Funzioni Built-in, di ordine superiore e Struct in Matlab

Informatica B AA 17/18

Giacomo Boracchi

giacomo.boracchi@polimi.it

29 Novembre 2017

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 xx ed yy contenenti i punti che stanno sulla curva (e le cui ascisse stanno in [a,b])

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

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
```

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 più lunga del vettore
size(arr)	Ritorna un vettore [r c] con il numero r di righe e c di colonne della matrice; se **`arr`** ha più dimensioni ritorna array con numero elementi per ogni dimensione

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]; $\longrightarrow \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$
- d = zeros(size(c));

Funzioni Aritmetiche

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

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

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

Altre Funzioni Logiche: find

`indx = find(x)` restituisce gli indici degli elementi **`true`** dell'array logical **`x`**.

```
a = [5 6 7 2 10];
b = a > 5; % b =[0 1 1 0 1]
k = find(b)
k = 2 3 5
```

Altre Funzioni Logiche: find

Spesso **x** è sostituito da un'espressione logica.

Esempio

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

Attenzione,

la sintassi NON è **find(expr)** ma **find(logical)**

Altre Funzioni Logiche: find

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];
vals = y((x>0) & (y>0)) -> vals = [2 7]
indx = find((x>0) & (y>0))-> indx = [1 5]
```

L'estrazione di un sottovettore è più rapida solo con le operazioni logiche. Matlab suggerisce di evitare find

```
k = x(find(x>0))
```

Matlab suggerisce di usare perchè più rapido.

```
k = x(x>0)
```

Esempio

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

Esempi

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

```
>> 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
```

Esercizio: implementare usando find()

Esempi

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

Esempi

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

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

Esercizio

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

Esercizio

Scrivere 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); % returns unique values in a
vector
```

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

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

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

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!

Esercizio TODO

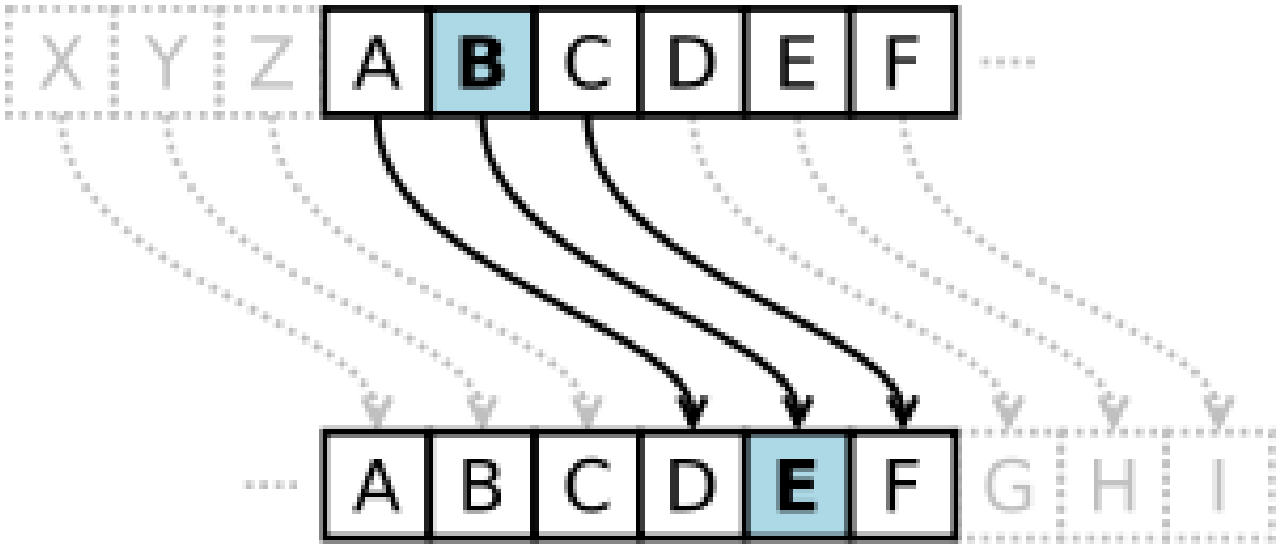
Scrivere un programma che richiede in ingresso due parole e determina se l'una è l'anagramma dell'altra

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

TODO: 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 e Return

Funzioni per Stringhe

Esiste la funzione di **confronto**

TF = strcmp(str1 , str2)

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

NB: in linea di principio è possibile confrontare le stringhe come due vettori, con l'operatore == . Questo però richiede che le **due stringhe abbiano le stesse dimensioni.** Altrimenti genera errori

- La funzione strcmp permette di confrontare anche stringhe di dimensione diverse (restituendo false).

```
if('cane'== 'canguro')
 disp('uguali')
else
    disp('diverse')
End
›› Error using  ==
Matrix dimensions must agree.

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

Funzioni per Stringhe

Non occorre strlen (si usa length o size)

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

Esiste la funzione di **ricerca**

K = strfind(TEXT,PATTERN)

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

Return

Non necessario in Matlab,

- I valori ritornati sono definiti dall'header della funzione
- Può essere usato per terminare l'esecuzione di una 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
```

TODO

Implementare la propria versione delle funzioni

- **`strcmpr`**
- **`any`**
- **`all`**

Scrivere una funzione **`replacePattern`** per sostituire, all'interno di una stringa **`txt`**, tutte le occorrenze di una determinata stringa **`s1`** con **`s2`**. Si assuma inizialmente che **`s1`** e **`s2`** abbiano la stessa lunghezza. Poi si rilassi questa ipotesi.

N.B **`replacePattern`** è implementata in Matlab dalla funzione **`strrep`**. La richiesta è quindi equivalente ad implementare la vostra versione di **`strrep`**.

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

a = input(txtToShow) visualizza txtToShow nella command window e apre un prompt per 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 apre un prompt per inserire 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 ;
- Altri due modi
  - disp
    - accetta come parametro un array. Se questo array è di tipo char, lo stampa
    - viene usato in congiunzione con num2str
    - Esempio:
      - str = [‘il valore di pi e` ‘ num2str(pi)];
      - disp(str);
    - Stampa: “il valore di pi e` 3.1416”

Stampa dei risultati (2)

- …altro modo
  - fprintf
    - fprintf('Il valore di pi e` %f \n', pi);
    - stringhe di formato: %d (interi), %e (formato esponenziale), %f (virgola mobile), %g (il più corto tra il formato esponenziale e quello in virgola mobile)
- disp vs fprintf
  - disp è in grado di stampare anche valori complessi
    - x=2*(1-2*i)^3;
    - str=['disp: x = ' num2str(x)];
    - disp(str); disp: x = -22+4i
  - fprintf ne stampa solo la parte reale
    - fprintf('fprintf: x = %8.4f\n', x); fprintf: x = -22.0000

Esempio

