ', num2str(ii), ' ']);
    s(ii).longitudine= input(['longitudine rilievo nr ', num2str(ii), ' ']);
end
% creo dei vettori con i valori dei campi
LAT = [s.latitudine];
LON = [s.longitudine];
ALT = [s.altezza];
% operazioni logiche per definire il sottovettore da estrarre da altezza
latOK = (LAT > 30) & (LAT <60);
lonOK = (LON> 10) & (LON<100);
posOK =latOK & lonOK;

% estrazione sottovettore e calcolo media
mean(ALT(posOK));
```

Funzioni

Richiami

Un esempio

```
function f = fattoriale(n)
f = 1;
for ii = 2 : n
    f = f * ii;
end
```

Un esempio

Parametro formale in uscita

```
function f = fattoriale(n)
f = 1;
for ii = 2 : n
    f = f * ii;
end
```

Parametro formale in ingresso

Definizione di $n!$

Il fattoriale di un intero $n > 0$ è definito come segue:

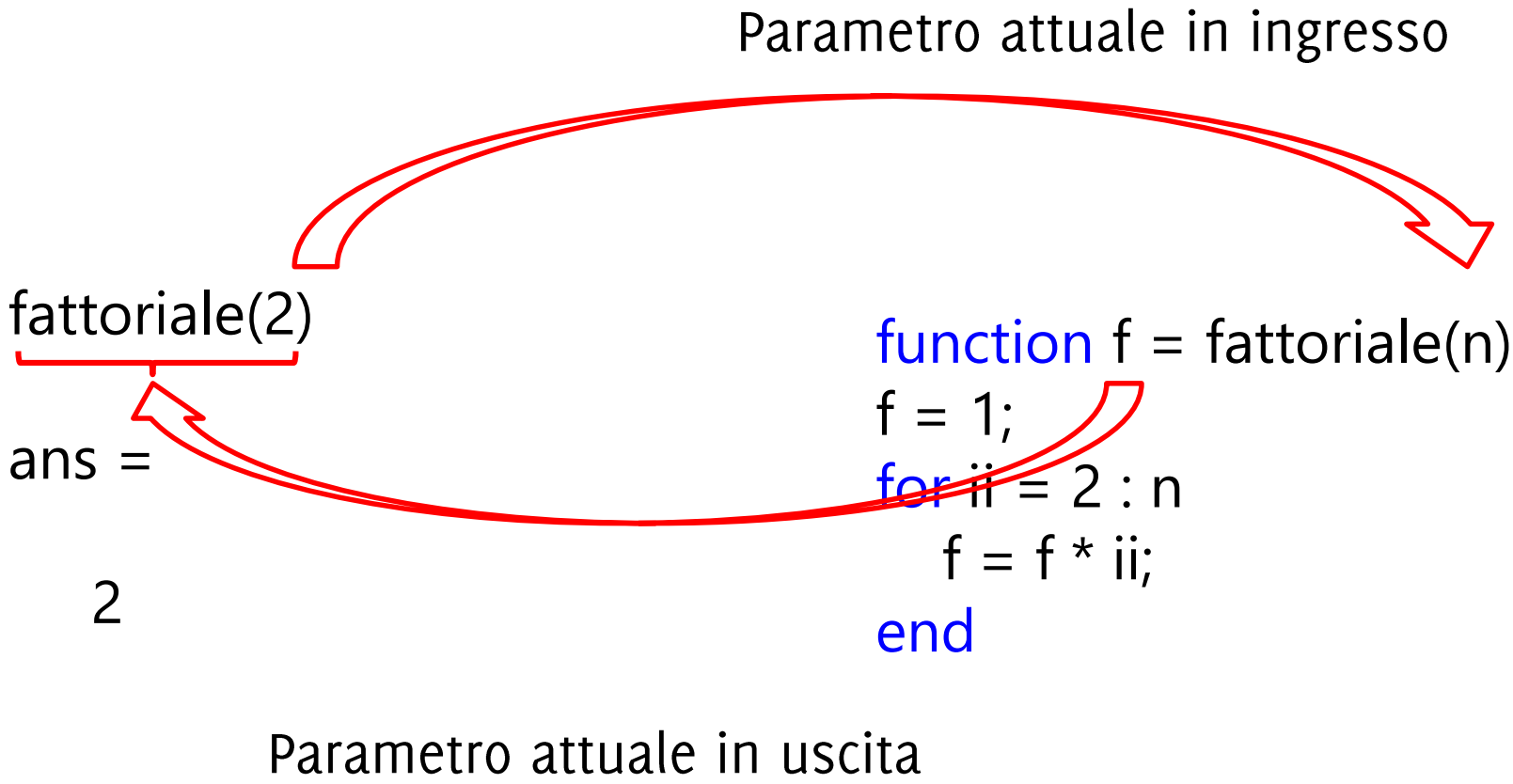
$$n! = n * (n-1) * (n-2) * \cdots * 2 * 1$$

Un esempio di chiamata

```
fattoriale(2)

ans =

     2
```

```
function f = fattoriale(n)
f = 1;
for ii = 2 : n
    f = f * ii;
end
```

Un esempio di chiamata



Un esempio di chiamata

```
k = 2;
f2 = fattoriale(k);
```

Workspace principale

- Da qui si invoca la funzione
- Contiene le variabili k, f2
- Non ha visibilità di f,ii,n

Un esempio di chiamata

```
k = 2;
f2 = fattoriale(k);
```

Workspace principale

- Da qui si invoca la funzione
- Contiene le variabili k, f2
- Non ha visibilità di f,ii,n

```
function f = fattoriale(n)
f = 1;
for ii = 2 : n
    f = f * ii;
end
```

Workspace locale

- Creato al momento dell'invocazione di fattoriale
- Contiene le variabili n, f,ii
- Non ha visibilità su k ed f2
- Non ha legami con il workspace principale se non per i parametri attuali che vengono copiati (sia in ingresso che in uscita)
- Distrutto terminata l'esecuzione

Un esempio

f2 = fattoriale(2)

f2 =

2

f3 = fattoriale(3)

f3 =

6

f4 = fattoriale(4)

f4 =

24

f5 = fattoriale(5)

f5 =

120

f6 = fattoriale(6)

f6 =

720

Un esempio

Vale la seguente relazione

$$n! = n * (n-1)!$$

si dimostra immediatamente dalla definizione di fattoriale

$$n! = n * \underbrace{(n-1) * (n-2) * \cdots * 2 * 1}_{(n-1)!}$$

È possibile usare questa proprietà per definire un'implementazione di fattoriale totalmente diversa?

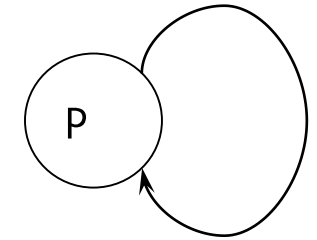
Ricorsione

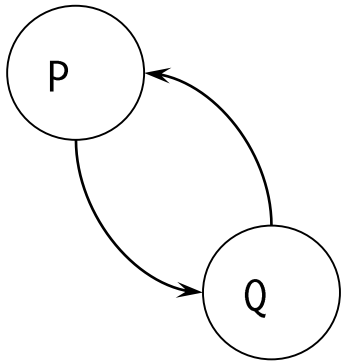
Programmi che chiamano se stessi

Ricorsione

Che cos'è la ricorsione?

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





È una tecnica di programmazione molto potente

Permette di risolvere in maniera elegante problemi complessi

Le funzioni che richiamano se stesse (direttamente o indirettamente) sono dette **funzioni ricorsive**

Esempio: il fattoriale Ricorsivo

```
function [f] = factRic(n)
  if (n == 0)
      f = 1;
  else
      f = n * factRic(n - 1);
  end
```

Esempio: il fattoriale Ricorsivo

```
function [f] = factRic(n)
  if (n == 0)
     f = 1;
  else
     f = n * factRic(n - 1);
  end
```

Questa ci ricorda
$0! = 1$

Esempio: il fattoriale Ricorsivo

```
function [f] = factRic(n)
  if (n == 0)
     f = 1;
  else
     f = n * factRic(n - 1);
  end
```

Questa ci ricorda
$n! = n * (n-1)!$

Una funzione che chiama se stessa

Una Funzione che Chiama se Stessa?

In ogni istante possono essere in corso **diverse attivazioni** dello **stesso** sottoprogramma

- Ovviamente sono **tutte sospese tranne** una, **l'ultima invocata**, all'interno della quale si sta svolgendo il **flusso di esecuzione**.

Una Funzione che Chiama se Stessa?

In ogni istante possono essere in corso **diverse attivazioni** dello **stesso** sottoprogramma

- Ovviamente sono **tutte sospese tranne** una, **l'ultima invocata**, all'interno della quale si sta svolgendo il **flusso di esecuzione**.

Ogni attivazione esegue **lo stesso codice** ma opera su **workspace distinti** (in Matlab, ogni funzione attivata ha un workspace distinto)

- Si hanno quindi **copie distinte** dei **parametri attuali** e delle **variabili locali** nelle varie invocazioni

Una funzione che chiama se stessa?

… se ogni volta la funzione richiama se stessa… *perché la catena di invocazioni non continua* ***all'infinito***?

Una funzione che chiama se stessa?

… se ogni volta la funzione richiama se stessa… *perché la catena di invocazioni non continua* ***all'infinito****?*

La funzione ricorsiva deve prevedere una situazione in cui non richiama se stessa, i.e., il **caso base**

Programmazione Ricorsiva

Per risolvere un problema attraverso la programmazione ricorsiva sono **necessari alcuni elementi**

- **Caso base**: caso elementare del problema che può essere risolto immediatamente
- **Passo ricorsivo**: chiamata ricorsiva per risolvere uno o più problemi più semplici
- **Costruzione della soluzione**: costruzione della soluzione sulla base del risultato delle chiamate ricorsive

Esempio: il fattoriale

Definizione:
f(n) = n! = n*(n-1)*(n-2)*…*3*2*1

Definire caso base, e passo ricorsivo

Esempio: il fattoriale

Definizione:
f(n) = n! = n*(n-1)*(n-2)*…*3*2*1

Passo ricorsivo:
f(n) = n*f(n-1)

Caso base:
f(0)=1

Esempio: il fattoriale

Definizione:
f(n) = n! = n*(n-1)*(n-2)*…*3*2*1

Passo ricorsivo:
f(n) = n*f(n-1)

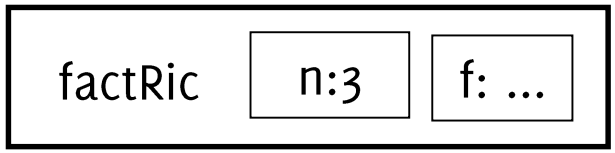
Caso base:
f(0)=1

```
function [f]=factRic(n)
  if (n==0)
      f=1;
  else
      f=n*factRic(n-1);
  end
```

Funzionamento ricorsione

```
function [f]=factRic(n)
  if (n==0)
      f=1;
  else
      f=n*factRic(n-1);
  end
```

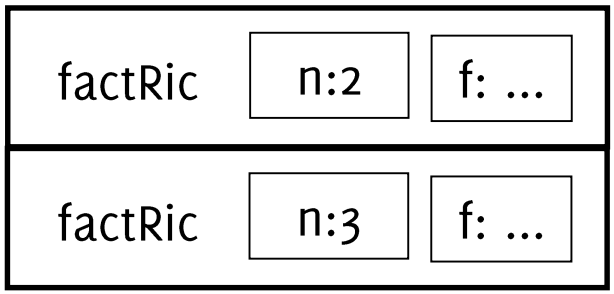
factRic(3)



Funzionamento ricorsione

```
function [f]=factRic(n)
  if (n==0)
      f=1;
  else
      f=n*factRic(n-1);
  end
```

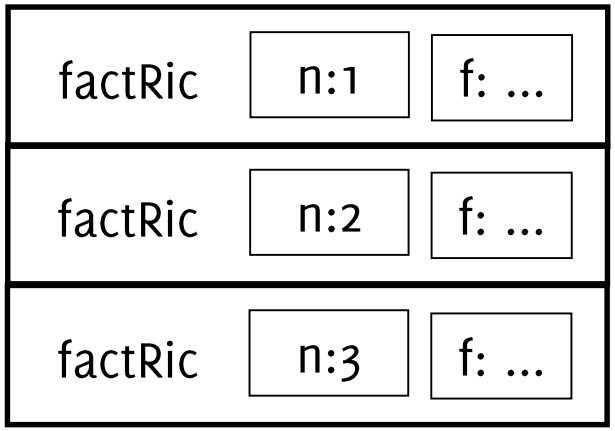
factRic(3)



Funzionamento ricorsione

