Strutture e Funzioni Built-In

Informatica B AA 2021/2022

Luca Cassano

luca.cassano@polimi.it

2 Dicembre 2021

Funzioni Built-In

Alcune funzioni built in per gestire array

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 maggiore della matrice
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

Alcune funzioni built in per gestire array

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)	Restituisce il risultato della divisione intera m/n
round(x)	approssima x all'intero più vicino
rand(N)	Restituisce una matrice di NxN numeri casuali con distribuzione uniforme tra 0,1

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

```
>> b = [4 7 2 6 5]
b =  4     7     2     6
>> 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

sum(vettore) calcola la somma di tutti gli elementi di vettore

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

Calcolo dimensione array

- 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: length su matrici restituisce la dimensione avente più elementi. In pratica length(A) == max(size(A))

Funzioni Logiche Built in

Funzioni Logiche

Nome	Elemento restituito
all(x)	un vettore riga, con un elemento per ogni colonna della matrice x. Ogni elemento del risultato vale 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 un elemento per ogni colonna della matrice x. Ogni elemento del risultato vale 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

Altre Funzioni Logiche: find

La funzione **`find`** permette di cercare all'interno di un vettore

```
a = [5 6 7 2 10]
find(a>5) -> ans = 2 3 5
```

Nota: **find restituisce** gli **indici** e **non estrae un sottovettore** (come invece posso fare utilizzando vettori di interi o vettori logici come indici di un vettore)

```
x = [5, -3, 0, 0, 8];
y = [2, 4, 0, 5, 7];
values = y(x&y) -> values = [2 4 7]
indexes = find(x&y) -> indexes = [1 2 5]
values = y(x<=y) -> values = [4 0 5]
indexes = find(x<=y) -> indexes = [2 3 4]
```

Altre Funzioni Logiche: find

```
indx = find(x)
```

(che equivale a `indx = find(x > 0)`)

restituisce gli indici degli elementi non nulli dell'array `x`.

Esempio

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

Esempio (senza find())

```
function [pres, pos] = cerca(x, vett)
    p=0; pos=[];
    for i=1:length(vett)
        if vett(i)==x
            p=p+1;
            pos(p)=i;
        end
    end
    pres=p>0;
end
```

Esempio (senza find())

```
function [pres, pos] = cerca(x, vett)
    p=0; pos=[];
    for i=1:length(vett)
        if vett(i)==x
            p=p+1;
            pos(p)=i;
        end
    end
    pres=p>0;
end
```

```
>> A=[1, 2, 3, 4, 3, 4, 5, 4, 5, 6]
A = 1 2 3 4 3 4 5 4 5 6
>> [p, i]=cerca(4,A)
p = 1
i = 4 6 8
```

Esempio (con find())

```
function [pres, pos] = cerca2(x, vett)
    pres = 1;
    pos = find(vett == x);
    if isempty(pos)
        pres = 0;
    end
end
```

Esempio (con any() e find())

```
function [pres, pos] = cerca3(x, vett)
   pres = any(vett == x);
   pos = find(vett == x);
end
```

Esempio

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**

Esempio

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**

```
function [closest, pos_closest] = closestVal(vett, val)
    diff = vett - val;
    abs_diff = abs(diff);
    [~ , pos_closest] = min(abs_diff);
    closest = vett(pos_closest);
    closest = unique(closest); % takes unique values
end
```

La funzione min ritorna due argomenti:
il valore minimo e le posizioni in cui questo compare.
A me serve solo il secondo argomento,
quindi scarto il primo inserendo la ~ al momento della chiamata

Esempio Importante

Scrivere una funzione che prende in ingresso un vettore e rimuove tutti i valori uguali a 7

Invocare la funzione sul seguente vettore v = [12, 4, 7, 14]

Stampare il risultato.

Esempio Importante

Scrivere una funzione che prende in ingresso un vettore e rimuove tutti i valori uguali a 7

Invocare la funzione sul seguente vettore v = [12, 4, 7, 14]

Stampare il risultato.

```
function rimuovi7(vettore)
      vettore(vettore == 7) = [];
end

>> v = [12, 4, 7, 14]
>> v =
>>      12   4   7   14
>> rimuovi7(v)
```

Esempio Importante

Scrivere una funzione che prende in ingresso un vettore e rimuove tutti i valori uguali a 7

Invocare la funzione sul seguente vettore v = [12, 4, 7, 14]

Stampare il risultato.

```
function rimuovi7(vettore)
      vettore(vettore == 7) = [];
end

>> v = [12, 4, 7, 14]
>> v =
>>      12   4   7   14
>> rimuovi7(v)
>> v =
>>      12   4   7   14
```

Esempio Importante

Scrivere una funzione che prende in ingresso un vettore e rimuove tutti i valori uguali a 7

Invocare la funzione sul seguente vettore v = [12, 4, 7, 14]

Stampare il risultato.

```
function rimuovi7(vettore)
      vettore(vettore == 7) = [];
end
```

```
>> v = [12, 4, 7, 14]
>> v =
>>      12   4   7   14
>> rimuovi7(v)
>> v =
>>      12   4   7   14
```

Non c'è modo di modificare una variabile nel workspace principale all'interno del corpo di una funzione.
Workspace locale e principale sono separati.
Occorre sovrascrivere!

Esempio Importante

Scrivere una funzione che prende in ingresso un vettore e rimuove tutti i valori uguali a 7

Invocare la funzione sul seguente vettore v = [12, 4, 7, 14]

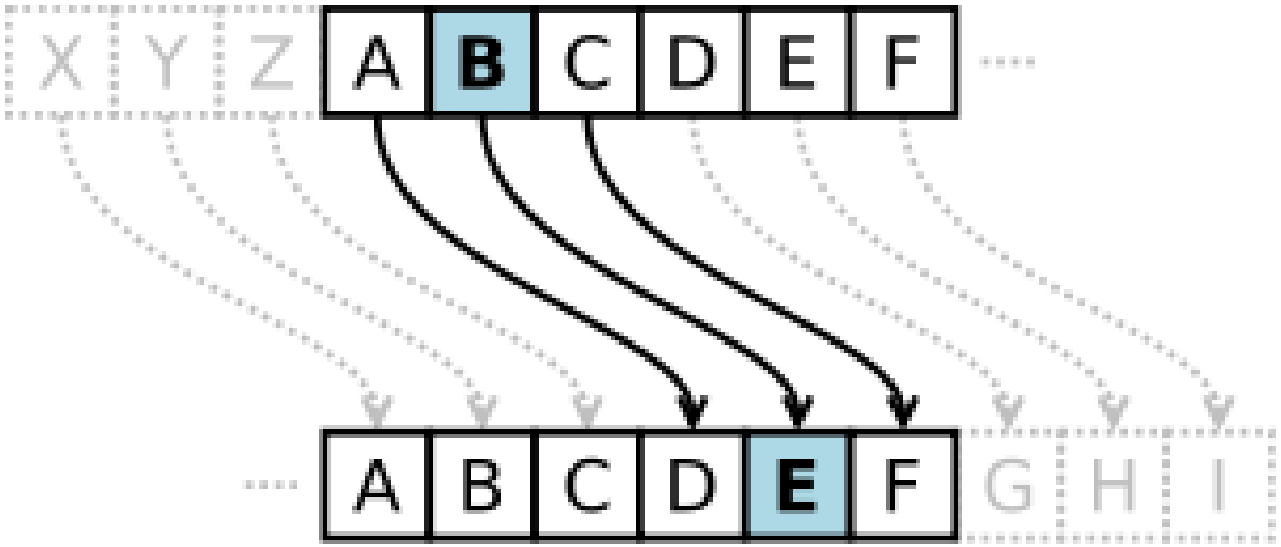
Stampare il risultato.

```
function vettore = rimuovi7(vettore)
      vettore(vettore == 7) = [];
end

>> v = [12, 4, 7, 14]
>> v =
>>      12   4   7   14
>> v = rimuovi7(v)
>> v =
>>      12   4   14
```

Esercizio Cifrario di Cesare

Scrivere un programma che esegue la codifica di un testo utilizzando il cifrario di Cesare

Assicurarsi che la funzione sia in grado di eseguire anche la decodifica



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 0 ,1 **(è diverso dal C)**

Funzioni per Stringhe

Esiste la funzione di **confronto**

```
TF = strcmp(str1 , str2)
```

- INPUT: `str1`, `str2` stringhe da confrontare
- OUTPUT: TF valore booleano 0 ,1 **(è diverso dal C)**
- Similmente `strcmpi(str1, str2)` non fa differenze tra maiuscole e minuscole

Funzioni per Stringhe

Esiste la funzione di **confronto**

```
TF = strcmp(str1 , str2)
```

- INPUT: `str1`, `str2` stringhe da confrontare
- OUTPUT: TF valore booleano 0 ,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`).

Funzioni per Stringhe

Non occorre `strlen` (si usa length o size)

Non occorre `strcpy` (la copia tra stringhe è nativa in Matlab)

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`** vettore contenente gli indici di tutte le occorrenze di **`PATTERN`** in **`TEXT`** (vuoto se non ce ne sono)

Esempio STRFIND