```
u = input(' inserire vettore: ');
v = input(' inserire vettore: ');
% in questo modo u e v saranno certamente vettori riga
u = u(:)';
v = v(:)';
if (u> v)
    disp([num2str(u) , ' è sempre maggiore di ' , num2str(v)]);
elseif(u<v)
        disp([num2str(u) , ' è sempre minore di ' , num2str(v)]);
elseif(u==v)
    disp([num2str(u) , ' e ' , num2str(v) , ' coincidono']);
else
    disp([num2str(u) , ' ha valori sia maggiori che minori di ' ,
num2str(v) ]);
end
```

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

Funzioni Variabile

Function Handles

Variabili funzioni: function handles

In Matlab esistono **variabili di "tipo funzione”**

Un valore di tipo funzione può essere assegnato a una variabile detta handle

- l’hanlde è una funzione: quindi all’handle possono essere passati alcuni parametri in ingresso e l’handle restituirà dei parametri in uscita
- l’hanlde è una variabile: quindi può essere utilizzato come parametro attuale di una funzione.
    - Quindi l’handle permette di passare una funzione (l’handle) come argomento di un’altra funzione (che in tal caso è detta «di ordine superiore»)

Assegnamento di un valore di tipo funzione

Sintassi per definire l'handle ad una funzione esistente

```
f = @nome_funzione
```

- Esempio

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

Funzioni Anonime

È possibile definire una funzione e assegnarla direttamente all'handle

- Non esiste un file che contiene la funzione
- La funzione viene detta anonima: non ha un nome proprio ma solo il nome dell'handle che la contiene

Sintassi

```
f = @(x,y...)‹expr›
```

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

Esempio

Handle che definisce una funzione anonima per elevare al quadrato un input

```
>> 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
Undefined operator '+' for input arguments of type
'function_handle'.
```

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 (o non sono state scritte per operare su vettori). Ad esempio:

- La funzione per vedere se un numero è primo (codici sviluppati nella prima parte del corso)
- La funzione per calcolare se un anno è biestile …

Come facciamo ad applicare la funzione **`f`**, definita solo per input scalari, a tutti gli elementi di un vettore?

- Scriviamo una seconda funzione o uno script che esplicitcamente invoca la funzione **`f`**
- Facciamo un function handle ad **`f`** e passiamo il functionn handle alla funzione map: map(@controllaSePrimo, vett)

La seconda opzione è decisamente preferibile

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

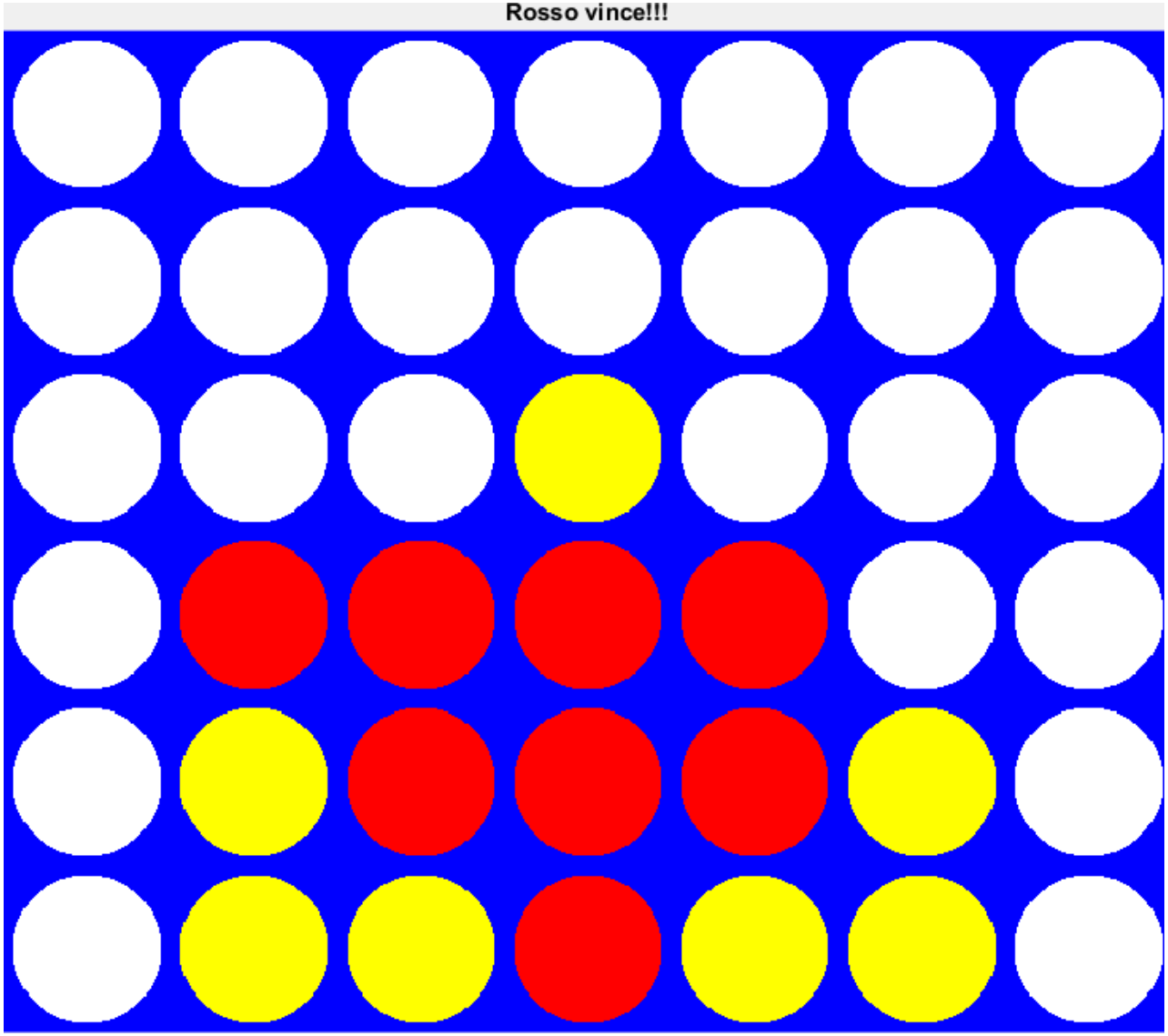
```
p = 1;
for ii = 1 : numel(x)
   p= p * x(ii);
end
```

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

A Great Challenge

Forza 4

Forza 4



Il vosto compito

Scrivere una funzione chiamata `myStrategy892312` dove sostituite a `892312` il vostro numero di matericola

```
function colonna = myStrategy892312(Griglia, colore)
```

Dove

`Griglia,` è una matrice 6x7 che rappresenta il tabellone (0 celle vuote, 1 celle rosse, -1 celle gialle)

`colore,` è 1 per il giocatore rosso, -1 per il giallo

`colonna,` un intero da 1 a 7 che identifica la colonna in cui inserire la pedina

Il Match: funzione di ordine superiore

```
% Partita fra due strategie
clear
close all

% inizializza la generazione dei numeri casuali
rng('shuffle');

%mostrare o non mostrare la griglia
show = 1;