```
function [f]=factRic(n)
  if (n==0)
      f=1;
  else
      f=n*factRic(n-1);
  end
```

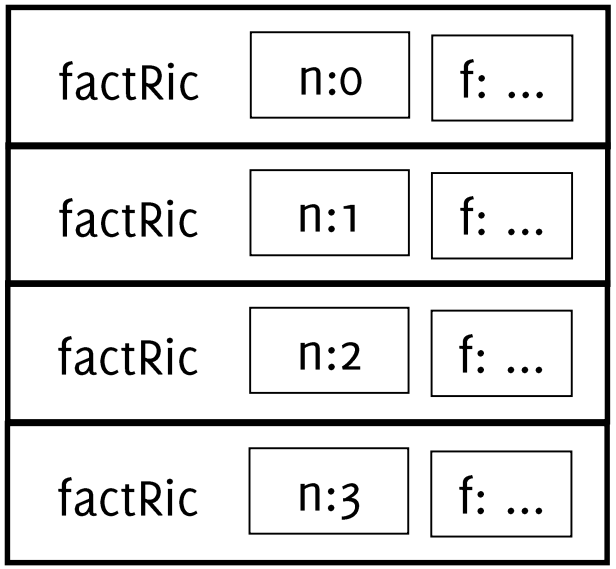
factRic(3)



Funzionamento ricorsione

```
function [f]=factRic(n)
  if (n==0)
      f=1;
  else
      f=n*factRic(n-1);
  end
```

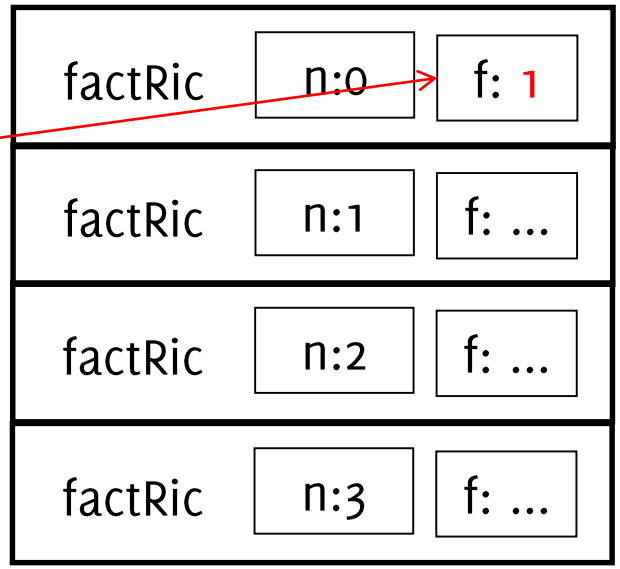
factRic(3)



Funzionamento ricorsione

```
function [f]=factRic(n)
  if (n==0)
      f=1;
  else
      f=n*factRic(n-1);
  end
```

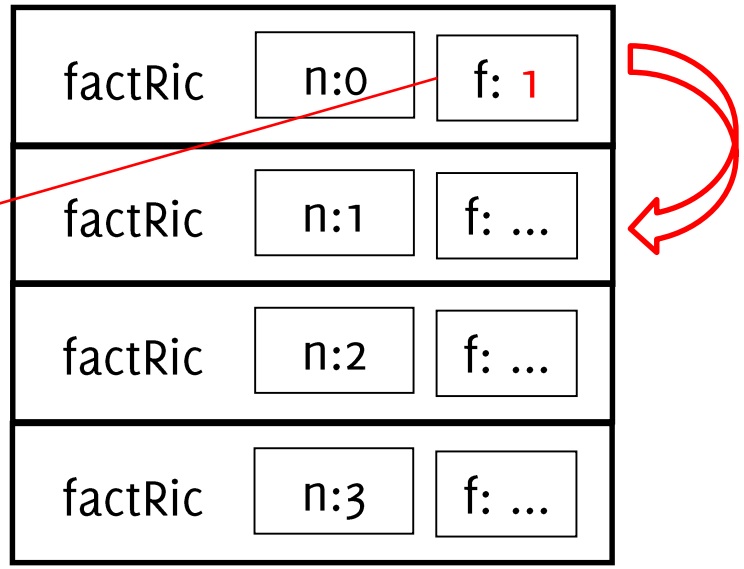
factRic(3)



Funzionamento ricorsione

```
function [f]=factRic(n)
  if (n==0)
      f=1;
  else
      f=n*factRic(n-1);
  end
```

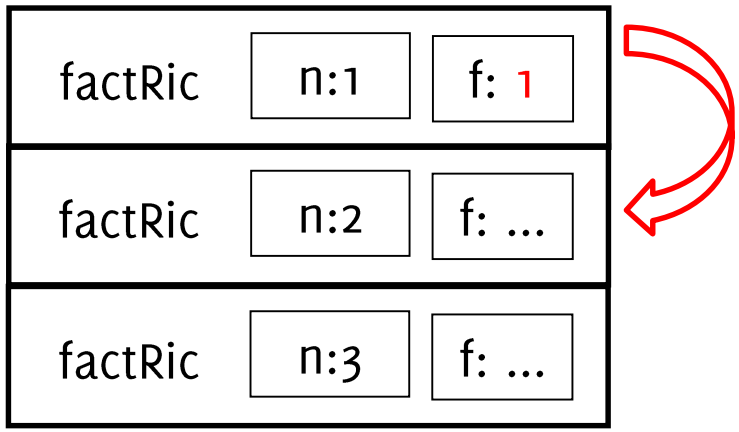
factRic(3)



Funzionamento ricorsione

```
function [f]=factRic(n)
  if (n==0)
      f=1;
  else
      f=n*factRic(n-1);
  end
```

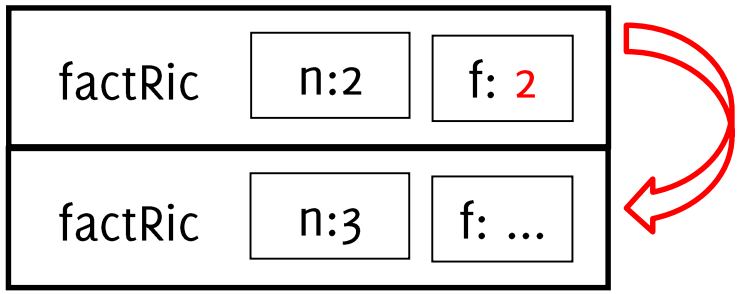
factRic(3)



Funzionamento ricorsione

```
function [f]=factRic(n)
  if (n==0)
      f=1;
  else
      f=n*factRic(n-1);
  end
```

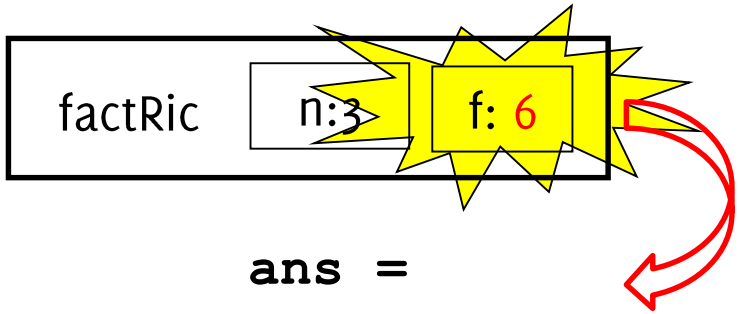
factRic(3)



Funzionamento ricorsione

```
function [f]=factRic(n)
  if (n==0)
      f=1;
  else
      f=n*factRic(n-1);
  end
```

factRic(3)



ans =

6

Ricorsione in Coda

Ricorsione in cui la funzione:

- prevede **una sola chiamata ricorsiva**
- esegue la chiamata ricorsiva come **ultima istruzione**

```
function [f]=factRic(n)
  if (n==0)
      f=1;
  else
      f=n*factRic(n-1);
  end
```

Problemi nell'Uso della Ricorsione

Terminazione della catena ricorsiva

- È presente il caso base?
- Viene raggiunto sempre dalla catena di chiamate ricorsive?

Errori nell'Uso della Ricorsione

Catena infinita di chiamate incondizionate

- **Deve esistere** una **condizione** tale per cui **non** viene eseguita la **chiamata ricorsiva** (**caso base**)

```
function [f]=factRic(n)
    f=n*factRic(n-1);
```

Es la chiamata a factRic(7) chiama factRic(7),
che chiama factRic(6),
che chiama factRic(5),
che chiama factRic(4),
che chiama factRic(3),….

che chiama factRic(-1000),….

Errori nell'Uso della Ricorsione

Catena infinita di chiamate incondizionate

- Attenzione che **è necessario che questa condizione venga raggiunta**: anche programmi formalmente corretti potrebbero non funzionare per alcuni valori degli ingressi

```
function [f]=factRic(n)
  if (n==0)
      f=1;
  else
      f=n*factRic(n-1);
  end
```

- Ad es, factRic(-1) da luogo ad una sequenza di chiamate infinite

Errori nell'Uso della Ricorsione

Catena infinita di chiamate identiche:

- La chiamata ricorsiva **non** può avere i **parametri formali uguali a quelli attuali**

```
function [f]=factRic(n)
  if (n==0)
      f=1;
  else
      f=n*factRic(n);
  end
```

Es la chiamata a factRic(7) chiama factRic(7),
che chiama factRic(7),
che chiama factRic(7),
che chiama factRic(7),….

Di Default…

Matlab si blocca a 500 chiamate ricorsive

>> factRic(600)

Maximum recursion limit of 500 reached. Use set(0,'RecursionLimit',N) to change the limit. Be aware that exceeding your available stack space can crash MATLAB and/or your computer.

Error in **factRic**

Condizioni Necessarie

.. Per evitare ricorsione infinita:

Occorre che le **chiamate ricorsive** siano **soggette** a una **condizione** che prima o poi assicura che la catena termini

Occorre anche che **l'argomento sia *progressivamente ridotto*** dal passo induttivo, in modo da tendere prima o poi al caso base

- Nella pratica l'argomento si avvicina al valore nel caso base

Esempio

Scrivere una funzione ricorsiva che calcola la somma di tutti gli interi compresi tra i due argomenti

Esempio

Scrivere una funzione ricorsiva che calcola la somma di tutti gli interi compresi tra i due argomenti

```
function somma = sommaNumeriCompresi(a ,b)

if a == b
    somma = a;
    disp(['caso base somma vale ' , num2str(somma)]);

else
    disp(['prima chiamata a = ' ,num2str(a)]);
    somma = a + sommaNumeriCompresi(a+1 , b);
    disp(['dopo chiamata a = ' ,num2str(a)]);
end
```

Esempio

Scrivere una funzione ricorsiva che calcola la somma di tutti gli interi compresi tra i due argomenti

```
function somma=calcolaSommaCompresi2(a_temp,b_temp)
a=min([a_temp,b_temp]);
b=max([a_temp,b_temp]);
if(a==b)
    disp(['caso base 1 a=',num2str(a),'b=',num2str(b)])
    somma=a;
else
    if(b-a==1)
        disp(['caso base 2 a=',num2str(a),'b=',num2str(b)])
        somma=a+b;
    else
        disp(['prima della chiamata ricorsiva a=',num2str(a),'b=',num2str(b)])
        somma=a+calcolaSommaCompresi2(a+1,b-1)+b;
        disp(['dopo la chiamata ricorsiva a=',num2str(a),'b=',num2str(b),'
somma =',num2str(somma)])
    end
end
```

Esempio

Scrivere una funzione ricorsiva per calcolare la lunghezza di una stringa

Esempio

Scrivere una funzione ricorsiva per calcolare la lunghezza di una stringa

```
function s = calcolaLunghezza(str)
if(isempty(str))
    s = 0;
else
    s = 1 + calcolaLunghezza(str(2 : end));
end
```

Esempio:

Scrivere una funzione per stampare una stringa al contrario

Esempio:

Scrivere una funzione per stampare una stringa al contrario

```
function stampaAlContrario(frase)

% caso base
if isempty(frase)
    % return
else
    % chiamata ricorsiva
    disp(frase(end)); % stampa al contrario
    stampaAlContrario(frase(1:end-1))
end
```

Esempio:

Modificare la funzione per stampare la stringa nello stesso ordine in cui viene inserita

```
function stampaAlContrario(frase)

% caso base
if isempty(frase)
    % return
else
    % chiamata ricorsiva
     stampaAlContrario(frase(1:end-1))
     disp(frase(end)); % stampa dritto
end
```

In questo caso la ricorsione non è in coda

Esempio:

Cosa stampa??

```
function stampa(frase)

% caso base
if isempty(frase)
    % return
else
        stampa(frase(2:end))
        disp(frase(1));
end
```

Esempio:

Cosa stampa??