```
>> s = 'Ei fu siccome immobile, dato il mortal sospiro';
>> v = strfind(s, 'fu')
>> v =
>>     4
>>
>> v = strfind(s, 'immobile')
>> v =
>>     15
```

Funzioni I/O

Operazioni di I/O

- La funzione **`input`** apre una finestra di dialogo con l'utente e permette di inserire generiche istruzioni Matlab

Operazioni di I/O

- La funzione **`input`** apre una finestra di dialogo con l'utente e permette di inserire generiche istruzioni Matlab
- **`a = input(txtToShow)`** visualizza **`txtToShow`** nella command window attende una generica istruzione Matlab. I valori inseriti vengono assegnati ad **`a`** al termine dell'esecuzione

Operazioni di I/O

- La funzione **`input`** apre una finestra di dialogo con l'utente e permette di inserire generiche istruzioni Matlab
- **`a = input(txtToShow)`** visualizza **`txtToShow`** nella command window attende una generica istruzione Matlab. I valori inseriti vengono assegnati ad **`a`** al termine dell'esecuzione
- **`a = input(txtToShow, 's')`** visualizza **`txtToShow`** nella command window e attende l'inserimento di una stringa. Non occorre usare apici quindi nell'inserimento.

Operazioni di I/O

- La funzione **`input`** apre una finestra di dialogo con l'utente e permette di inserire generiche istruzioni Matlab
- **`a = input(txtToShow)`** visualizza **`txtToShow`** nella command window attende una generica istruzione Matlab. I valori inseriti vengono assegnati ad **`a`** al termine dell'esecuzione
- **`a = input(txtToShow, 's')`** visualizza **`txtToShow`** nella command window e attende l'inserimento di una stringa. Non occorre usare apici quindi nell'inserimento.
- La funzione **`num2str(A)`** trasforma la matrice **`A`** in ingresso in una rappresentazione di stringa.

Operazioni di I/O

- La funzione **`input`** apre una finestra di dialogo con l'utente e permette di inserire generiche istruzioni Matlab
- **`a = input(txtToShow)`** visualizza **`txtToShow`** nella command window attende una generica istruzione Matlab. I valori inseriti vengono assegnati ad **`a`** al termine dell'esecuzione
- **`a = input(txtToShow, 's')`** visualizza **`txtToShow`** nella command window e attende l'inserimento di una stringa. Non occorre usare apici quindi nell'inserimento.
- La funzione **`num2str(A)`** trasforma la matrice **`A`** in ingresso in una rappresentazione di stringa.
- Permette di comporre stringhe contenenti il risultato di un'esecuzione

Stampa dei risultati (1)

- I risultati di un'operazione sono mostrati immediatamente se non si inserisce il ;
- **`disp`**
  - Stampa sia stringhe che numeri/vettori/matrici
  - viene usato in congiunzione con num2str per stampare sia numeri che stringhe

Lettura e scrittura di dati su file

- Formati di file gestiti
  - ascii = file di testo (.dat)
  - .mat = file proprietari di Matlab

Lettura e scrittura di dati su file

- Formati di file gestiti
  - ascii = file di testo (.dat)
  - .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

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

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 contengono
  - Nomi, tipi e valori di ogni variabile
  - La dimensione degli array
  - … in generale, tutto ciò che serve per **ripristinare** lo stato del workspace
  - 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)

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)

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');
```

Acquisizione dati da file

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

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 .mat

Acquisizione dati da file

- funzione **`load`**: carica l dati da file (formato mat o ascii) nel workspace corrente
  - **`load`**(filename): carica nello spazio di lavoro tutte le variabili nel file .mat
  - **`load`**(filename, x, y): carica nello spazio di lavoro solo le variabili x ed y contenute nel file .mat

Acquisizione dati da file

- funzione **`load`**: carica l dati da file (formato mat o ascii) nel workspace corrente
  - **`load`**(filename): carica nello spazio di lavoro tutte le variabili nel file .mat
  - **`load`**(filename, x, y): carica nello spazio di lavoro solo le variabili x ed y contenute nel file .mat
  - **`load`**(filename, '-ascii'): carica nello spazio di lavoro tutte le variabili nel file .dat
  - **`load`**(filename, x, y, '-ascii'): carica nello spazio di lavoro solo le variabili x ed y contenute nel file .dat

Acquisizione dati da file

- 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

Funzioni Built in per Visualizzazione

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

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.

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**

Diagrammi a due dimensioni

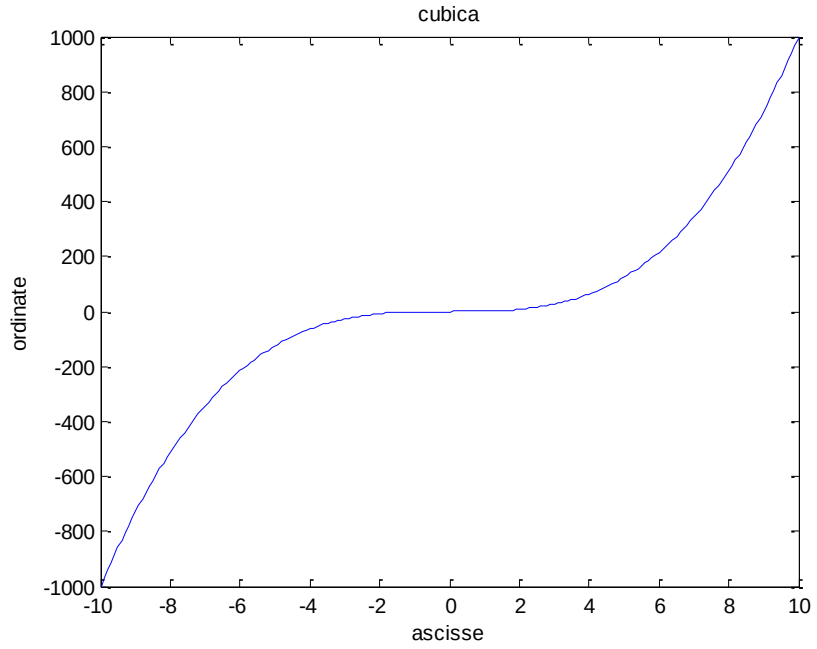
- La funzione `plot(x,y)` disegna il **diagramma** cartesiano dei punti che hanno valori delle ascisse nel vettore x, delle ordinate nel vettore y
- Il diagramma è l'insieme di **coppie di punti** [x(1), y(1)],…, [x(end),y(end)] rappresentanti le coordinate dei punti del piano cartesiano
  - La funzione `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**
- Le funzioni `xlabel`, `ylabel`, `title` visualizzano una stringa come nome asse ascisse, ordinate e titolo