%invoca la partita tra le due strategie
res = match(@crd_strategy, @human_strategy,
show);
```

Regole del tornero

Ogni studente o squadra (al massimo di due studenti) può sottomettere una sola funzione contenente una stragegia

Fase1:

- Tutte le strategie sottomesse giocheranno contro una nostra myStrategyBase implementata da Trovò e Carrera
- Si vince alla meglio delle 100 partite (50 in casa, 50 fuori casa)

Fase2:

- Le strategie che passano Fase1 giocheranno in un torneo all'italiana a cui parteciperà anche myStrategyCRD imlementata da Carrera

Deadlines e Premi

Deadlines e Submission Instruction:

- Consegna: 23:59 di Domenica 17 Dicembre (UTC +2), invio via mail al **Dr. Trovò** (cc Dr. Carrera).
  Oggetto mail CONTEST FORZA 4.
- Nella mail specificare nome e cognome dei partecipanti

Il framework con strategie base da provare sarà rilasciato a breve.

Presentazione risultati torneo:

- Mercoledì 20 Dicembre a lezione

Ricchi premi a chi passa Fase 1 e ancor più chi vince Fase 2

Further information

Venerdì ad esecitazione il Dr. Trovò vi presenta il framework e fornisce maggiori vari dettagli

Per implementare la vostra soluzione potrebbero essere utili le funzioni ricorsive che verranno mostrate nella prossima lezione

Funzioni Built in per Visualizzazione

Funzioni Grafiche

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

Diagrammi a due dimensioni

La funzione `plot(x,y)` disegna il **diagramma** cartesiano dei punti che hanno valori delle ascisse nel vettore x, delle ordinate nel vettore y