```
function stampa(vettore)
if (length(vettore)==1)
    % caso base
    fprintf('%c',vettore(1));
    fprintf('%c',vettore(1));
else
    fprintf('%c',vettore(1));
    stampa(vettore(2:end));
    fprintf('%c',vettore(1));
end
```

Cosa Stampa?

```
function stampaAlContrario(str)

if(isempty(str))
% return
else
    disp(str(end));
    stampaAlContrario(str(1 : end -1));
    disp(str(end));
end
```

Cosa Stampa?

```
function stampaAlContrario(str)

if(isempty(str))
% return
else
    disp(str(end));
    stampaAlContrario(str(1 : end -1));
    disp(str(end));
end
```

```
>> stampaAlContrario('ciao')
    o
    a
    i
    c
    c
    i
    a
    o
```

Cosa Stampa?

```
function stampaAlContrario(str)

if(isempty(str))
% return
else
    disp(str(end));
    stampaAlContrario(str(2 : end));
    disp(str(end));
end
```

Cosa Stampa?

```
function stampaAlContrario(str)

if(isempty(str))
% return
else
    disp(str(end));
    stampaAlContrario(str(2 : end));
    disp(str(end));
end
```

```
>> stampaAlContrario('ciao')
     o
     o
     o
     o
     o
     o
     o
     o
```

Cosa Stampa?

```
function stampaAlContrario(str)

if(isempty(str))
% return
else
    disp(str(1));
    stampaAlContrario(str(2 : end));
    disp(str(1));
end
```

Cosa Stampa?

```
function stampaAlContrario(str)

if(isempty(str))
% return
else
    disp(str(1));
    stampaAlContrario(str(2 : end));
    disp(str(1));
end
```

```
›› stampaAlContrario('ciao')
    c
    i
    a
    o
    o
    a
    i
    c
```

Cosa Stampa?

```
function stampaAlContrario(str)

if(isempty(str))
% return
else
    disp(str(1));
    stampaAlContrario(str(1 : end -1));
    disp(str(1));
end
```

Cosa Stampa?

```
function stampaAlContrario(str)

if(isempty(str))
% return
else
    disp(str(1));
    stampaAlContrario(str(1 : end -1));
    disp(str(1));
end
```

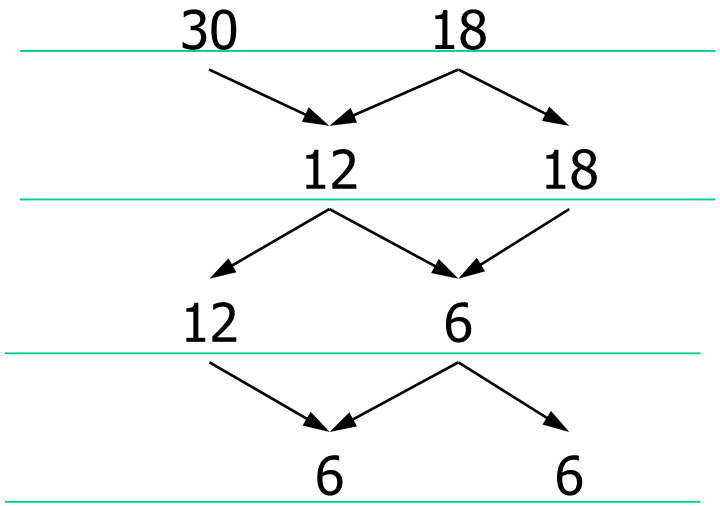
```
>> stampaAlContrario('ciao')
     c
     c
     c
     c
     c
     c
     c
     c
```

Esempio: MCD

Algoritmo di Euclide

- se m = n, MCD(m,n) = m (caso base)
- se m > n, MCD(m,n) = MCD(m-n,n) (caso risorsivo)
- se m < n, MCD(m,n) = MCD(m,n-m) (caso risorsivo)

Esempio: MCD(30,18)



Esempio: MCD

Algoritmo di Euclide

- se m = n, MCD(m,n) = m (caso base)
- se m > n, MCD(m,n) = MCD(m-n,n) (caso risorsivo)
- se m < n, MCD(m,n) = MCD(m,n-m) (caso risorsivo)

```
function [M]=MCDeuclidRic(m,n)
  if m==n
      M=m;
  else
      if m>n
          M = MCDeuclidRic(m-n,n);
      else
          M = MCDeuclidRic(m,n-m);
      end
  end
```

Due possibili chiamate ricorsive, e la chiamata è condizionata

Un altro esempio: la serie di Fibonacci

Fibonacci (1202) partì dallo studio sullo sviluppo di una colonia di conigli in circostanze ideali

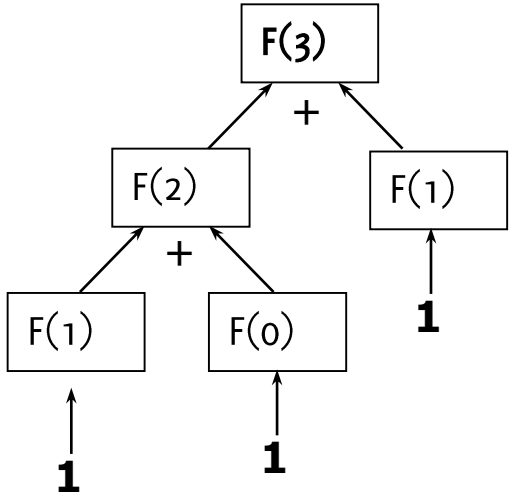
- Partiamo da una coppia di conigli
- I conigli possono riprodursi all'età di un mese
- Supponiamo che dal secondo mese di vita in poi, ogni femmina produca una nuova coppia
- e inoltre che i conigli non muoiano mai…

- Quante coppie ci sono dopo n mesi?

Definizione ricorsiva della serie

I numeri di Fibonacci

- Modello a base di molte dinamiche evolutive delle popolazioni



$$F = \{f_0, \ldots, f_n\}$$

- $f_0 = 1$
- $f_1 = 1$

} casi base (sono 2!)

- Per $n > 1$, $f_n = f_{n-1} + f_{n-2}$

} 1 passo induttivo

Notazione "funzionale": $F(i) = f_i$

Esempio: Fibonacci

È una sequenza di numeri interi in cui ogni numero si ottiene sommando i due precedenti nella sequenza. I primi due numeri della sequenza sono per definizione pari ad 1.

- $f_1 = 1$ (caso base)
- $f_2 = 1$ (caso base)
- $f_n = f_{n-1} + f_{n-2}$ (passo ricorsivo)

```
function [fib]=FiboRic(n)
  if n==1 | n==2
      fib=1;
  else
      fib=FiboRic(n-2)+FiboRic(n-1);
  end
```

Problemi nell'uso della ricorsione

Uso della memoria

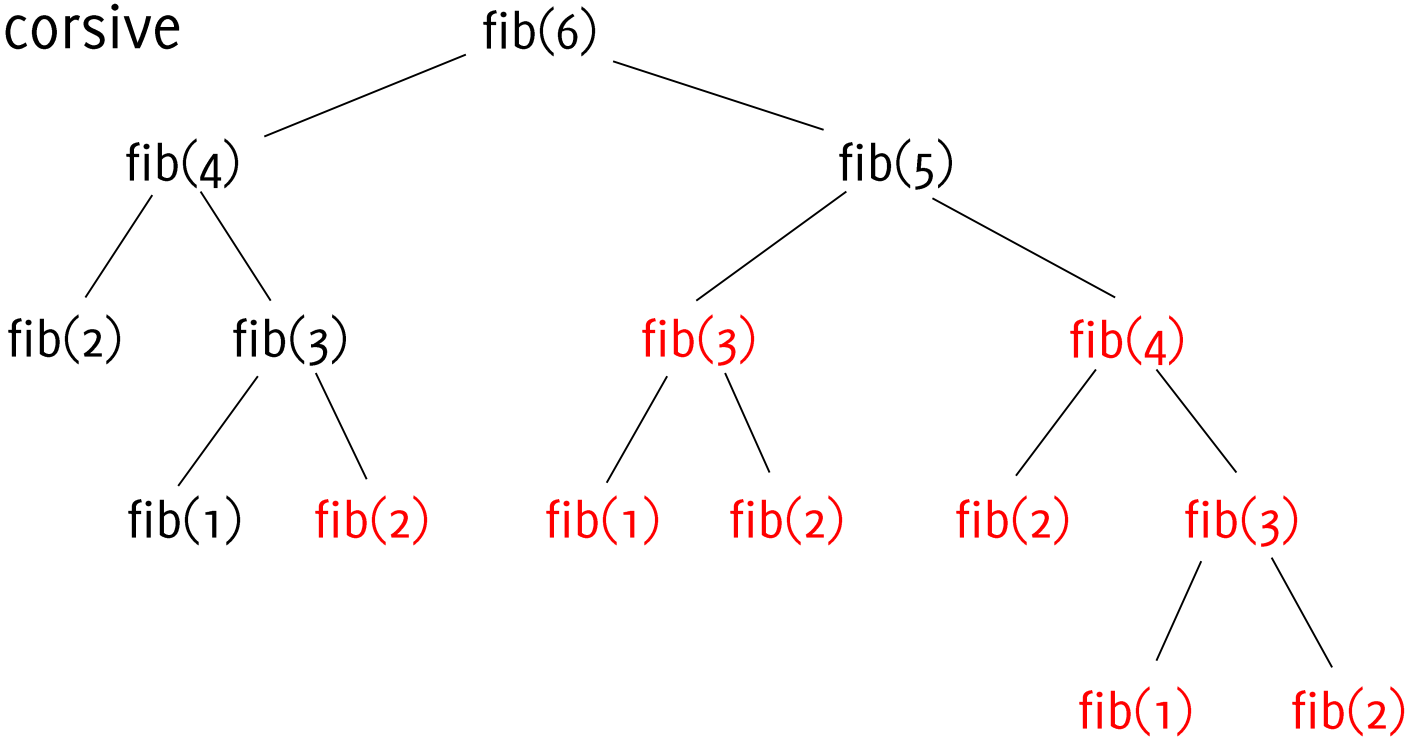
- La programmazione ricorsiva comporta spesso un uso inefficiente della memoria per la gestione degli spazi di lavoro delle chiamate generate
- In alcuni casi viene comunque preferita ad altri approcci per la sua eleganza e semplicità
- In altri casi, si può ricorrere ad implementazioni iterative
- Esempio

```
function [fl]=Fiblist(n)
  fl(1)=1;
  fl(2)=1;
  for k=3:n
      fl(k)=fl(k-2)+fl(k-1);
  end
```

Funzione iterativa che calcola i primi n numeri di fibonacci

Ricorsione Eccessiva

Soluzione elegante ma dispendiosa: numero esorbitante di chiamate ricorsive



Provate a far calcolare fib(5), poi fib(10), fib(15), fib(20), fib(25), fib(30), ...

Meglio usare una soluzione non ricorsiva...

TODO

Scrivere una funzione ricorsiva in Matlab che stampi a schermo tutti i valori del triangolo di Tartaglia per un certo ordine massimo N. i.e., e che restituisca al chiamante la riga N-sima

```
1                                   n = 0
1     1                             n = 1
1     2     1                       n = 2
1     3     3     1                 n = 3
1     4     6     4     1           n = 4
1     5    10    10     5     1     n = 5
1     6    15    20    15     6     1 n = 6
```

Le ninfee

Uno stagno è pieno di ninfee

- Ogni ninfea in 1 giorno si riproduce, generando un'altra ninfea
- Dopo 30 gg lo stagno è pieno

Quanto hanno impiegato le ninfee a riempire a metà stagno?

Le ninfee

Uno stagno è pieno di ninfee

- Ogni ninfea in 1 giorno si riproduce, generando un'altra ninfea
- Dopo 30 gg lo stagno è pieno

Quanto hanno impiegato le ninfee a riempire a metà stagno?

Scrivere una funzione ricorsiva per modellare

- una popolazione iniziale di *n* ninfee,
- che si riproduce ad un fattore *f* ogni giorno,

e per dire quanti giorni richiede per raggiungere una popolazione di *N* individui

Soluzione

```
function gg = stagno(n, f, N)
 % caso base: se ho già N ninfee nello stagno
% non devo aspettare alcun giorno
if (n >= N)
    gg = 0;
else
    % non abbiamo ancora N ninfee -> chiamata ricorsiva:
    % il numero di giorni necessari per coprire lo stagno è
    % uguale a: un giorno +
    % il numero di giorni che saranno necessari domani
    % (quando le ninfee saranno diventate n*f).
     gg = 1 + stagno(n * f, f, N);
end
```

Ricorsione:

Scrivere una **funzione ricorsiva** che prende in ingresso due numeri n e d che determina se n è una potenza di d

Soluzione