Diagrammi a due dimensioni: esempi

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

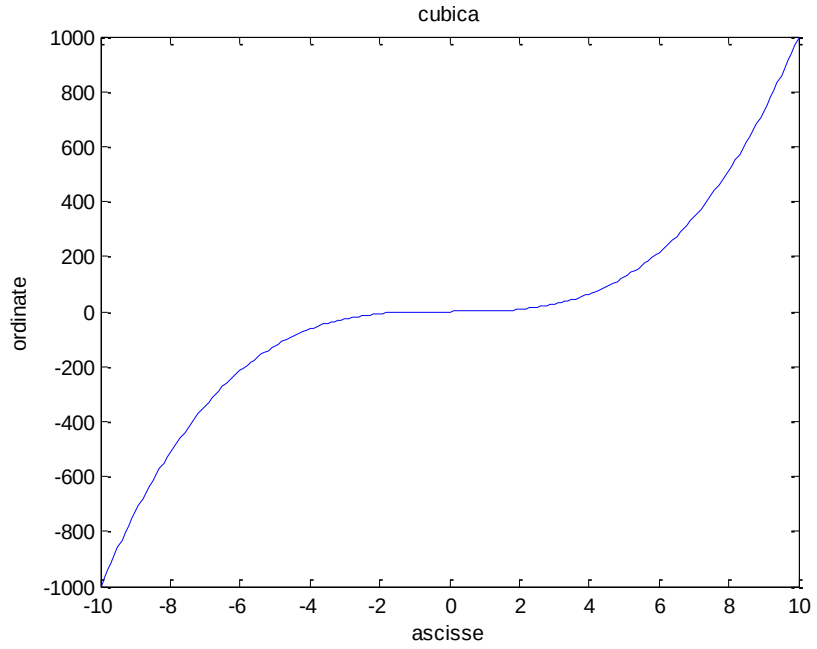
Diagrammi a due dimensioni: esempi

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



Diagrammi a due dimensioni: esempi

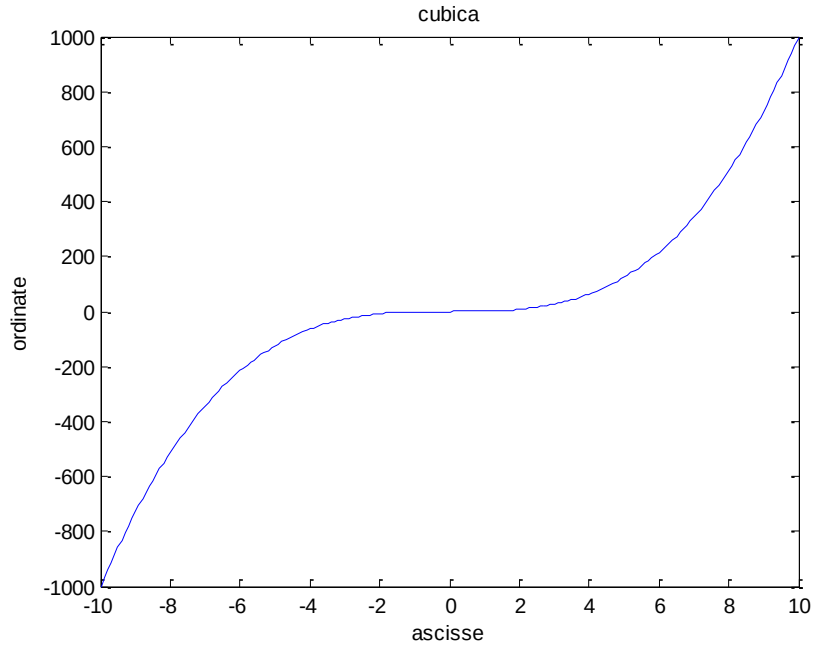
```
>> x = -10:0.1:10;
>> y=x.^3;
>> plot(x,y);
>> xlabel('ascisse');
>> ylabel('ordinate');
>> title('cubica');
```



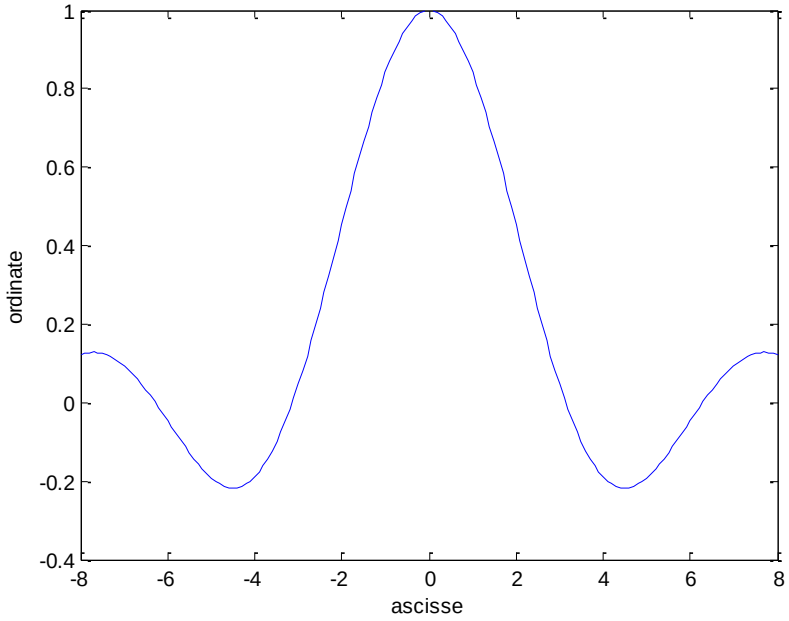

```
>> x=[-8:0.1:8];
>> y= sin (x) ./ x;
>> plot(x, y);
>> xlabel('ascisse');
>> ylabel('ordinate');
```

Diagrammi a due dimensioni: 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: NOTA BENE

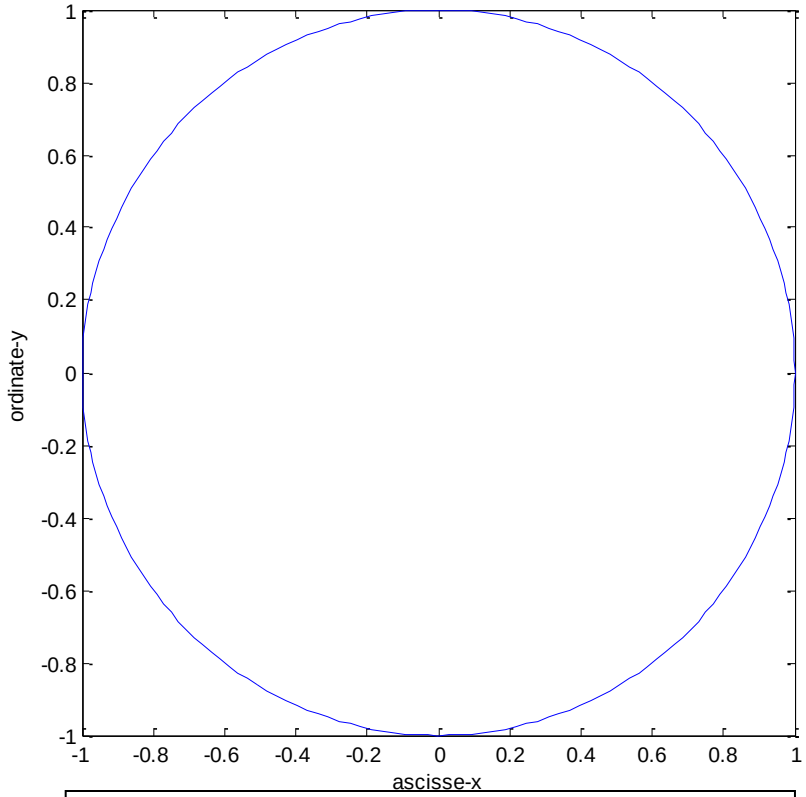
- Un diagramma è semplicemente una sequenza ordinata di punti, di coppie di coordinate cartesiane

- 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.

Esempi

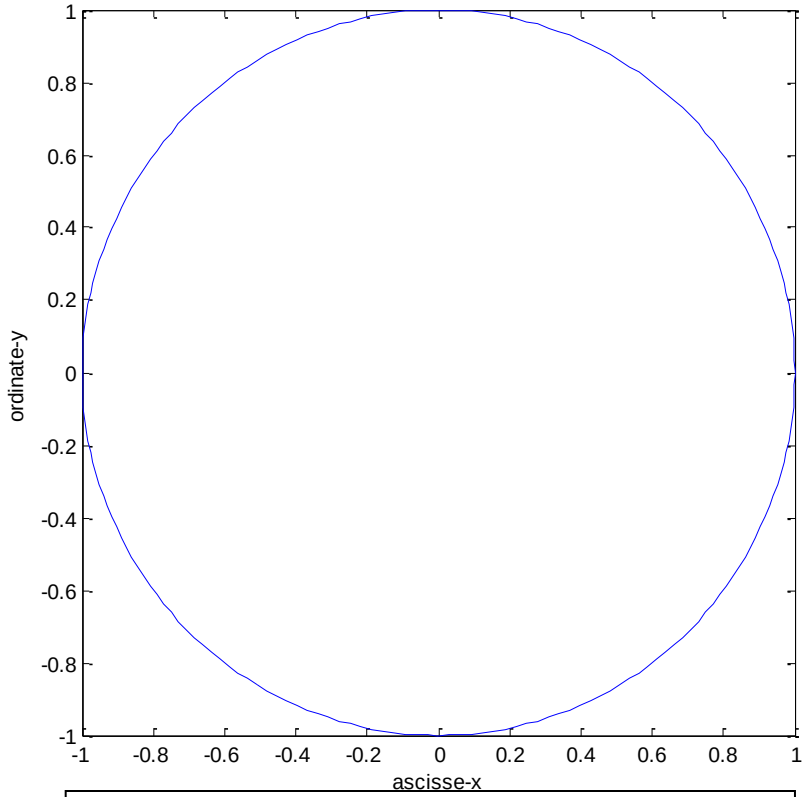
```
>> t=[0:pi/100:2*pi];
>> x=cos(t);
>> y=sin(t);
>> plot(x,y);
>> xlabel('ascisse-x');
>> ylabel('ordinate-y');
```