Il diagramma è l'insieme di **coppie di punti** [x(1), y(1)],…, [x(end),y(end)] rappresentanti le coordinate dei punti del piano cartesiano

- La funzione `plot` congiunge i punti con una linea, per dare continuità al grafico.

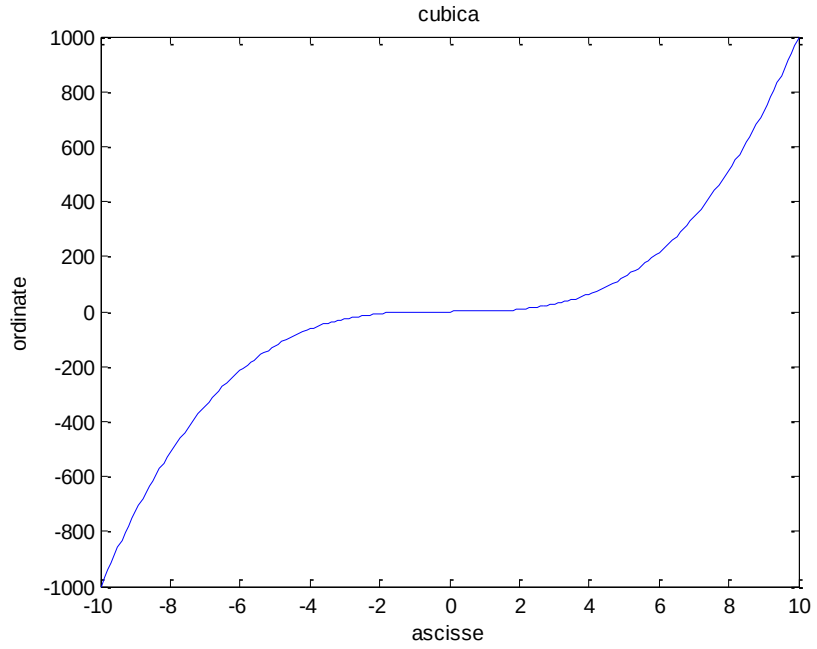
In `plot(x,y)`, `x` e `y` devono essere **due vettori aventi le stesse dimensioni**

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

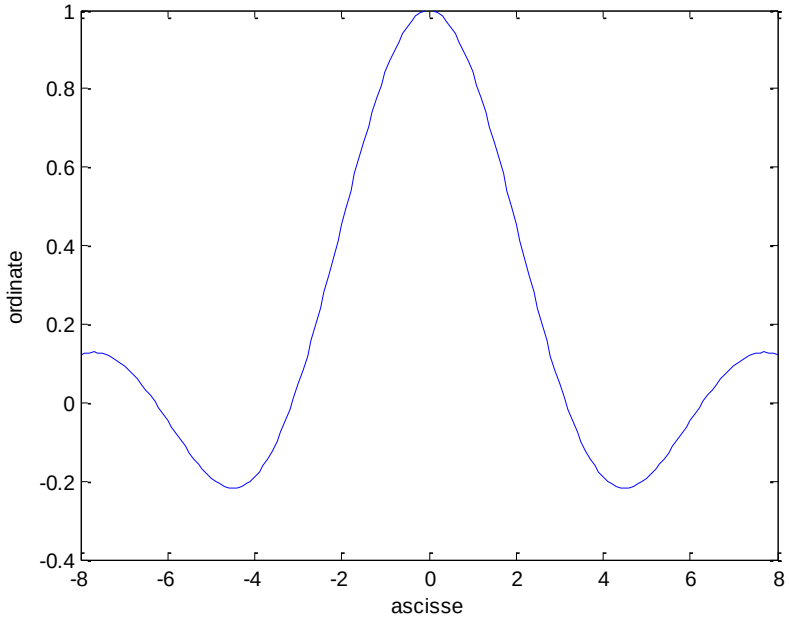
Le funzioni `xlabel` visualizzano una stringa come nome asse ascisse, `ylabel` per ordinate, `title` per il titolo

Diagrammi a due dimensioni: esempi

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




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



Diagrammi a due dimensioni: ancora esempi

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

In `plot(x,y)` non necessariamente `x` contiene valori equispaziati e `y` non è necessariamente funzione di `x`. Sia `x` che `y` possono essere, ad esempio, funzioni di qualche altro parametro.

Che diagrammi disegnano i seguenti esempi?

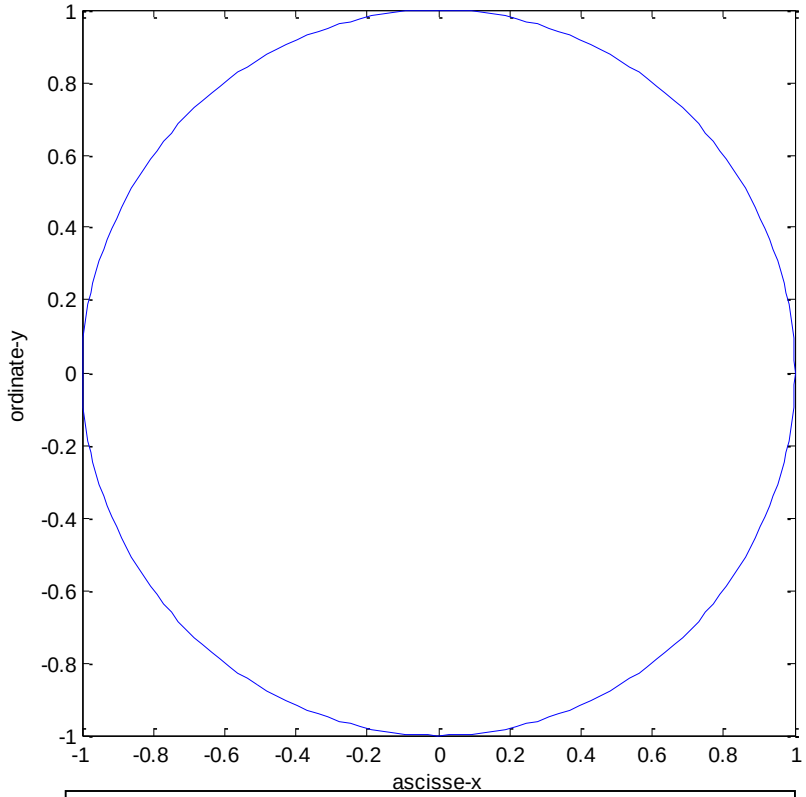
```
>> t=[0:pi/100:2*pi];
>> x=cos(t);
>> y=sin(t);
>> plot(x,y);
>> xlabel('ascisse-x');
>> ylabel('ordinate-y');
```

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

Esempi

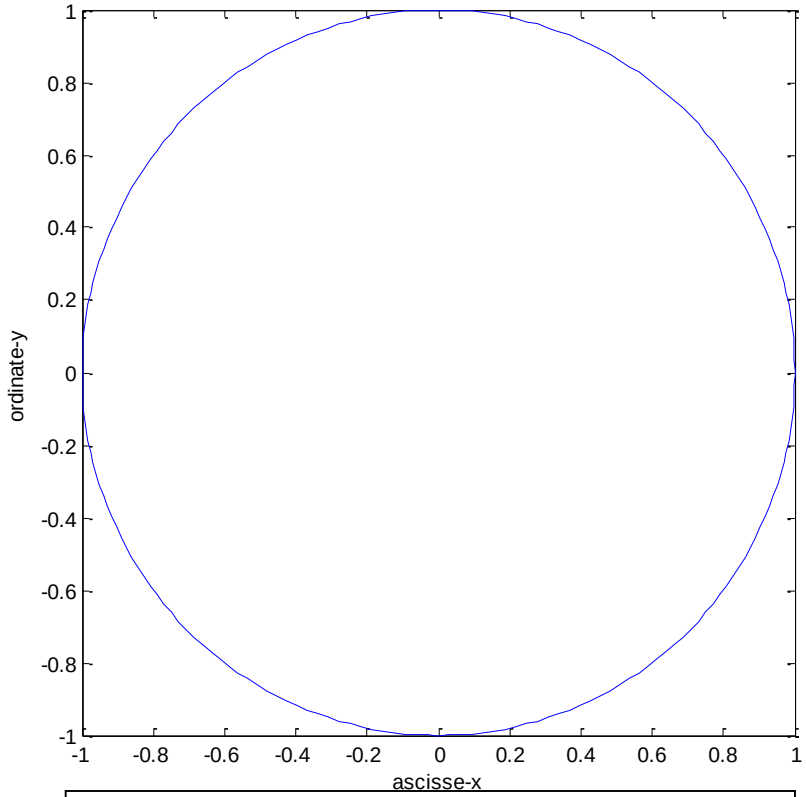
```
>> t=[0:pi/100:2*pi];
>> x=cos(t);
>> y=sin(t);
>> plot(x,y);
>> xlabel('ascisse-x');
>> ylabel('ordinate-y');
```

Esempi




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

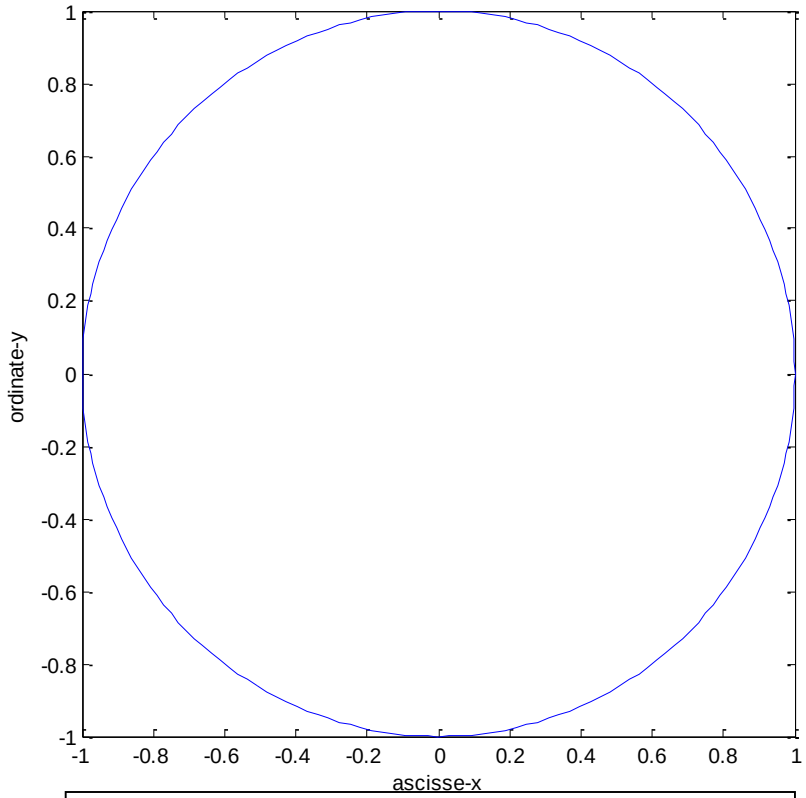
Esempi



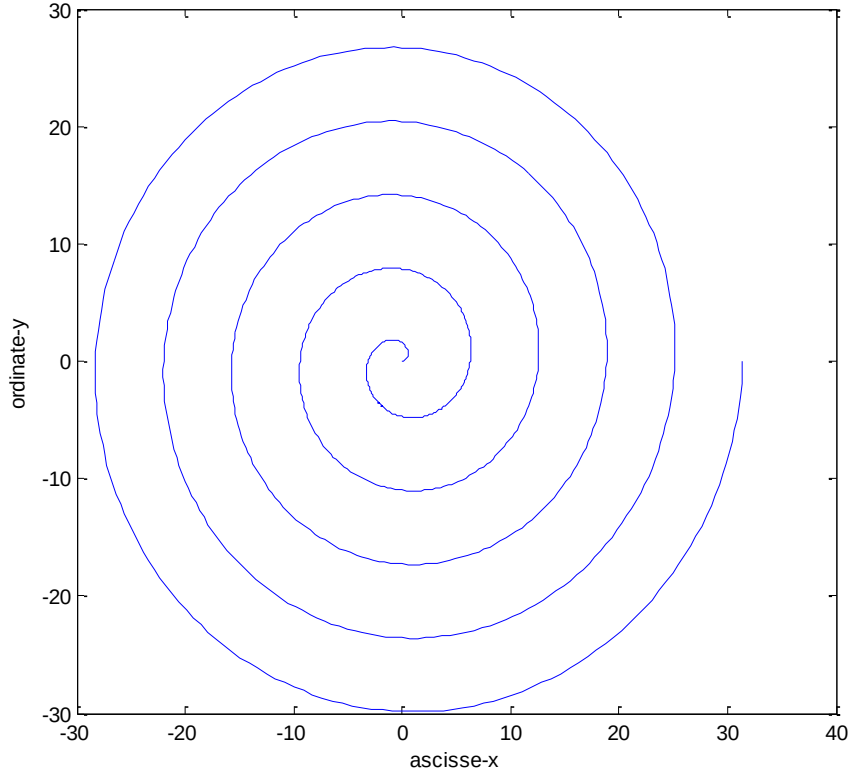