```
function [res , esp] = controllaSePotenzaDi(numero , base)
% caso base
if numero == base
    res = 1;
    esp = 1;
else
    if(mod(numero , base) == 0)
        %         esp = esp + 1; % da errore
        [res , esp] = controllaSePotenzaDi(numero/base , base);
        esp = esp + 1;
    else
        res = 0;
        esp = NaN;
    end
end
```

Altra soluzione

```
function [ris, esp] = controllaSePotenza(n , d)
% caso base
if(n == d)
    ris = 1;
    esp = 1;
elseif(n < d)
    ris =0;
    esp = NaN;
% interrompo se n non è intero
elseif(round(n) ~= n)
    ris = 0;
    esp = NaN;
else % chiamata ricorsiva
    [ris, esp] = controllaSePotenza(n/d , d);
    esp = esp +1;
end
```

Altra soluzione

```
function [ris, esp] = controllaSePotenza(n , d)
% caso base
if(n == d)
    ris = 1;
    esp = 1;
elseif(n < d)
    ris =0;
    esp = NaN;
% interrompo se n non è intero
elseif(round(n) ~= n)
    ris = 0;
    esp = NaN;
else % chiamata ricorsiva
    [ris, esp] = controllaSePotenza(n/d , d);
    esp = esp +1;
end
```

N.B.
res = controllaSePotenza(n,d*d);
è sbagliata perchè
la prima volta diventa d^2 la seconda volta d^4…

Interpretazione del codice

```
function r=cosafa(array)
k=size(array,2);

if (k == 1)
  r = 1;
elseif (k==2)
  if (array(1)+array(2)==10)
      r = 1;
  else
      r=0;
  end
else
  if (array(1)+array(k)==10)
      r = cosafa(array(2:k-1));
  else
      r=0;
  end
end
```

TDE Febbraio 2012

Si consideri la seguente funzione in Matlab

```
function [ris] = mistero(v, n)
if (n > 1)
    v2 = v(mod(v, n) ~= 0 | v == n);
    ris = mistero(v2, n-1);
else
    ris = v;
end
```

1. Dire qual è il contenuto di x dopo la seguente chiamata

```
x = mistero([2 3 4 5 7 9 11 13 15], 10)
```

Si consideri la seguente funzione in Matlab

```
function [ris] = mistero(v, n)
if (n > 1)
    v2 = v(mod(v, n) ~= 0 | v == n);
    ris = mistero(v2, n-1);
else
    ris = v;
end
```

Elementi di v alle posizioni dove v non è multiplo di n oppure dove v è uguale ad n

1. Dire qual è il contenuto di x dopo la seguente chiamata

```
x = mistero([2 3 4 5 7 9 11 13 15], 10)
```

Si consideri la seguente funzione in Matlab

```
function [ris] = mistero(v, n)
if (n > 1)
    v2 = v(mod(v, n) ~= 0 | v == n);
    ris = mistero(v2, n-1);
else
    ris = v;
end
```

Elementi di v alle posizioni dove v non è multiplo di n oppure dove v è uguale ad n

1. Dire qual è il contenuto di x dopo la seguente chiamata

```
x = mistero([2 3 4 5 7 9 11 13 15], 10)
```

```
x = [2 3 5 7 11 13]
```

2. Implementare un frammento di programma in linguaggio Matlab che legga da tastiera un numero intero positivo A. Quindi, chiami la funzione mistero passandole, come primo argomento, un vettore contenente tutti i numeri interi compresi fra 2 e A e, come secondo argomento, il numero A. Infine, stampi a video il valore ritornato dalla chiamata alla funzione mistero precedentemente descritta

```
A = input('Inserisci un numero intero positivo: ');
ris = mistero(2:A, A);
disp(ris);
```

2. Implementare un frammento di programma in linguaggio Matlab che legga da tastiera un numero intero positivo A. Quindi, chiami la funzione mistero passandole, come primo argomento, un vettore contenente tutti i numeri interi compresi fra 2 e A e, come secondo argomento, il numero A. Infine, stampi a video il valore ritornato dalla chiamata alla funzione mistero precedentemente descritta

3. Spiegare in maniera sintetica quale problema risolve il frammento di programma implementato al punto 2.

La funzione mantiene nel vettore v tutti i numeri che non sono strettamente divisibili per un intero minore o uguale ad n e maggiore di 1

TDE 25 Gennaio 2010

Un supermercato ha memorizzato il proprio archivio di scontrini nel file scontrini.mat, che contiene l'array struttura scontrini i cui elementi hanno i seguenti campi:
IDcliente: id numerico del cliente (>0)
totale: totale della spesa in Euro
punti: punti premio extra associati alle promozioni

Per ogni spesa, viene assegnato al cliente un quantitativo di punti premio pari alla somma dei punti più un ulteriore punto premio per ogni 10 euro spesi.
Scrivere uno script in MATLAB che legge il file scontrini.mat e costruisce un opportuno array struttura saldo, contenente per ciascun cliente, l'ID del cliente e il totale dei suoi punti premio.

Nota: Si faccia attenzione al fatto che un cliente può comparire in più di uno scontrino

Soluzione

```
load scontrini
clienti = [scontrini.IDcliente];
saldo .ID=-1;
saldo .punti=0;
n=1; %contatore clienti unici in saldo
for cc = 1 : length(clienti)
   if (~any([saldo.ID]==clienti(cc))
% considero il cliente solo se non è già presente nel saldo
      ii = find(clienti==clienti(cc));
% tutti gli scontrini riferiti a clienti(cc)
      pFagiolo = sum ([scontrini(ii).puntiFagiolo]);
% totale punti fagiolo
      puntiSpesa = sum(floor([scontrini(ii).totale]/10)));
%totale punti per ogni 10 euro di spesa
      saldo(n).ID = clienti(cc);
      saldo(n).punti = pFagiolo + puntiSpesa;
      n = n + 1;
   end
end
```

Mette in un vettore tutti gli elementi che compaiono nei campi Idcliente dell'array Di strutture scontrini

Inizializza l'array di strutture saldo

Trova le posizioni all'interno di scontrini che hanno un campo pari a clienti(cc)

Aggiungi l'elemento a saldo

Infine, si costruisca un array statistiche di tre elementi rispettivamente pari al numero di clienti che possiede meno di 1000 punti premio, il numero di clienti con più di 1000 punti ma meno di 5000 punti e infine il numero di clienti con più di 5000 punti.

```
…
statistica(1)=sum([saldo.punti]<=1000);
statistica(2)=sum([saldo.punti]>1000 & [saldo.punti]<=5000);
statistica(3)=sum([saldo.punti]>5000);
```

Funzioni Variabile

Function Handles

Variabili funzioni

Matlab permette di assegnare a variabili valori di tipo “funzione”

Un valore di tipo funzione può essere assegnato a una variabile (quindi passarlo come parametro), detta handle

l’hanlde è una variabile: quindi può essere utilizzato per trasmettere una funzione ad un’altra funzione

l’hanlde è una funzione: quindi all’handle possono essere passati alcuni argomenti per valutare la funzione

Assegnamento di un valore di tipo funzione

Handle di una funzione esistente

```
f = @nome_funzione
```

- Esempio

```
>> seno=@sin
seno = @sin
>> seno(pi/2)
ans = 1
```

```
f = @(x,y…)‹expr›
```

Handle di una funzione definita ex-novo

- x,y,… sono i parametri della funzione
- ‹expr› è un'espressione che calcola il valore della funzione
- Esempio

```
>> sq=@(x)x^2
sq = @(x)x^2
>> sq(8)
ans =  64
```

Esempi

```
% definizione della funzione inline
h = @(x) -x
% valutazione della funzione h nei punti 2 e 9
h(2);
h(9);

% g che viene definita in maniera tale da operare su vettori
g = @(x)x.^2
g(2)
g([2 : 2])
g([-2 : 2])

% è possibile "comporre" le funzioni (si da l'output di g come input di h)
h(g([-2 : 2]))
% NB: non è possibile sommare i function hanldes
g + h
```

Funzioni di Ordine Superiore

Funzioni i cui parametri sono function handles

Funzioni di ordine superiore

Se un parametro di una funzione **f** è un handle (cioè contiene un valore di tipo funzione) allora **f** è una **funzione di ordine superiore**

L'handle passato come parametro consente ad **f** di invocare la funzione nell'hanlde

Esempi

```
% creo una funzione per fare quello che fa g
function y = elevaAlQuadrato(x)
y = x.^2;

elevaAlQuadrato(8)

% faccio una funzione di ordine superiore
per valutare se una funzione (passata come argomento mediante un
function handle ) è idempotente in un punto x
% se f(f(x)) == f(x)
function res = controllaSeIdempotente(f , x)
if ( f(f(x)) == f(x))
    res = 1;
else
    res = 0;
end
```

Esempi

```
% chiamata della funzione di ordine superiore
controllaSeIdempotente(g , 1)

% ma non funziona se passo la funzione elevaAlQuadrato
controllaSeIdempotente(elevaAlQuadrato , 1)
Error using elevaAlQuadrato (line 2)
Not enough input arguments.

% occorre creare un function handle per elevaAlQuadrato
controllaSeIdempotente(@elevaAlQuadrato, 1)
% oppure
controllaSeIdempotente(@(x)elevaAlQuadrato(x) , 2)

% il nome delle variabili utilizzate per definire function handle non
conta
k=@(asd) elevaAlQuadrato(asd)
controllaSeIdempotente(k , 2)
```

Funzioni di ordine superiore

Esempio: funzione map che applica una funzione f a tutti gli elementi conetenuti nel parametro vin e ritorna i risultati in vout

```
function [vout]=map(f, vin)
    for ii=1:length(vin)
        vout(ii)=f(vin(ii));
    end;
```

handle

Invoca la funzione passata come argomento

```
>> A=[1,2,3,4,5,6];
>> map(sq,A)
ans = 1  4  9  16  25  36
```

A cosa serve la funzione map?

Non tutte le funzioni possono essere applicate a vettori

Si prenda -- ad esempio– la funzione per vedere se un numero è primo (la si ottiene facilmente a partire dai codici sviluppati nella prima parte del corso)

Come facciamo ad applicare la funzione a tutti gli elementi del vettore

- Scriviamo un ciclo esplicito
- Chiamiamo map(@controllaSePrimo, vett)

Esempio: funzione accumulatore

Funzione accumulatore: `[x] =acc(f, a, u)`

- f è una funzione con due argomenti, con elemento neutro u
- f viene applicata in maniera *cumulativa* a tutti gli elementi di a nel seguente modo:
- applico f ad a(1) e all'elemento neutro, f(u, a(1)),
- Passo al secondo elemento e applico f al risultato dell'operazione precedente e con a(2), i.e, f(f(u,a(1)), a(2)) ..
- Fino a f(…f(f(f(u,a(1)),a(2)),a(3))…,a(length(a)))

```
function [x]=acc(f, a, u)
    x=u;
    for ii=1:length(a)
        x=f(x, a(ii));
    end
```

Esempio: funzione accumulatore

Funzione sommatoria: `function [s]=sommatoria(a)`

- calcola la sommatoria degli elementi di a

```
function [s]=sommatoria(a)
    som=@(x,y)x+y;
    s=acc(som,a,0);
```

- calcola il prodotto degli elementi di a

```
function [s]=produttoria(a)
    prd=@(x,y)x*y;
    s=acc(prd,a,1);
```

- Sono le alternative a sum e prod (built in) e alle implementazioni con cicli tipo