Esempi




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

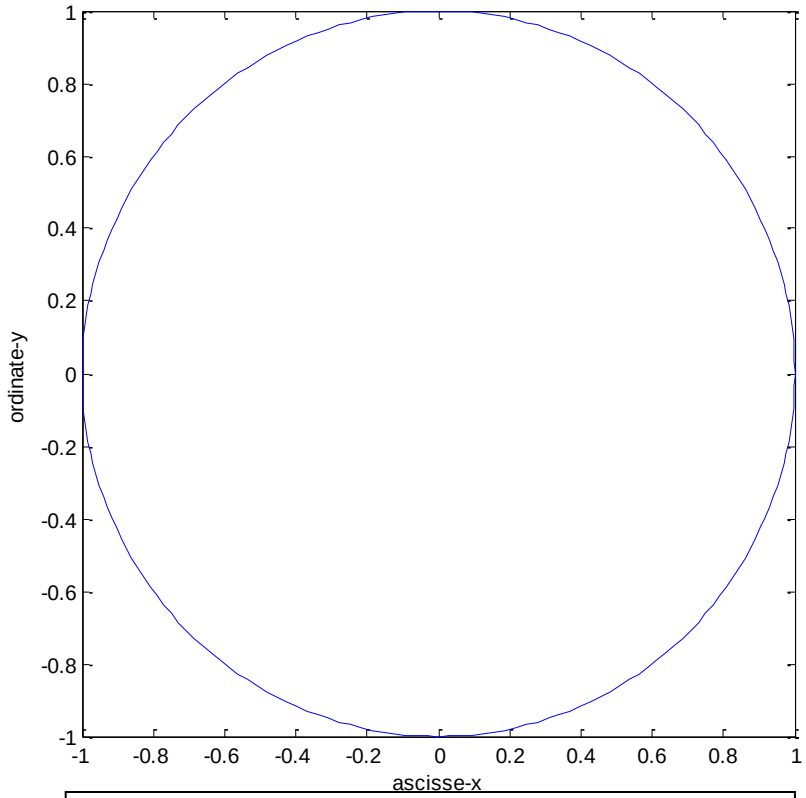
Esempi



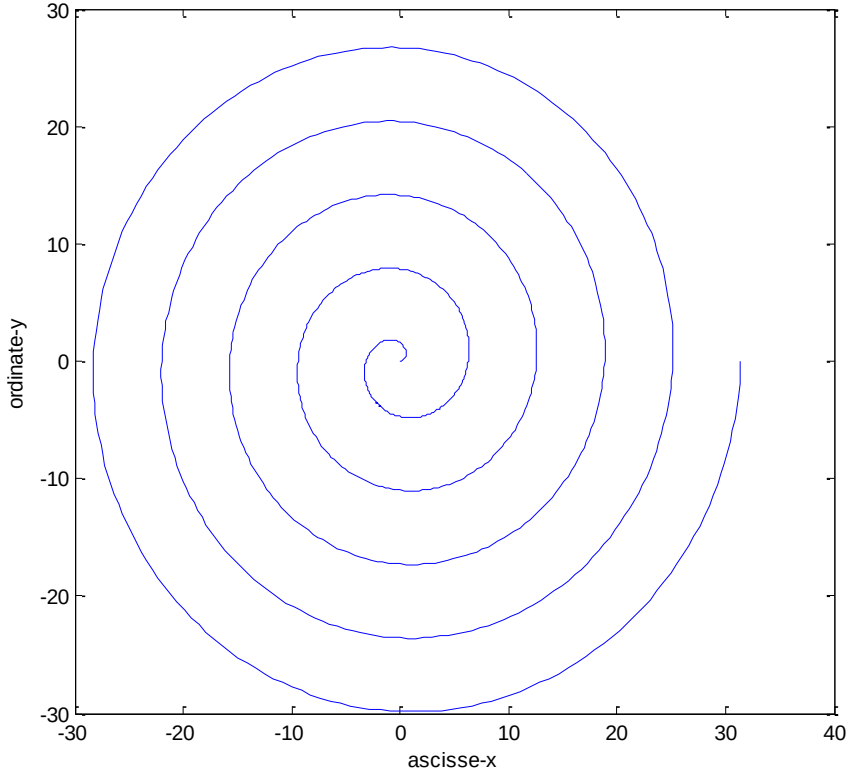

```
>> t=[0:pi/100:2*pi];
>> x=cos(t);
>> y=sin(t);
>> plot(x,y);
>> xlabel('ascisse-x');
>> ylabel('ordinate-y');
```

```
>> 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 due vettori di 100 punti x ed y contenenti i punti che stanno sulla curva

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

e le cui ascisse stanno all'interno dell'intervallo [a,b]

Soluzione

```
function [x, y] = samplePolynomial(polyCoeff, interval)

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

    x = [a  : (b-a) / 100 : b];
    y  = zeros(size(x));

    for ii = 1 :  1 : length(polyCoeff)
        y(ii) = polyCoeff(ii) * x(ii)^(length(polyCoeff) - ii);
    end

end
```

Esempi

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

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

e stampare le tre curve

Esempi

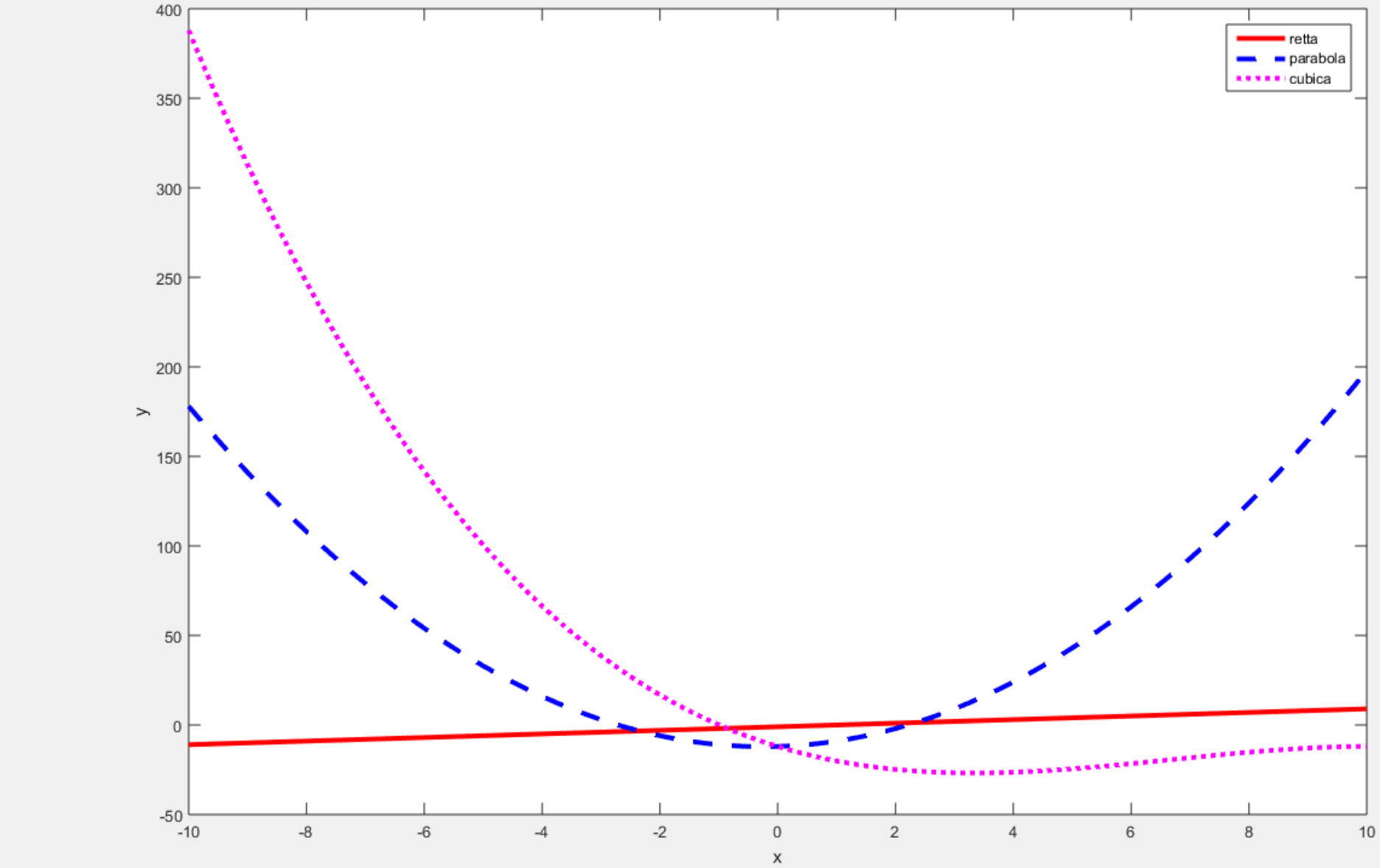
```
interval = [-10 , 10];
rettaCoeffs = [1 ,-1];
parabolaCoeffs = [ 2 ,  1 , -12] ;
cubicaCoeffs = [-0.1 , 2 ,-10 , -12];

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

Esempi

```
figure(1), plot(rx, ry, 'r-', 'LineWidth', 3)
hold on
plot(px, py, 'b--', 'LineWidth', 3)
plot(cx, cy, 'm:', 'LineWidth', 3)
hold off
legend('retta', 'parabola', 'cubica')
xlabel('x')
ylabel('y')
```

Figure 1

File Edit View Insert Tools Desktop Window Help



Funzioni Grafiche

Funzione	Scopo
figure(figNumber)	apre una figura identificata dall'handle figNumber . Se non presente definisce l'handle in maniera incrementale
hold	definisce se tenere (hold on) o cancellare (hold off) 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 tipo di linea ed il colore usato nella visualizzazione dei punti
imagesc(A)	Visualizza un'immagine A come un'immagine in colormap (tutti i colori) di default.
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 3D (VE LI GUARDATE DA SOLI!!!)