```
>> t=[0:pi/100:2*pi];
>> x=cos(t);
>> y=sin(t);
>> plot(x,y);
>> xlabel('ascisse-x');
>> ylabel('ordinate-y');
```

```
>> 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 xx ed yy contenenti i punti che stanno sulla curva (e le cui ascisse stanno in [a,b])

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

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 avente y 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);
```

Disegno le tre curve

```
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

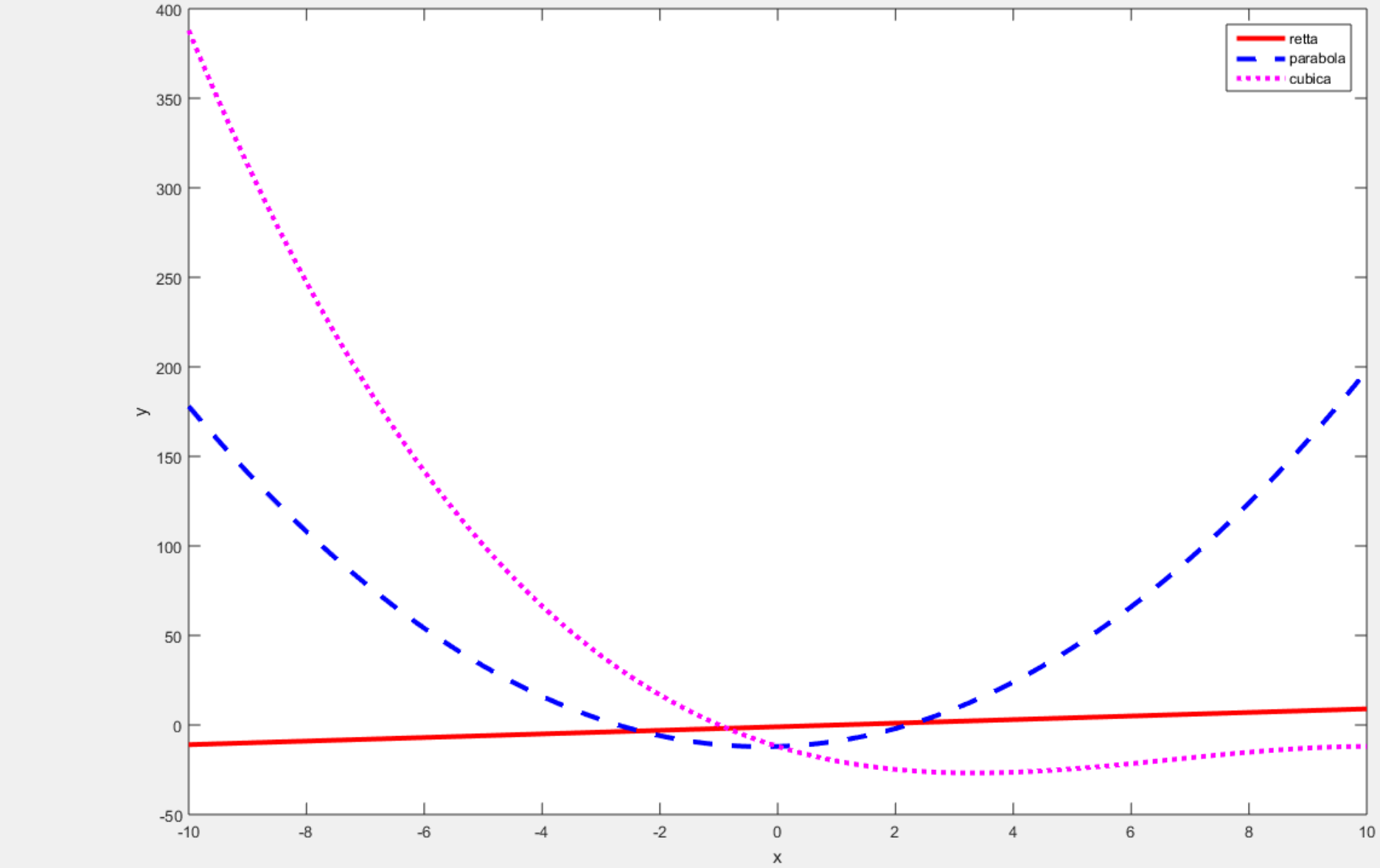
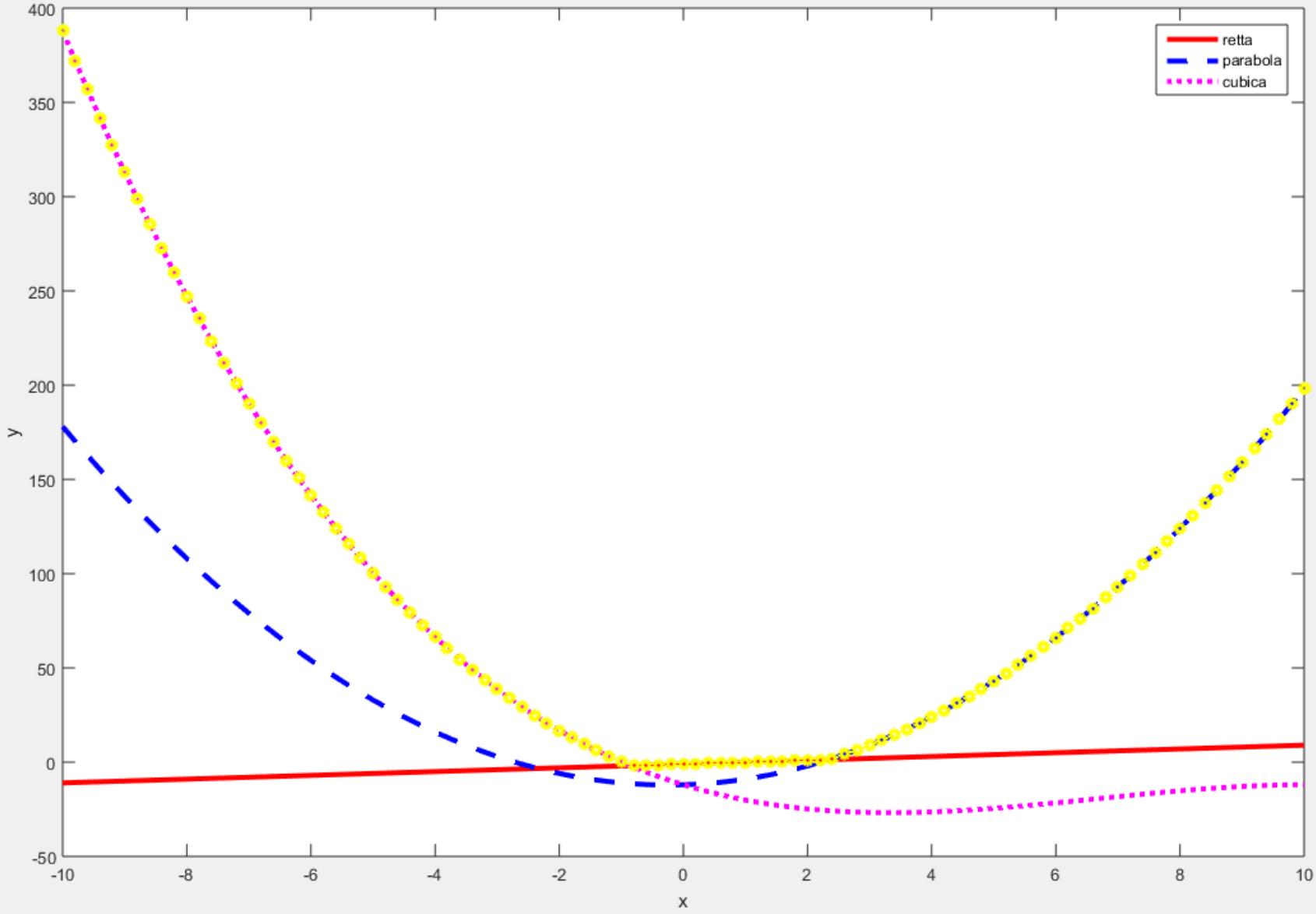