```
p = 1;
for ii = 1 : numel(x)
   p= p * x(ii);
end
```

Questa è la variabile che funziona da accumulatore… inizializzata all'elemento neutro

Esempio

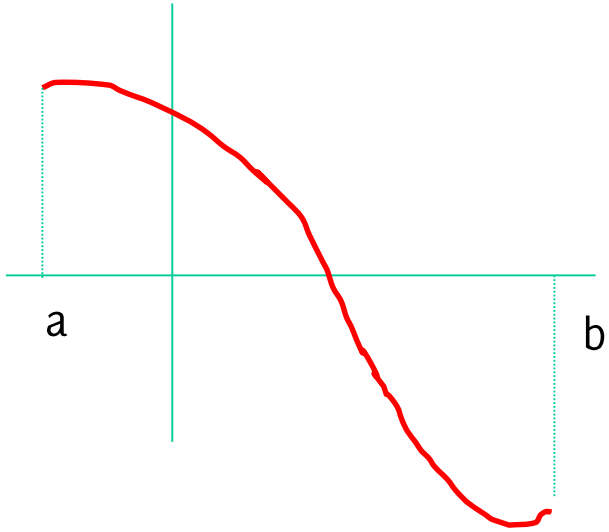
Scrivere una funzione ricorsiva *palindroma* che controlla se una stringa è palindroma

Esempio dal teorema di Bolzano

Sia f una funzione reale e continua in $[a, b]$ per cui

$$f(a) * f(b) \leq 0$$

allora estiste almeno un punto $c \in [a, b]$ tale che $f(c) = 0$.



Usare questa teorema per scrivere una funzione haUnoZero che prende in ingresso un function handle ad f, gli estremi dell'intervallo e determina se la funzione ha uno zero nell'intervallo

TODO:

Utilizzando il metodo di bisezione e il teorema di Bolzano, scrivere una funzione **ricorsiva calcolaZeri** che calcola gli zeri di una funzione analitica f (passata in un function handle) nell' intervallo continuo $[a, b]$ (gli estremi sono passati come parametri)

Si consideri come criterio di arresto invece di $f(x) == 0$

$|f(x)| < \epsilon$ dove ϵ viene passato come parametro.

Funzioni Built in per Visualizzazione

Qualche altra informazione

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.

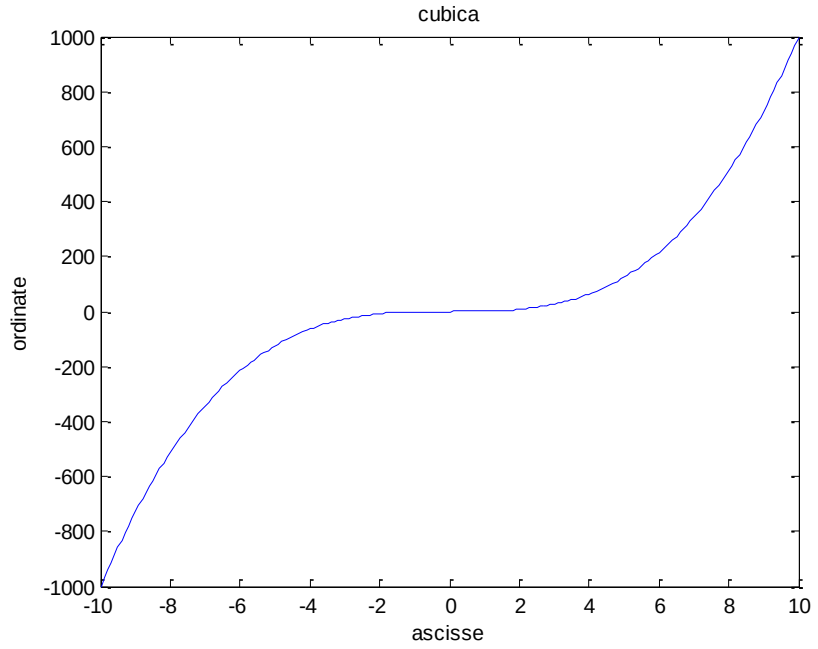
In `plot(x,y)`, `x` e `y` devono essere **due vettori aventi le stesse dimensioni**

E' possibile specificare diversi elementi grafici (help plot per una lista delle opzioni)

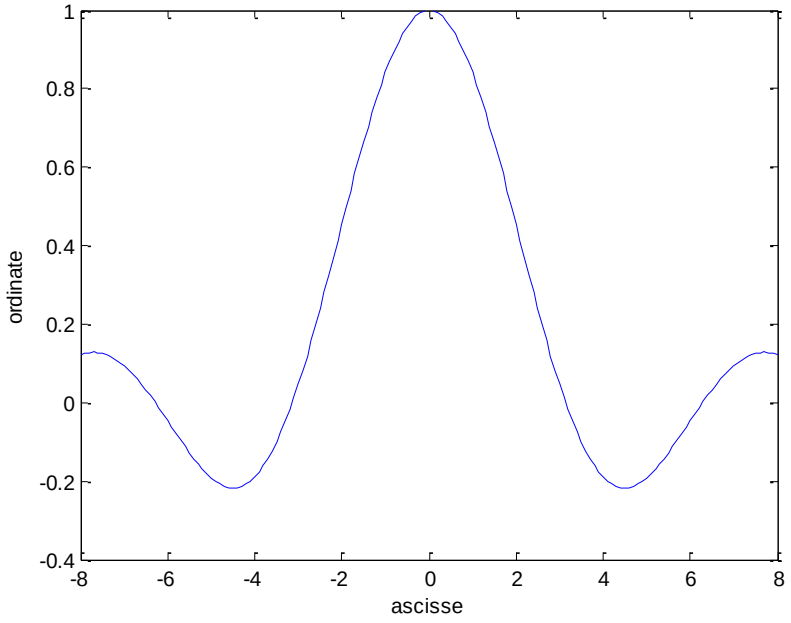
Le funzioni `xlabel` visualizzano una stringa come nome asse ascisse, `ylabel` per ordinate, `title` per il titolo

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?

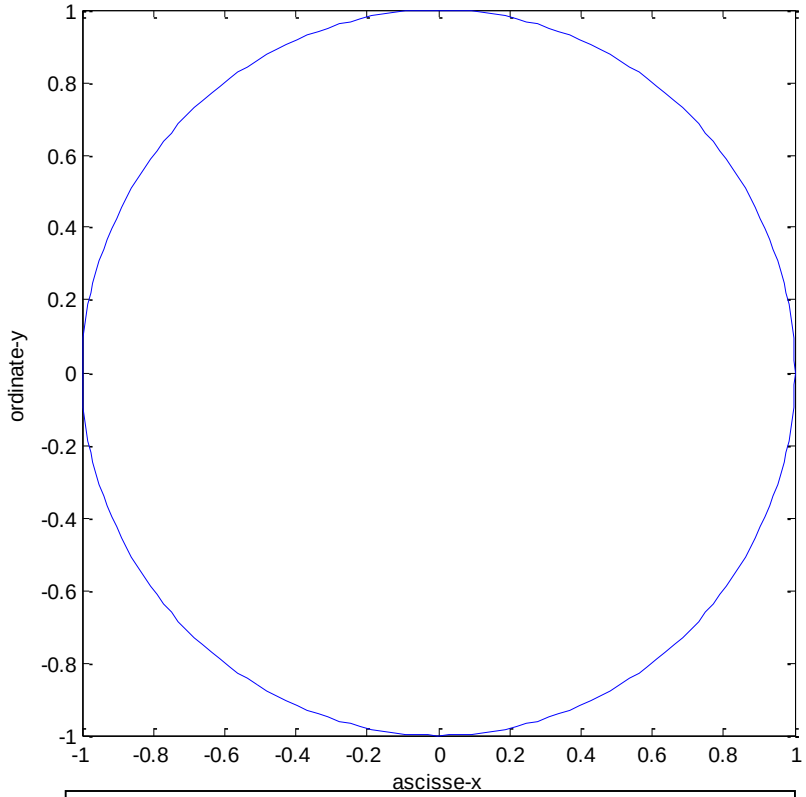
```
>> t=[0:pi/100:2*pi];
>> x=cos(t);
>> y=sin(t);
>> plot(x,y);
>> xlabel('ascisse-x');
>> ylabel('ordinate-y');
```

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

Esempi

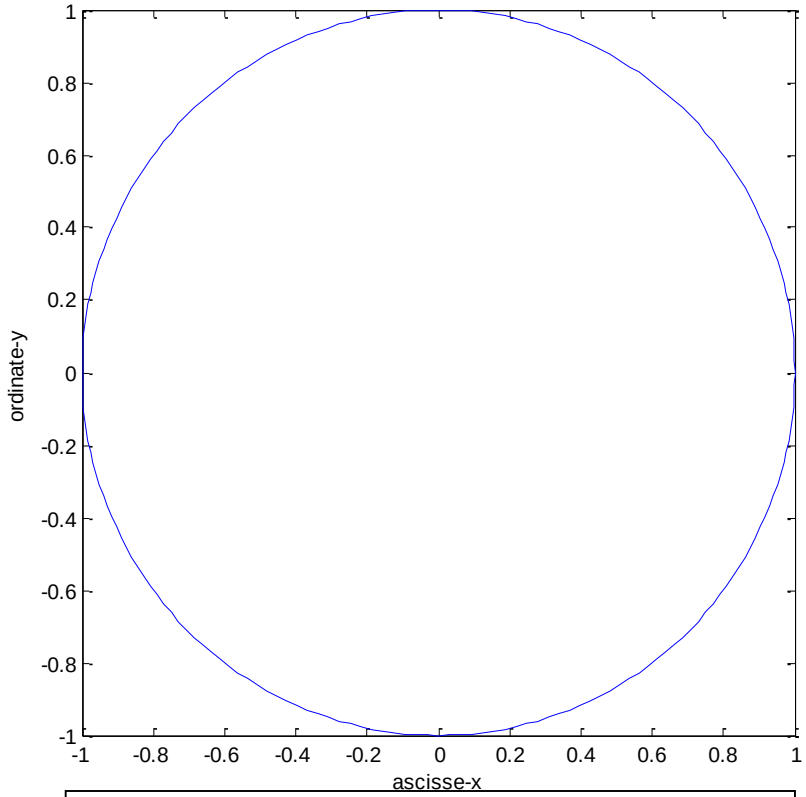
```
>> t=[0:pi/100:2*pi];
>> x=cos(t);
>> y=sin(t);
>> plot(x,y);
>> xlabel('ascisse-x');
>> ylabel('ordinate-y');
```

Esempi




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

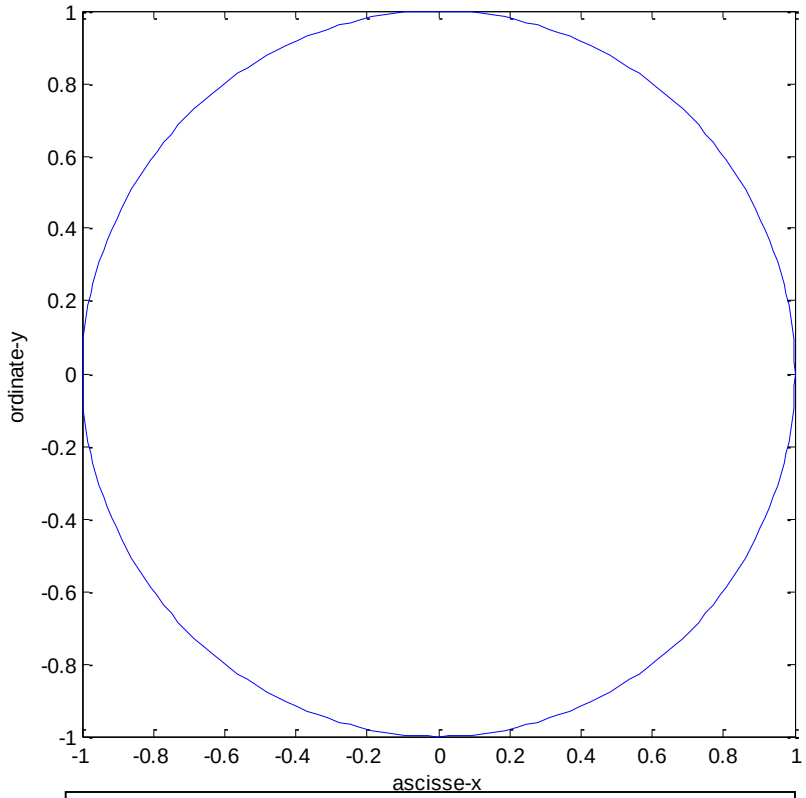
Esempi



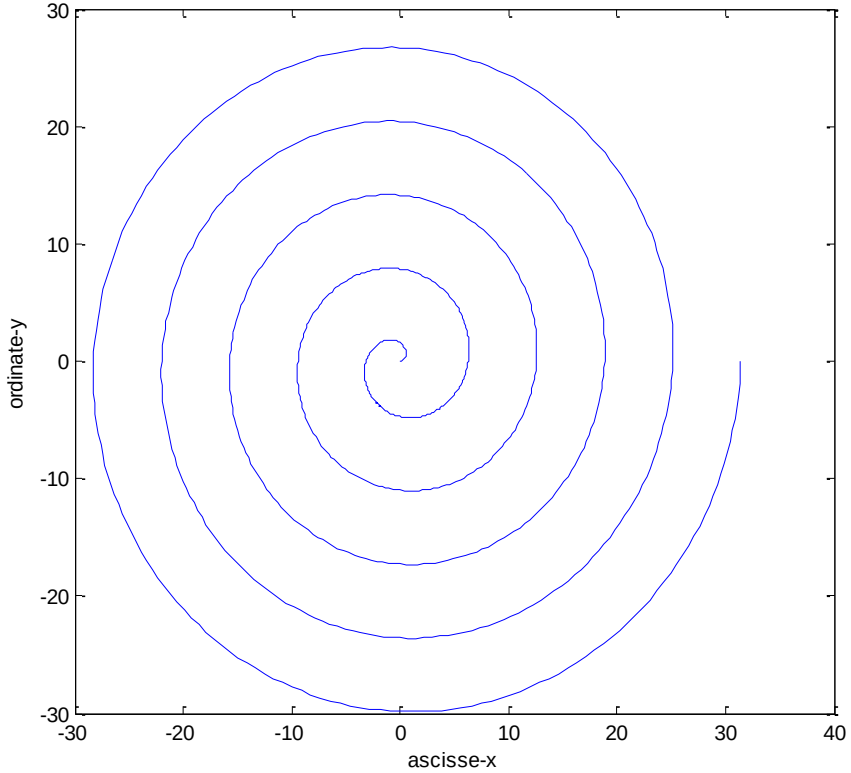