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`

Diagrammi lineari a tre dimensioni

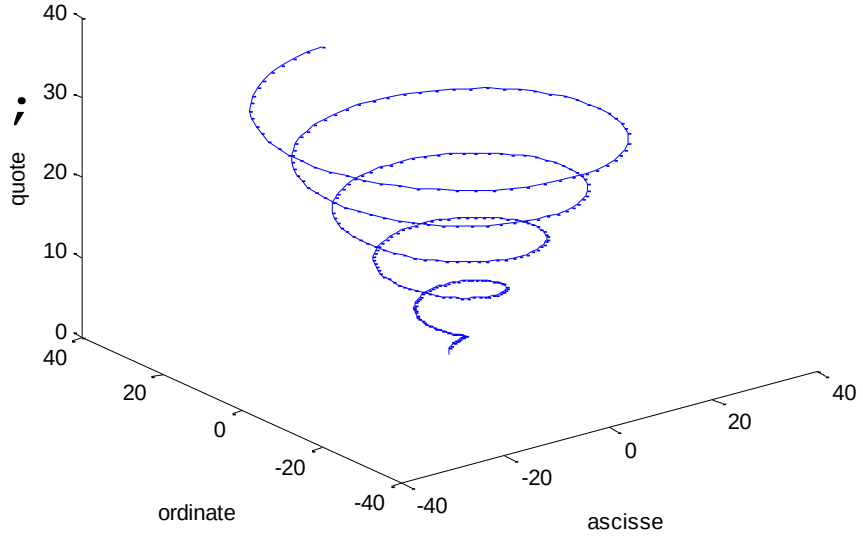
Generalizzazione del diagramma a due dimensione: insieme di terne di coordinate

`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');
```



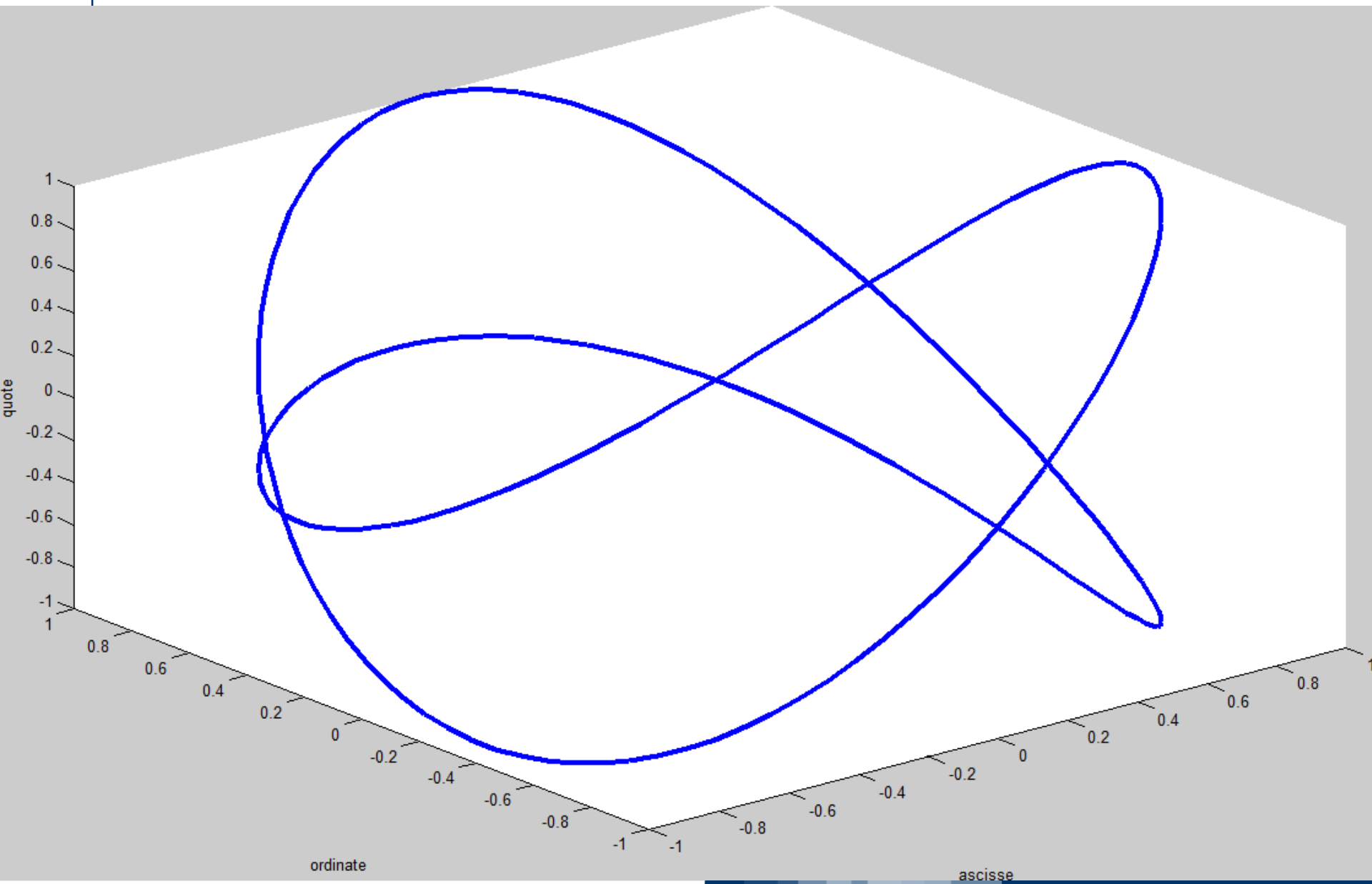
La funzione linspace

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

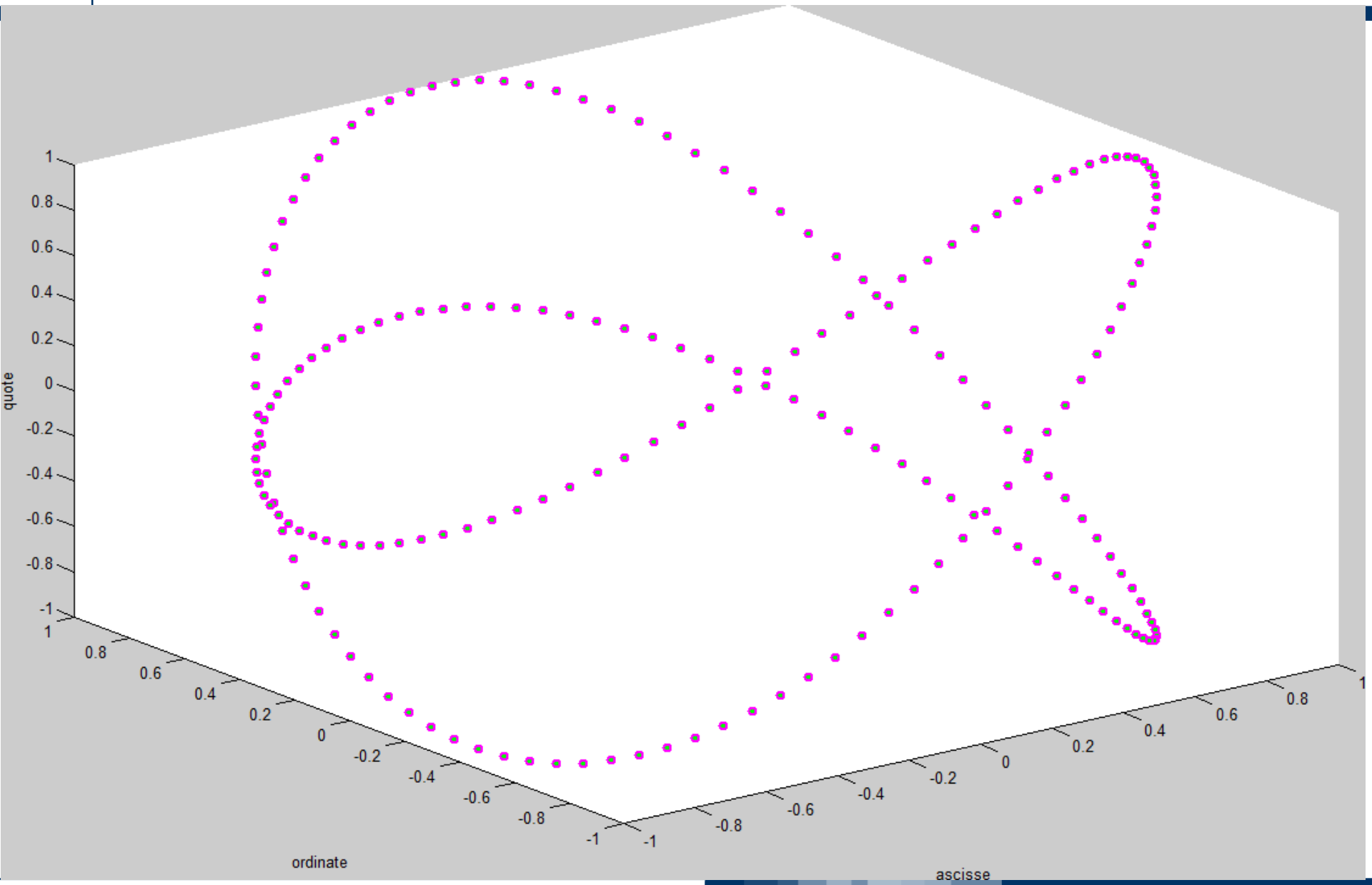
plot restituisce un hanlde: una variabile di riferimento per poter accedere nuovamente all'insieme di punti disegnato

set(plot_handle, 'Property Name', PropertyVal)
permette di modificare le proprietà dell'immagine

```
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 $z = f(x,y)$?