Figure 1

File Edit View Insert Tools Desktop Window Help



Plot in tre dimensioni

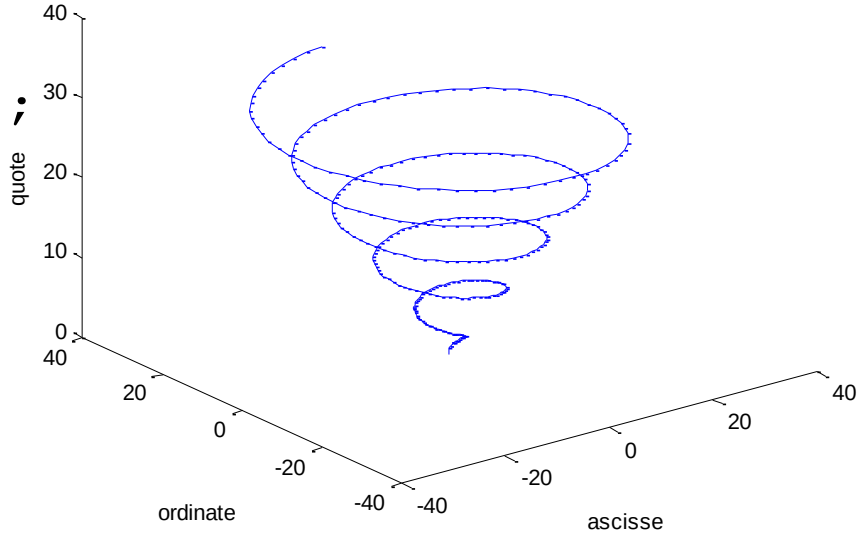
Generalizzazione del diagramma a due dimensione: insieme di terne di coordinate