```
>> t=[0:pi/100:2*pi];
>> x=cos(t);
>> y=sin(t);
>> plot(x,y);
>> xlabel('ascisse-x');
>> ylabel('ordinate-y');
```

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

Esempi




```
>> 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

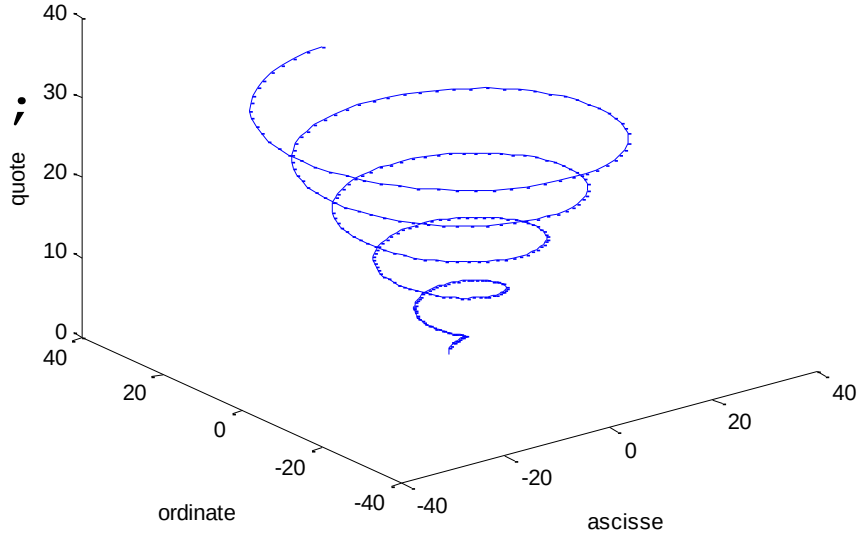
Generalizzazione del diagramma a due dimensione: insieme di terne di coordinate

`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');
```

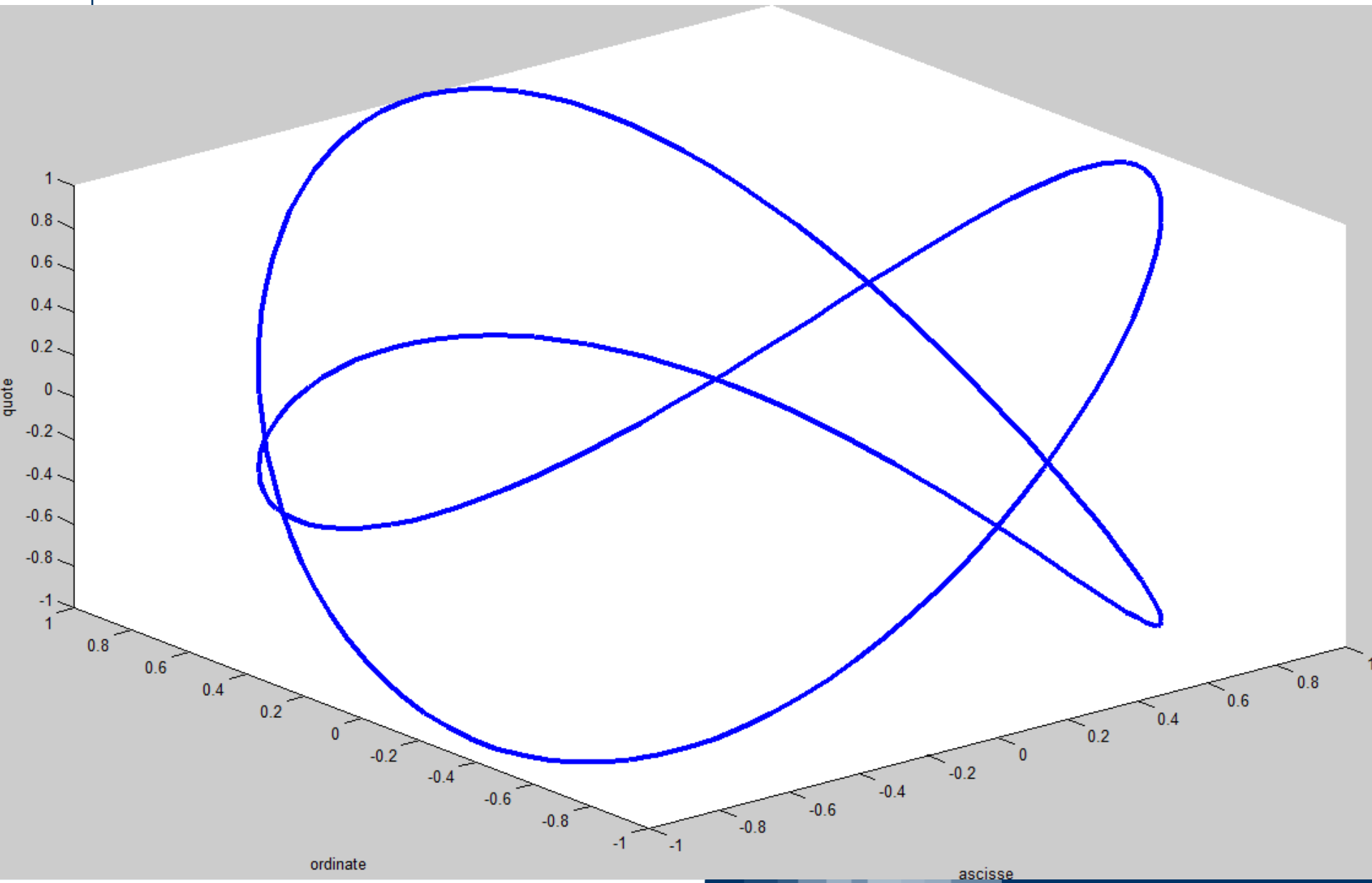


Linspace(a,b,n):
crea un vettore di n punti equispaziati tra a e b

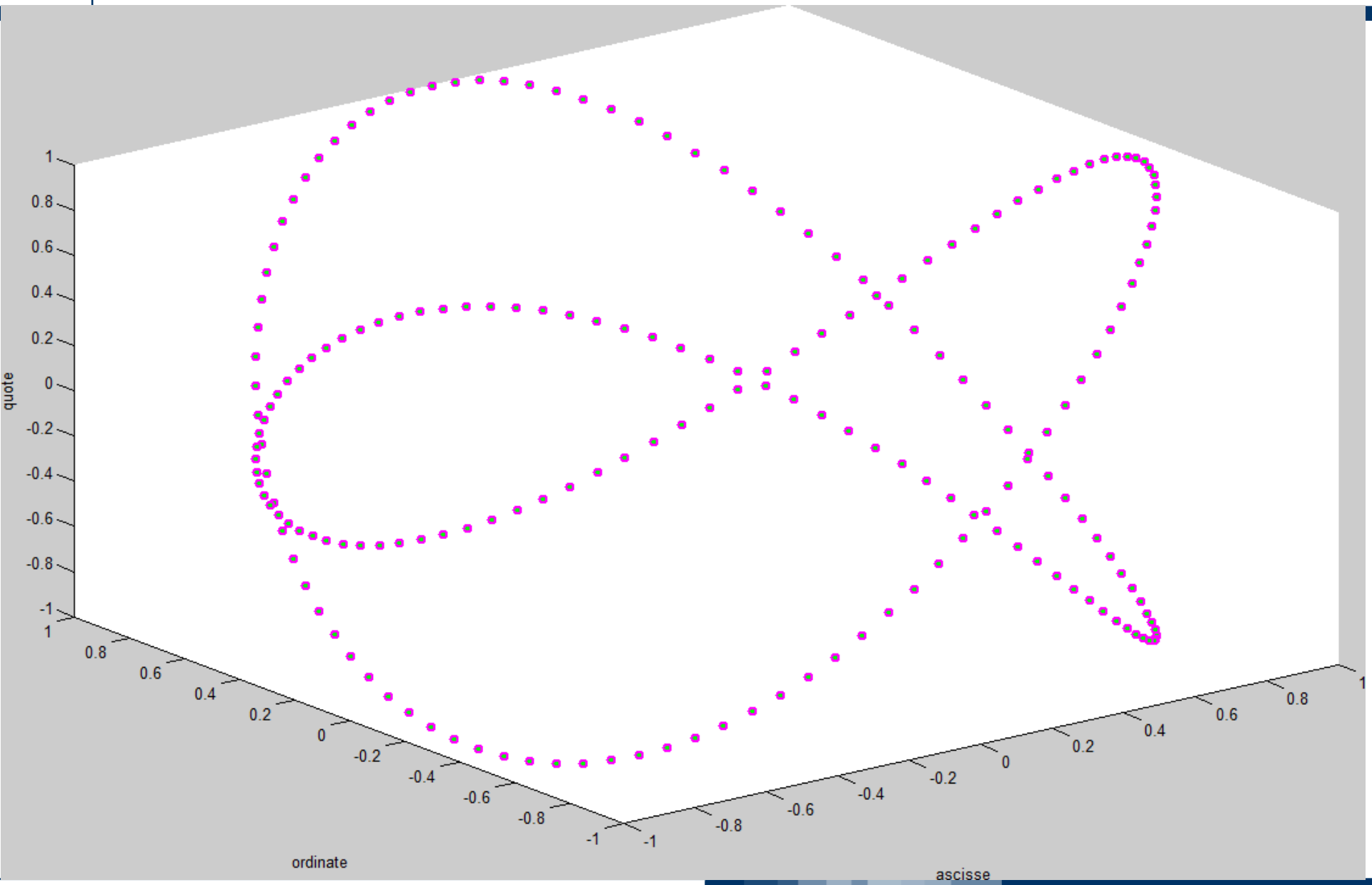
plot restituisce un hanlde, una variabile di riferimento per poter accedere nuovamente all'insieme di punti disegnato

Set(plot_handle, 'Property Name', PropertyVal)
permette di modificare

```
t = linspace(0,4*pi ,200);
plot_hnd = plot3(sin(t),cos(t),cos(3/2 *t))
set(plot_hnd, 'LineWidth', 3)
xlabel('ascisse');
ylabel('ordinate');
zlabel('quote');
```


quote
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
ordinate
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
ascisse
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8

```
t = linspace(0,4*pi ,200);
plot_hnd = plot3(sin(t),cos(t),cos(3/2 *t))
set(plot_hnd, 'LineWidth', 3)
xlabel('ascisse');
ylabel('ordinate');
zlabel('quote');

set(plot_hnd ,'LineStyle','none','Marker','o',
'MarkerFaceColor', [0 1 0],...
'MarkerEdgeColor',[1 0 1],...
'MarkerSize',5,'LineWidth' ,1.5)
```


quote
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
ordinate
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
ascisse
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

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

```
[xx, yy] = meshgrid([-3 : 3], [-4 : 4]);
```

xx =

-3	-2	-1	0	1	2	3
-3	-2	-1	0	1	2	3
-3	-2	-1	0	1	2	3
-3	-2	-1	0	1	2	3
-3	-2	-1	0	1	2	3
-3	-2	-1	0	1	2	3
-3	-2	-1	0	1	2	3
-3	-2	-1	0	1	2	3
-3	-2	-1	0	1	2	3

yy =

-4	-4	-4	-4	-4	-4	-4
-3	-3	-3	-3	-3	-3	-3
-2	-2	-2	-2	-2	-2	-2
-1	-1	-1	-1	-1	-1	-1
0	0	0	0	0	0	0
1	1	1	1	1	1	1
2	2	2	2	2	2	2
3	3	3	3	3	3	3
4	4	4	4	4	4	4

È possibile quindi valutare una funzione di queste due matrici, e.g., zz = xx + yy , e disegnarla mediante mesh

Superfici: esempi

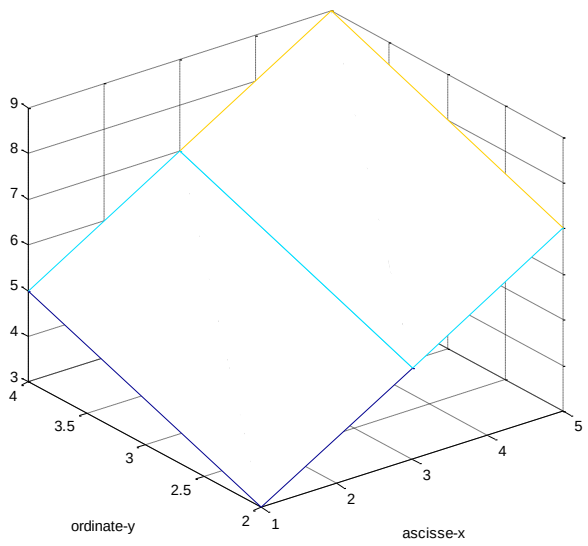
```
% Disegnamo z=x+y
x=[1, 3, 5];
y=[2, 4];
[xx, yy] = meshgrid(x, y);
zz = xx + yy;
mesh(xx, yy, zz);
xlabel('ascisse-x'); ylabel('ordinate-y');
```