Superfici

Come si disegna una superficie che rappresenta una funzione a due variabili $z = f(x,y)$?

La funzione `mesh(xx,yy,zz)` genera una superficie, a partire da tre argomenti

- `xx` contiene le ascisse
- `yy` contiene le ordinate
- `zz` contiene le 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 una superficie, a partire da tre argomenti

- `xx` contiene le ascisse
- `yy` contiene le ordinate
- `zz` contiene le quote

`xx` e `yy` identificano due griglie

Per ogni coppia di valori `xx` e `yy` è specificato un valore di `zz`

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

Funzione meshgrid

```
[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

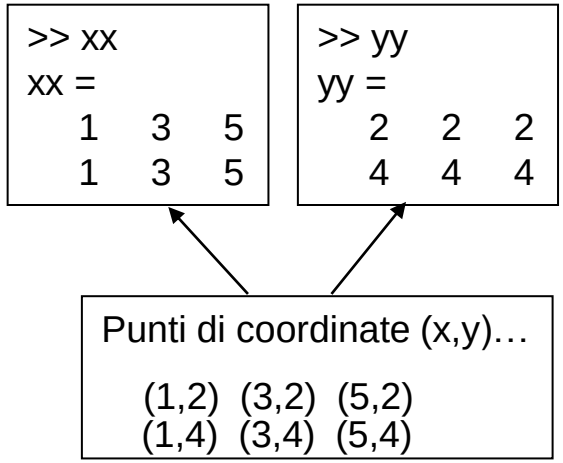
È possibile quindi valutare una funzione di queste due matrici, e.g., zz = xx + yy , e disegnarla mediante mesh

Superfici: esempi

```
% Disegnamo z=x+y
x=[1, 3, 5];
y=[2, 4];
```

Superfici: esempi

```
% Disegnamo z=x+y
x=[1, 3, 5];
y=[2, 4];
[xx, yy] = meshgrid(x, y);
```



Superfici: esempi

```
% Disegnamo z=x+y
x=[1, 3, 5];
y=[2, 4];
[xx, yy] = meshgrid(x, y);
zz = xx + yy;
```

```
>> 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
```

Superfici: esempi

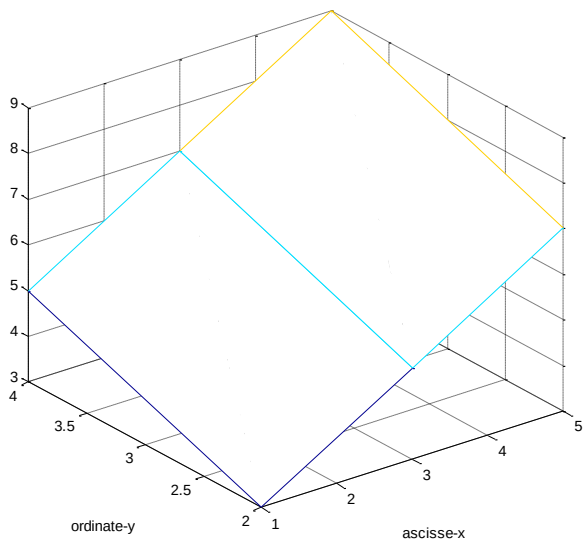
```
% Disegnamo z=x+y
x=[1, 3, 5];
y=[2, 4];
[xx, yy] = meshgrid(x, y);
zz = xx + yy;
mesh(xx, yy, zz);
xlabel('ascisse-x'); ylabel('ordinate-y');
```

```
>> 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)

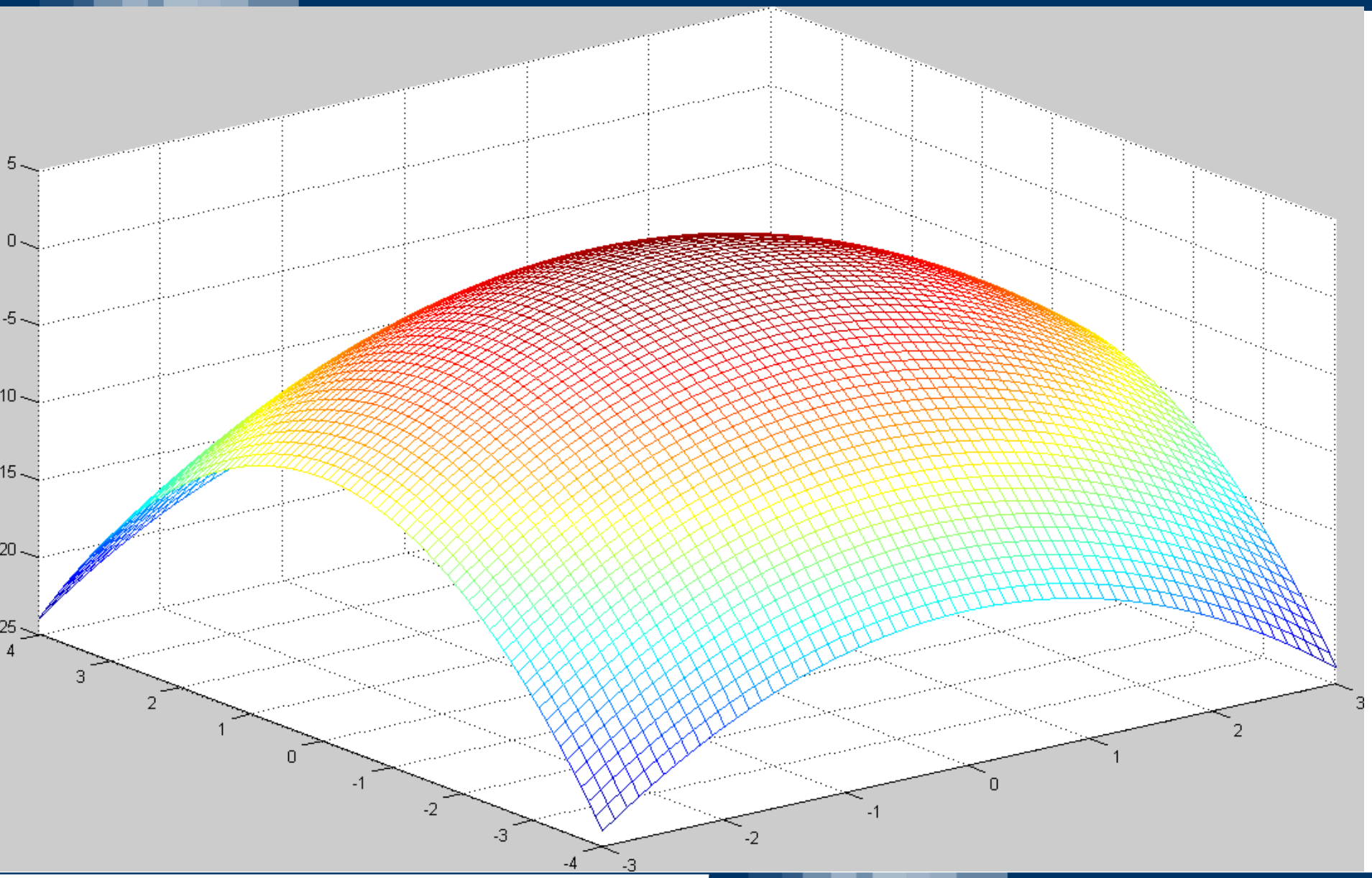
(1,2,3) (3,2,5) (5,2,7)
(1,4,5) (3,4,7) (5,4,9)

(NB: z=x+y)