`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');
```



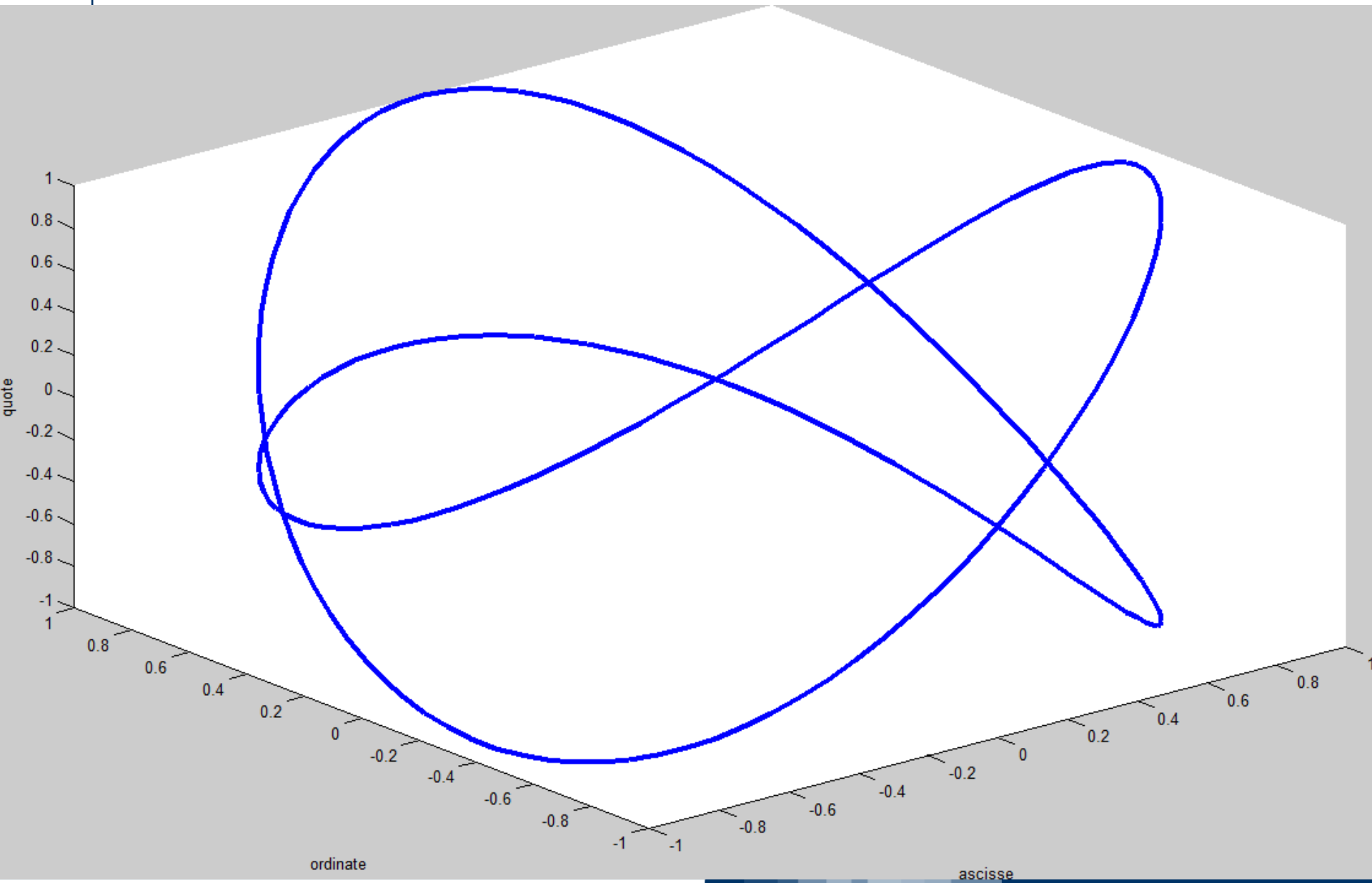
La funzione linspace

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

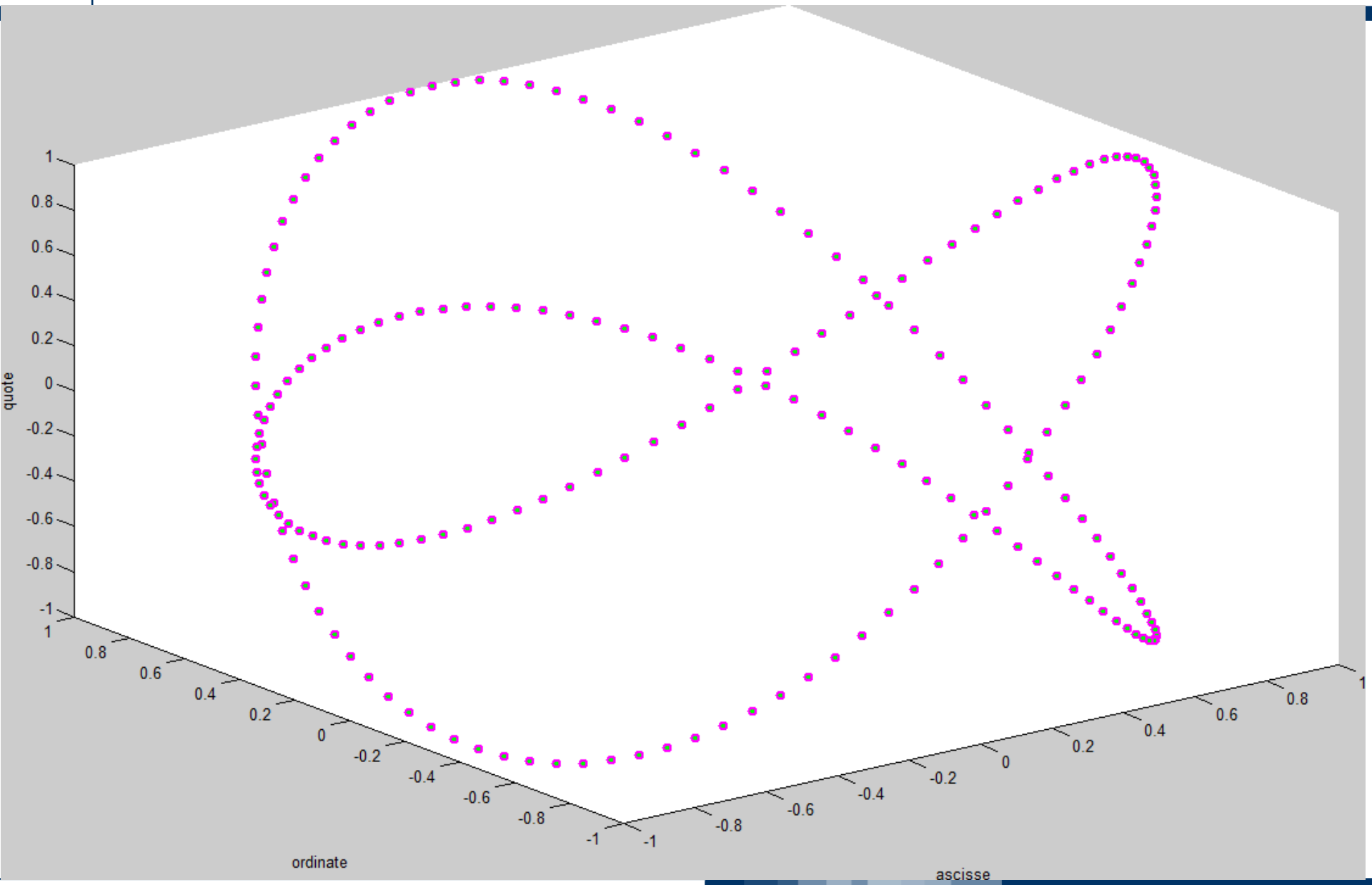
plot restituisce un hanlde, una variabile di riferimento per poter accedere nuovamente all'insieme di punti disegnato

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

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

Occorre definire il dominio che non è più un intervallo in una retta ma una porzione del piano

Superfici

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

Occorre definire il dominio che non è più un intervallo in una retta ma una porzione del piano

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` sono matrici che identificano una griglia in corrispondenza del quale per `zz` rappresenta il valore della funzione

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

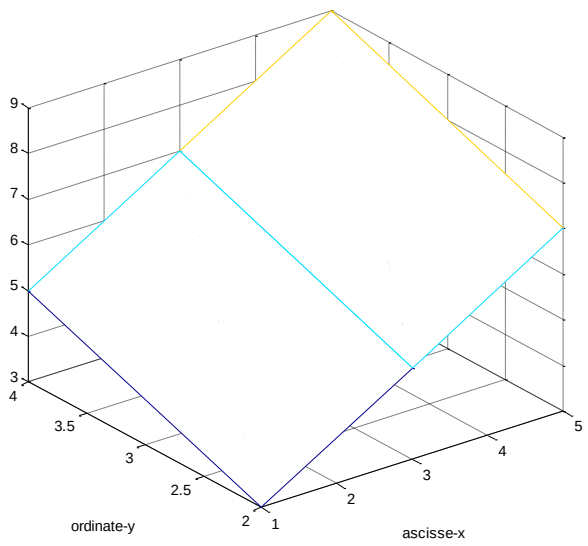
```
% Disegnamo z=x+y
x=[1, 3, 5];
y=[2, 4];
[xx, yy] = meshgrid(x, y);
zz = xx + yy;
mesh(xx, yy, zz);
xlabel('ascisse-x'); ylabel('ordinate-y');
```

```
>> 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)