```
>> 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)

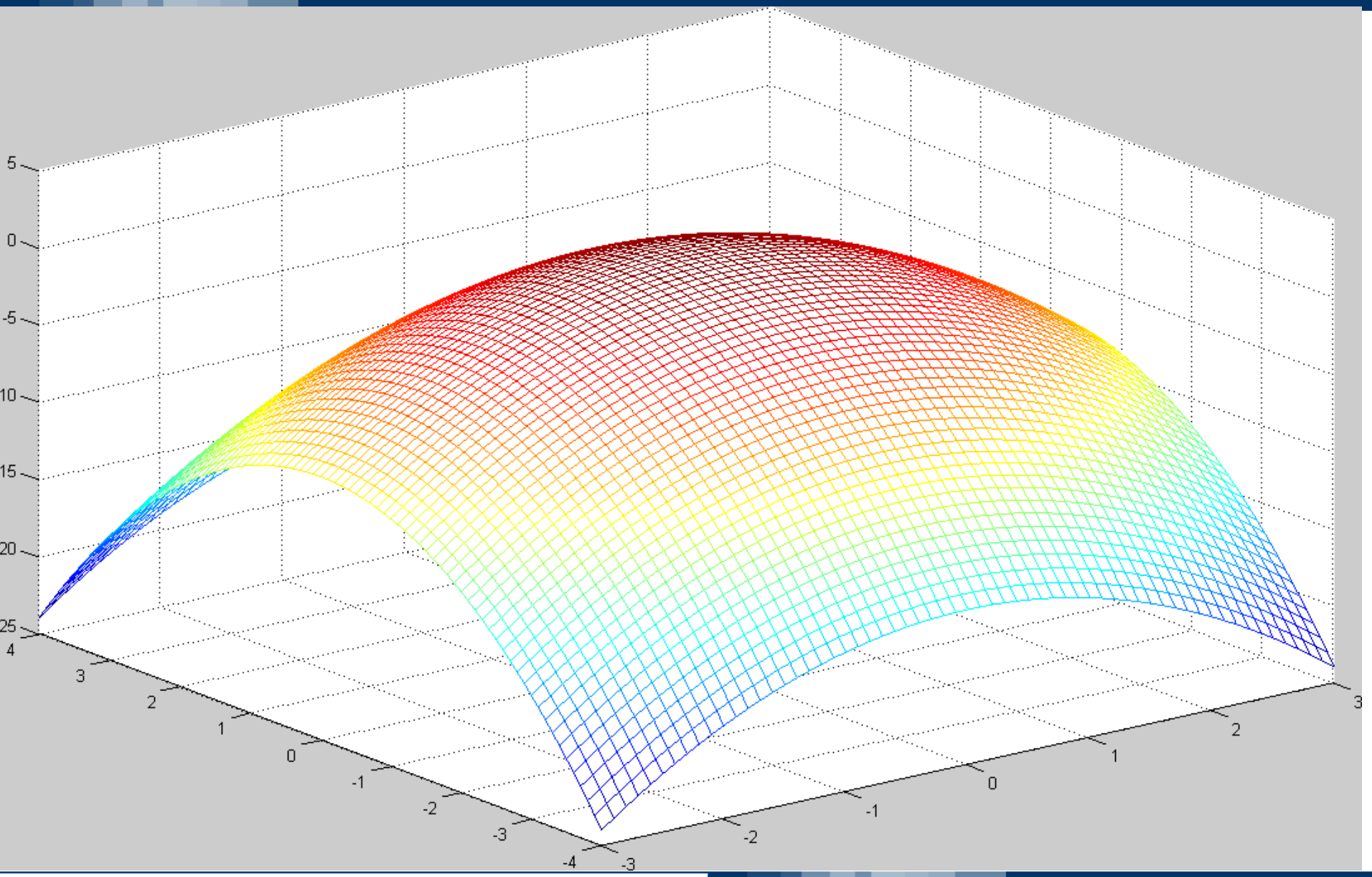
(1,2,3) (3,2,5) (5,2,7)
(1,4,5) (3,4,7) (5,4,9)

(NB: z=x+y)

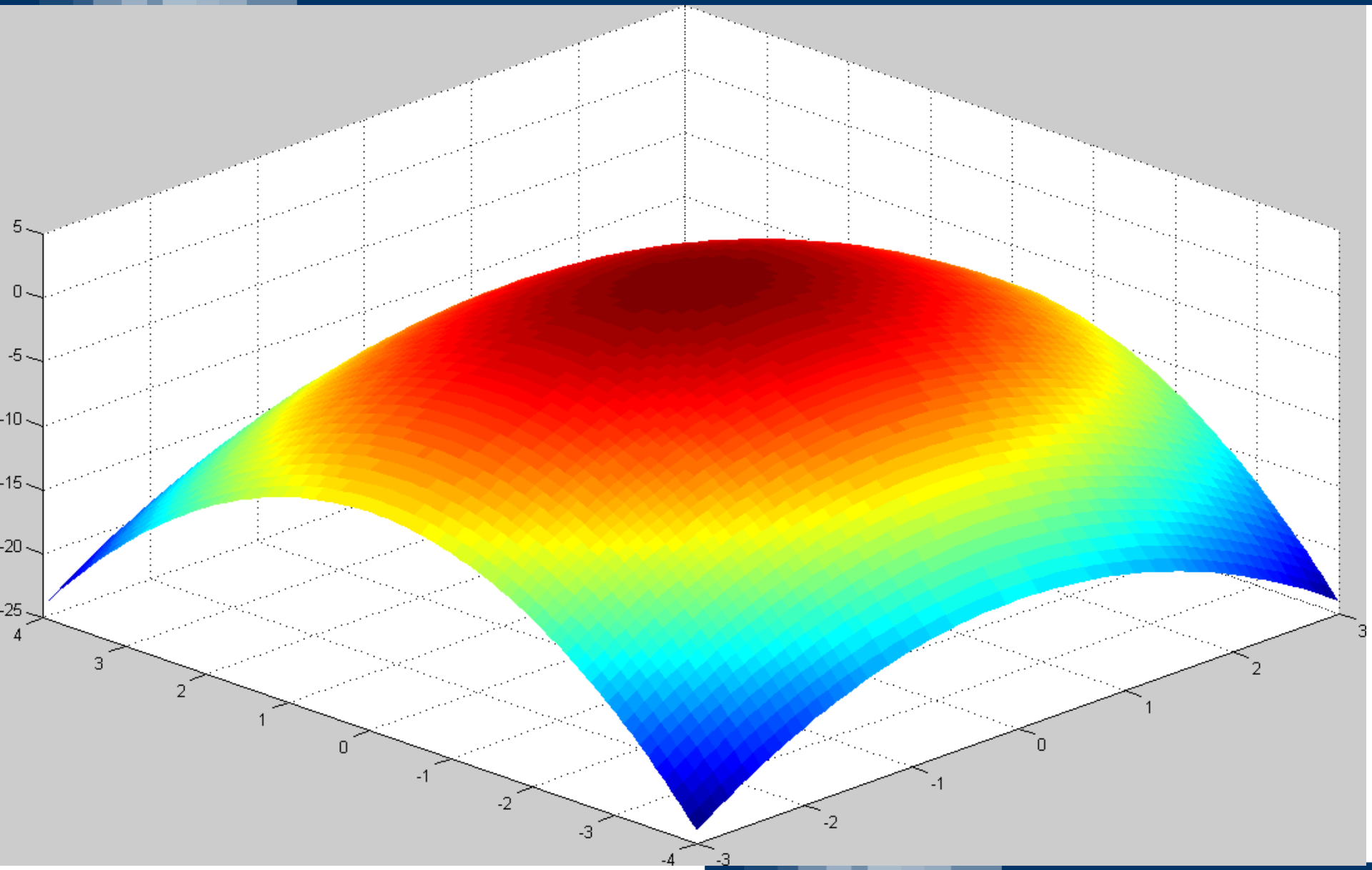
Mesh

```
[xx, yy] = meshgrid([-3 : 0.1 :3], [-4 : 0.1 :4]);
f = @(x, y)(1 - x.^2 - y.^2);

figure,
aa = mesh(xx, yy, f(xx, yy))
```

Mesh unisce i punti con delle linee colorate. Di default il colore indica il valore della quota

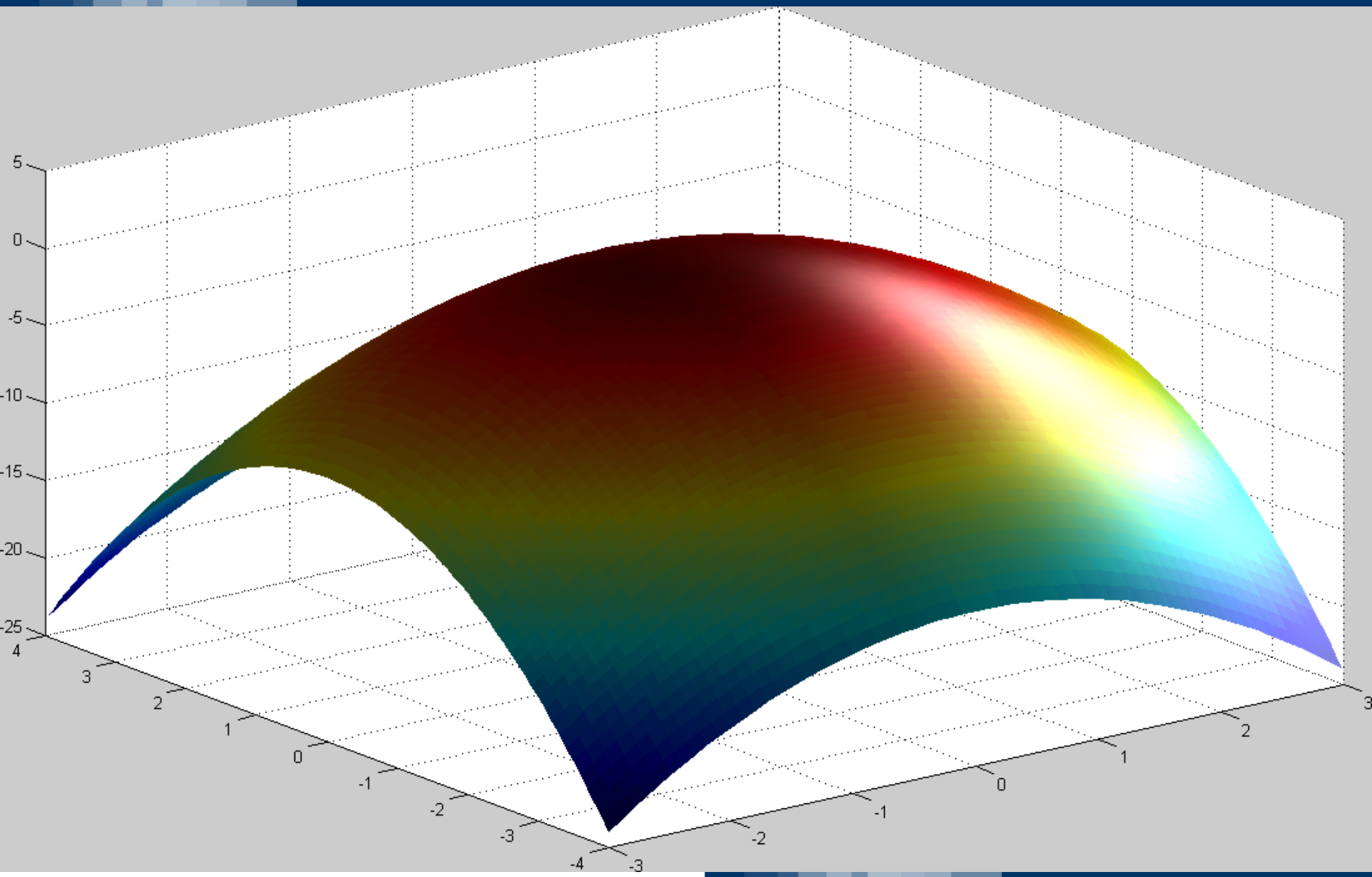

5
0
-5
-10
-15
-20
-25
4
3
2
1
0
-1
-2
-3
-4
-3
-2
-1
0
1
2
3

```
[xx, yy] = meshgrid([-3 : 0.1 :3], [-4 : 0.1 :4]);
f = @(x, y)(1 - x.^2 - y.^2);

figure,
aa = surf(xx, yy, f(xx, yy))
set(aa, 'EdgeColor', 'none')
```


5
0
-5
-10
-15
-20
-25
4
3
2
1
0
-1
-2
-3
-4
-3
-2
-1
0
1
2
3

```
[xx, yy] = meshgrid([-3 : 0.1 :3], [-4 : 0.1 :4]);
f = @(x, y)(1 - x.^2 - y.^2);


figure,
aa = surf(xx, yy, f(xx, yy))
set(aa, 'EdgeColor', 'none')
light % aggiunge una sorgente luminosa per il rendering
```

Surf riempie le regioni tra le linee con del colore che, di default, dipende dal valore della quota


5
0
-5
-10
-15
-20
-25
4
3
2
1
0
-1
-2
-3
-4
-3
-2
-1
0
1
2
3

Hold on

Le superfici vengono visualizzate su un grafico 3D.