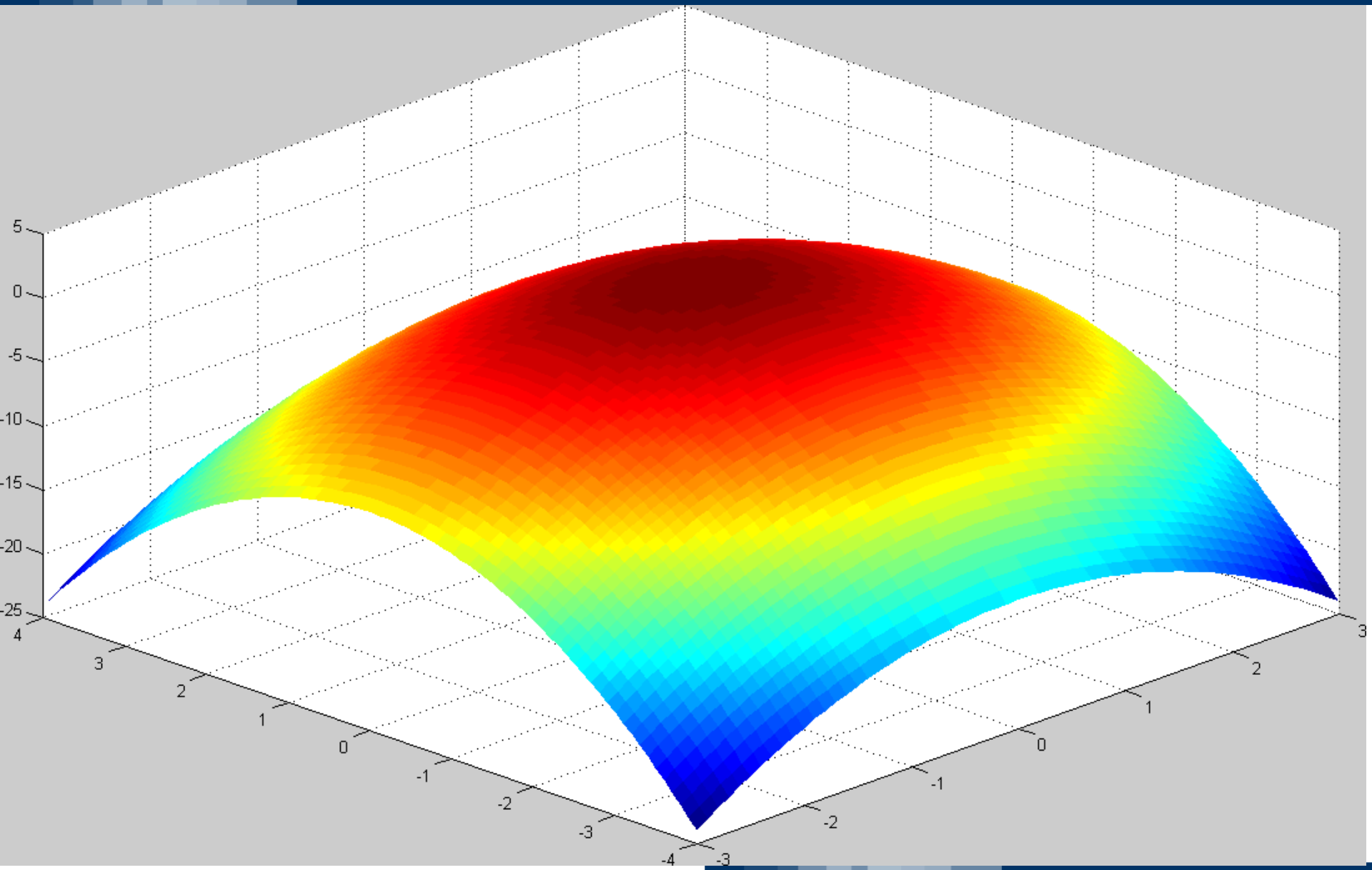
Mesh

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

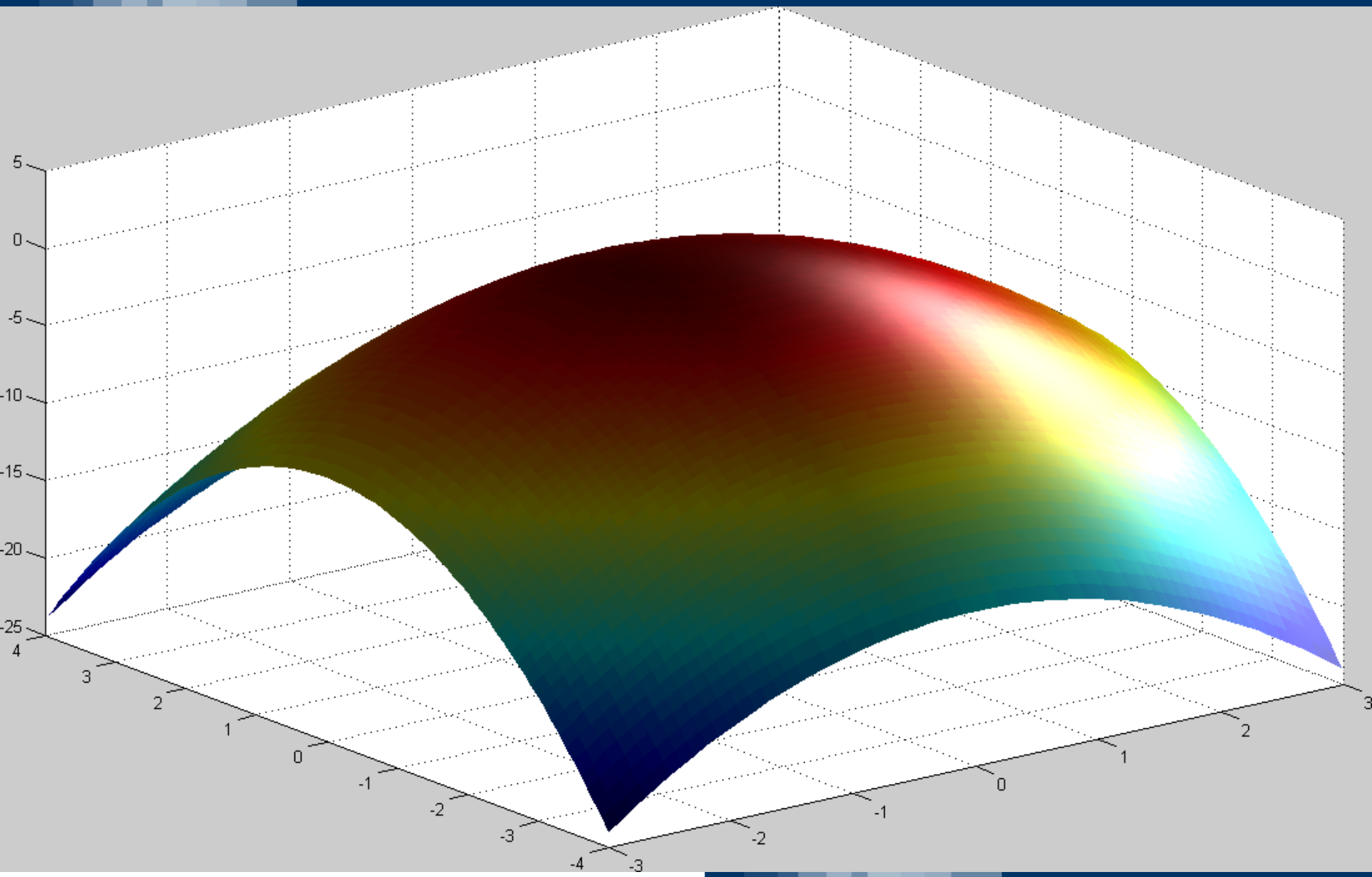
```
[xx, yy] = meshgrid([-3 : 0.1 :3], [-4 : 0.1 :4]);
f = @(x, y)(1 - x.^2 - y.^2);

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


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);

aa = surf(xx, yy, f(xx, yy))
set(aa, 'EdgeColor', 'none')
% EdgeColor = none "toglie" la griglia dal grafico
```