È quindi possibile aggiungere degli elementi in sovraimpressione utilizzando la funzione

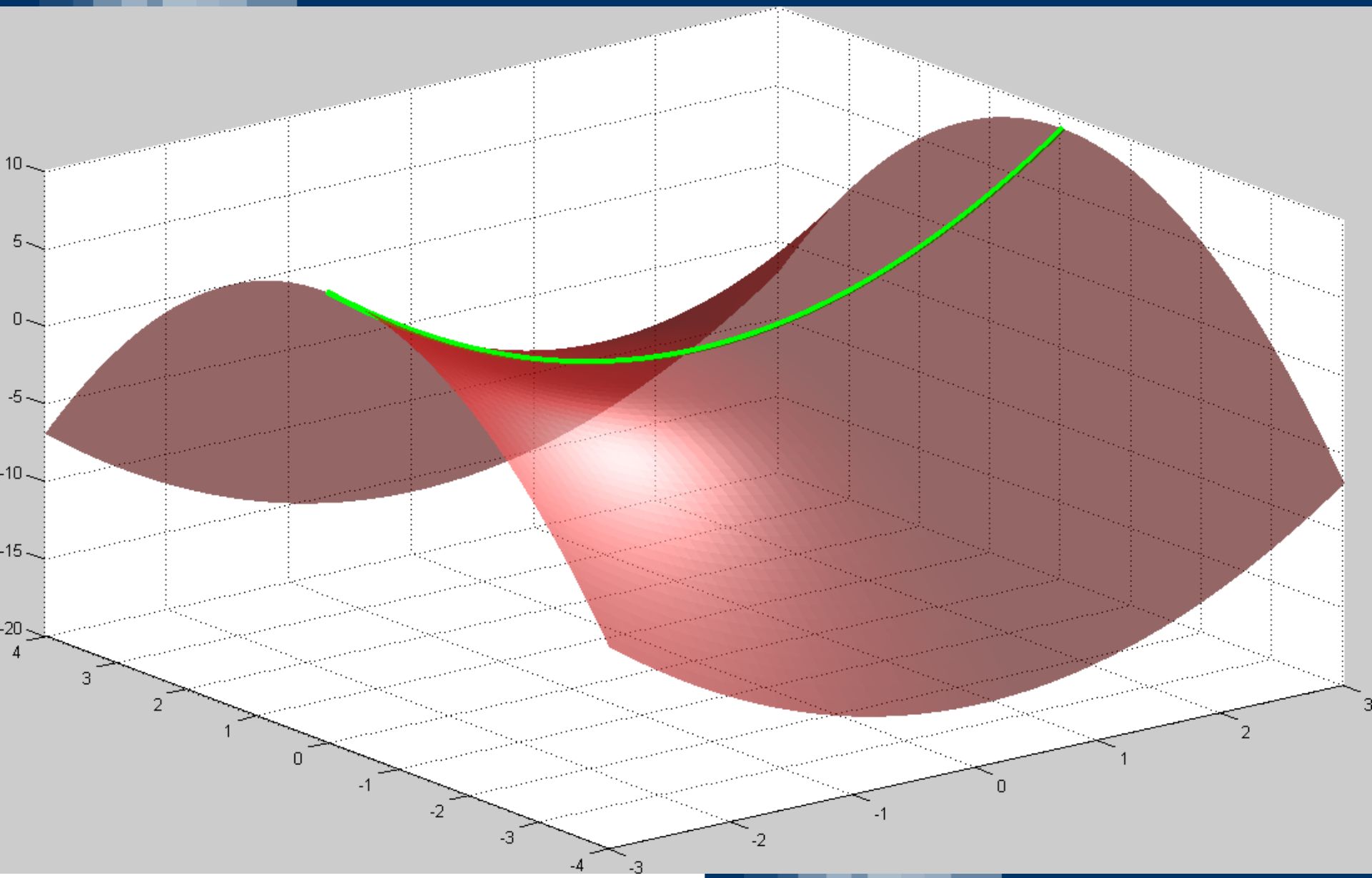
- plot3(), mesh(),altre funzioni grafiche quali surf() etc..

- Per sovrascrivere ad un grafico usare la funzione hold on e hold off quando si ha terminato

Esempio, disegnare in sovraimpressione alla quadrica

$$z = x^2 - y^2 \quad \text{la curva} \begin{cases} z = x^2 \\ y = 0 \end{cases}$$

```
[xx, yy] = meshgrid([-3 : 0.1 :3], [-4 : 0.1 :4]);
f = @(x, y)(x.^2 - y.^2);
figure(),
aa = surf(xx, yy, f(xx, yy))
hold on
x = xx(1, :);
y = zeros(size(x));
bb = plot3(x, y, f(x,y), 'g-')
set(aa, 'EdgeColor', 'none', 'FaceColor', 'red', 'FaceAlpha', 0.6)
set(bb, 'Linewidth', 3)
light
hold off
```


10
5
0
-5
-10
-15
-20
4
3
2
1
0
-1
-2
-3
-4
-3
-2
-1
0
1
2
3

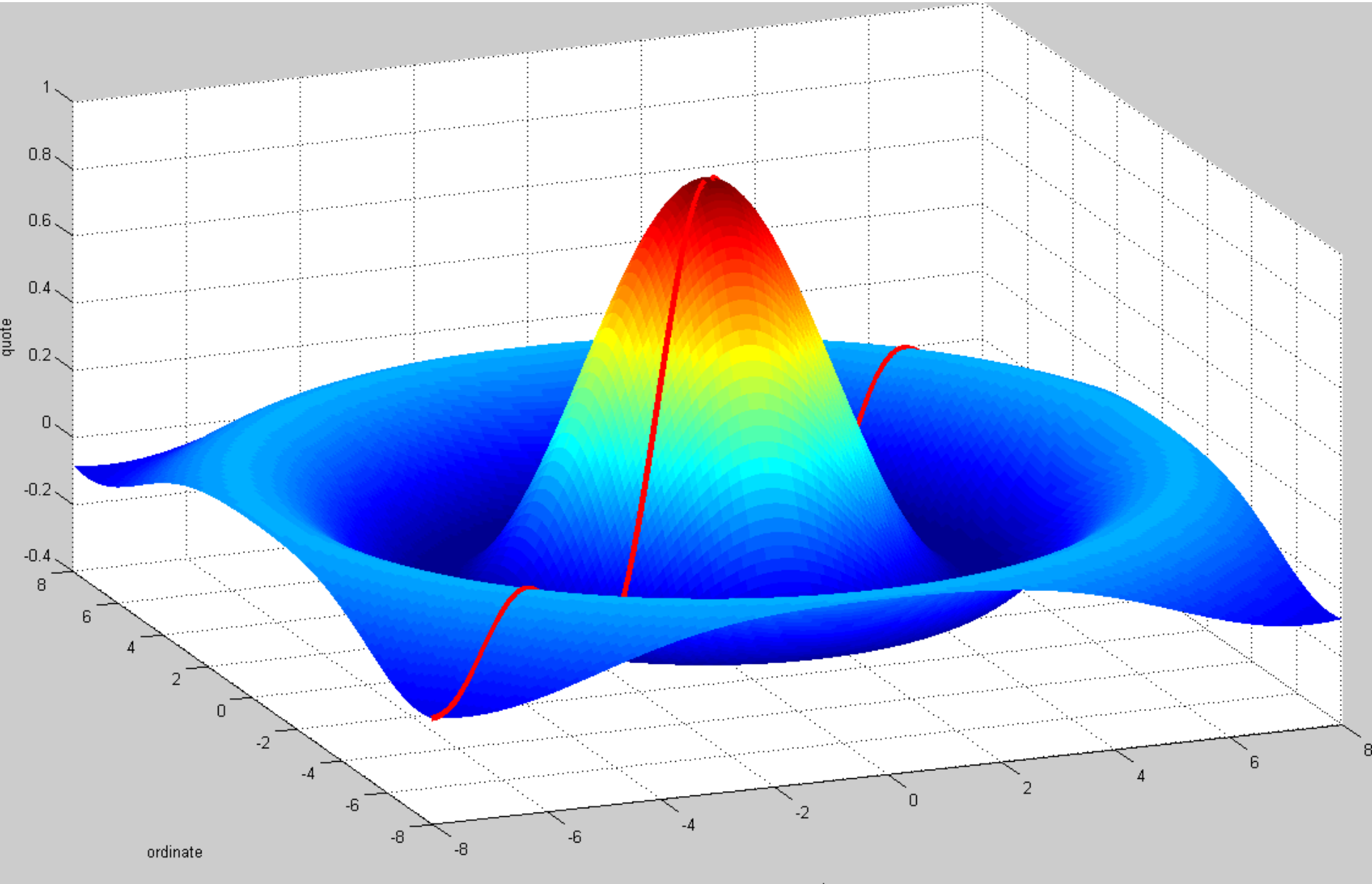
Disegnare la funzione

$$z = \frac{\sin\left(\sqrt{x^2 + y^2}\right)}{\sqrt{x^2 + y^2}}$$

e una curva su questa funzione passante per l'origine

```
tx=[-8:0.1:8];
ty=tx;
[xx, yy] = meshgrid (tx, ty);
f = @(x,y)(sin(sqrt(x.^2 + y.^2)) ./ sqrt(x.^2 + y.^2));
figure,
aa = surf(xx, yy, f(xx, yy));
hold on
bb = plot3(tx, tx, f(tx, tx), 'r-' ,'LineWidth', 3)
set(aa, 'EdgeColor', 'none')
```

Superfici: esempi (3)



Funzioni per Stringhe

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
```

Funzioni di I/O su Files

Lettura e scrittura di dati su file

Formati di file gestiti

- ascii = file di testo
- .mat = file proprietari di Matlab

Comandi più semplici da usare

- save
- load

Salvataggio dei dati su file (1)

funzione **save** per formato .mat

- save filename: salva su filename.mat tutte le variabili contenute nel workspace
- save('filename', 'array1' ,'array2'): salva su filename.mat le variabili array1 e array2

I file .mat hanno un formato compatto

Contengono

- Nomi, tipi e valori di ogni variabile
- La dimensione degli array
- … in generale, tutto ciò che serve per **ripristinare** lo stato dello spazio di lavoro
- Possono essere portati da un computer all'altro, anche con sistemi operativi diversi

Salvataggio dei dati su file (2)

Limitazione dei file .mat

- E` un formato proprietario di MATLAB.
- Non è utilizzabile per leggere/scrivere dati con un altro programma (e.g., editor di testi, excel)

Uso dei file di testo, specificando il formato ascii,

save(outputfileName, varNames,.., format)

- x = [1.23 3.14 6.28; -5.1 7.00 0];
- save('filename.dat', 'x' , '-ascii');

- Produce il file filename.dat organizzato come

```
1.2300000e+000    3.1400000e+000    6.2800000e+000
-5.1000000e+000   7.0000000e+000    0.0000000e+000
```

Nota: si può usare qualsiasi estensione per questi file, è buona norma distinguerli dai file .mat

Acquisizione dati da file

funzione **load**: carica I dati da file (formato mat o ascii) nel workspace corrente

- load filename: carica nello spazio di lavoro tutte le variabili nel file
- load filename x y: carica nello spazio di lavoro solo le variabili x ed y
- S = load(filename); carica il contenuto di filename in una struttura chiamata S
- Se filename non ha estensione o ha estensione .mat, viene trattato come un file .mat
- File ascii
  - load filename.dat: crea una variabile di nome filename che conterrà i dati in filename.dat
  - Il file deve contenere dati separati da virgole o spazi