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
% light aggiunge una sorgente luminosa per il rendering
```


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

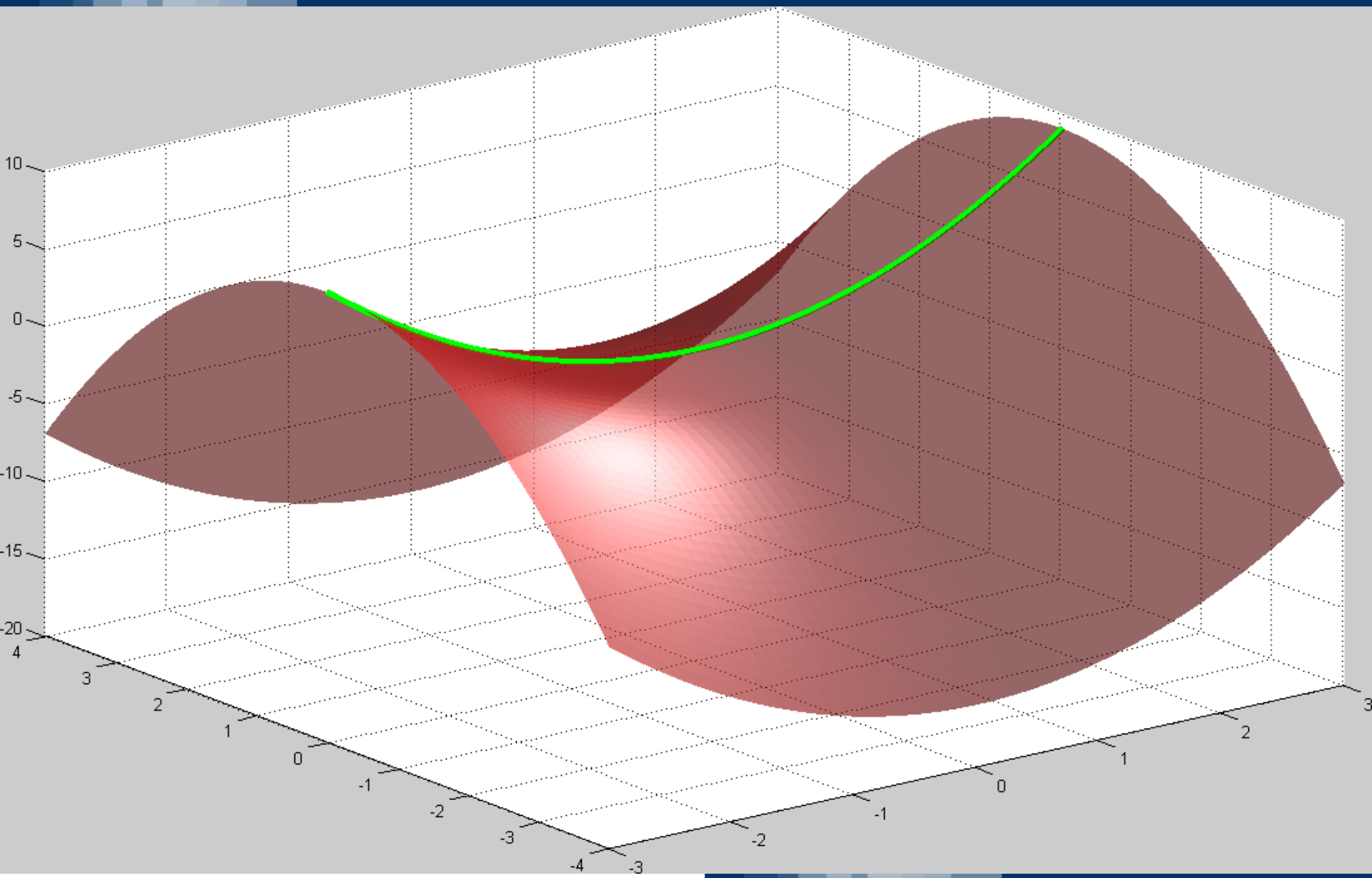
- plot3(), mesh(),altre funzioni grafiche quali surf() etc..

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

Esempio, disegnare in sovraimpressione alla quadrica

$z = x^2 - y^2$ 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

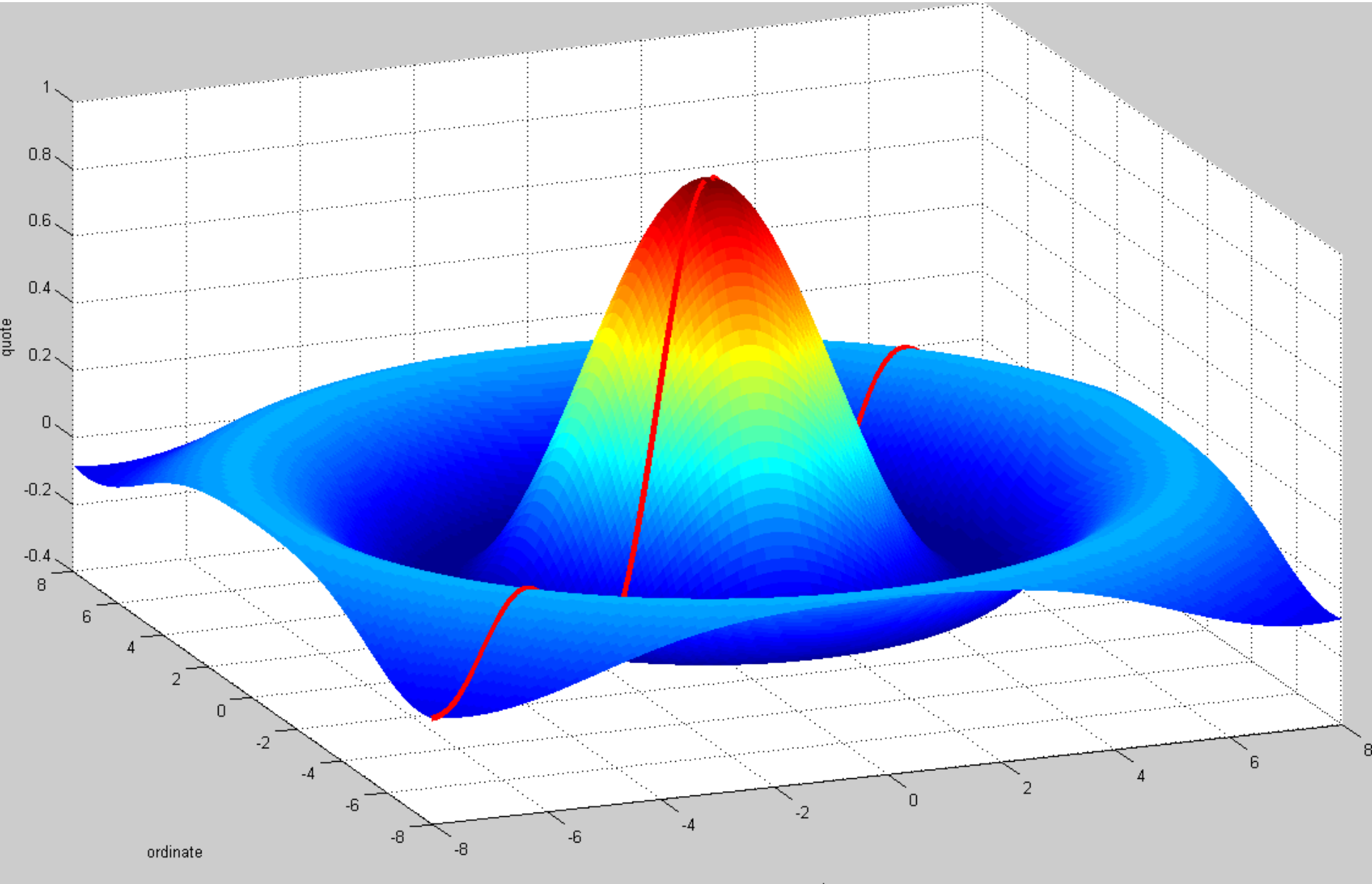
Disegnare la funzione

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

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)



Strutture in Matlab

Structure array (array di strutture)

- Una struttura è un tipo di dato composto da elementi individuali possibilmente non omogenei

Structure array (array di strutture)

- Una struttura è un tipo di dato composto da elementi individuali possibilmente non omogenei
- Ogni elemento individuale è chiamato *campo* ed ha un nome

Structure array (array di strutture)

- Una struttura è un tipo di dato composto da elementi individuali possibilmente non omogenei
- Ogni elemento individuale è chiamato *campo* ed ha un nome
- Una struttura può avere campi di tipo diverso

Structure array (array di strutture)

- Una struttura è un tipo di dato composto da elementi individuali possibilmente non omogenei
- Ogni elemento individuale è chiamato *campo* ed ha un nome
- Una struttura può avere campi di tipo diverso
- E’ possibile (naturale) creare array di strutture

Structure array (array di strutture)

- Una struttura è un tipo di dato composto da elementi individuali possibilmente non omogenei
- Ogni elemento individuale è chiamato *campo* ed ha un nome
- Una struttura può avere campi di tipo diverso
- E' possibile (naturale) creare array di strutture
- **Creazione di una struttura** (e di array di strutture):
  - Utilizzando la funzione struct()
  - Assegnamento dei valori ai campi (e contestuale definizione dei campi)

Creazione di una struct

Creazione di una struttura :

Utilizzando la funzione struct()

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

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;
```

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;
```

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;
```

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**).

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

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

Aggiunta di campi

- Aggiunta di un campo

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

- Il campo esami viene aggiunto a tutti gli elementi del vettore studente
  - Avrà un valore iniziale solo per studente(2)
  - Sarà vuoto per tutti gli altri elementi di studente

Rimozione di campi

- Rimozione di un campo da un oggetto struct
  - studente(1).nome = [];
  - studente(1) avrà il campo nome vuoto

Rimozione di campi

- Rimozione di un campo da un oggetto struct
  - studente(1).nome = [];
  - studente(1) avrà il campo nome vuoto

- Rimozione di un elemento da un array di strutture
  - Come per gli elementi dell'array è possibile usare l'assegnamento al vettore vuoto
  - Es per rimuovere il secondo studente

    studente(2) = []

Esercizio

Si sviluppi uno script matlab che acquisisce da tastiera i valori di latitudine, longitudine e altezza di 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

```
n = input(['quanti rilievi? ']);
```

Soluzione

```
n = input(['quanti rilievi? ']);
 % acquisizione dei rilievi
for ii = 1 : n



end
```

Soluzione

```
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
```

Soluzione

```
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];
```

Soluzione

```
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;
```

Soluzione

```
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
disp(mean(ALT(posOK)));
```

Soluzione con find

```
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
```

Soluzione con find

```
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

%% seleziona i rilievi richiesti
idx=find([s.latitudine] >= 10 & [s.latitudine]<=100 & [s.longitudine]>=30
& [s.longitudine]<=60);
```

Soluzione con find

```
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

%% seleziona i rilievi richiesti
idx=find([s.latitudine] >= 10 & [s.latitudine]<=100 & [s.longitudine]>=30
& [s.longitudine]<=60);

aa = [s.altezza];
altezzaMedia = mean(aa(idx));
```