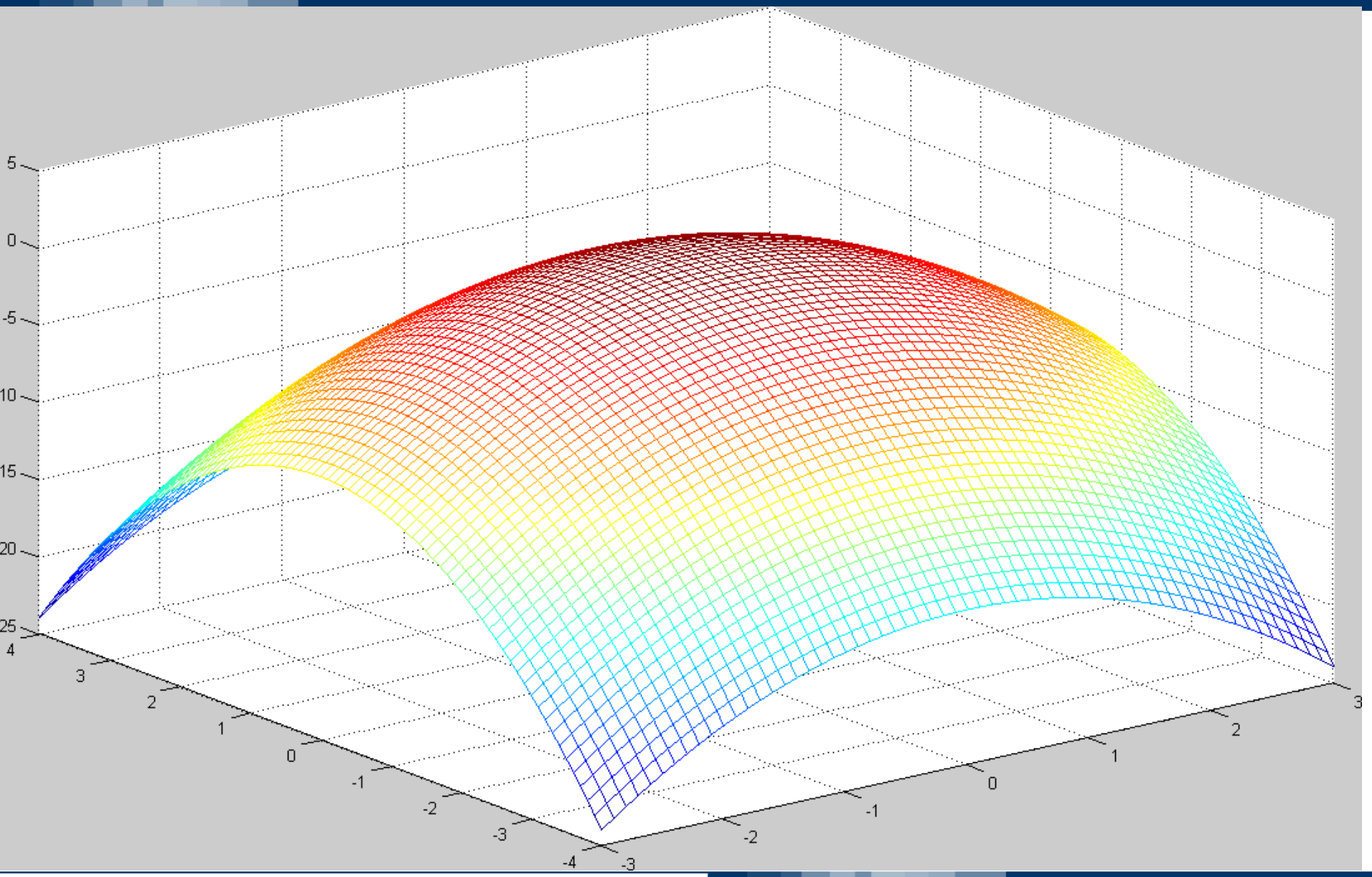
(1,2,3) (3,2,5) (5,2,7)
(1,4,5) (3,4,7) (5,4,9)

(NB: z=x+y)

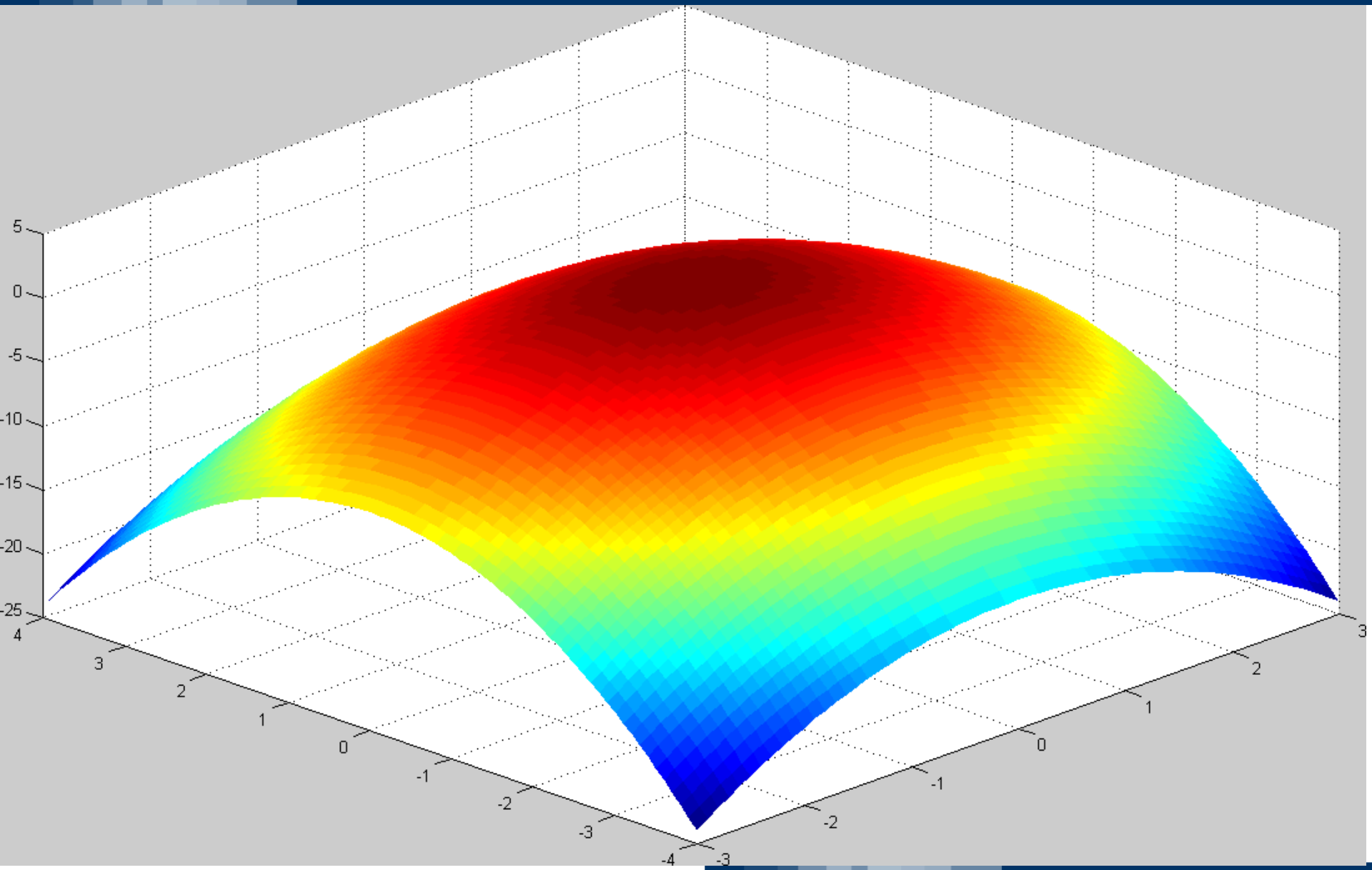
Mesh

```
[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

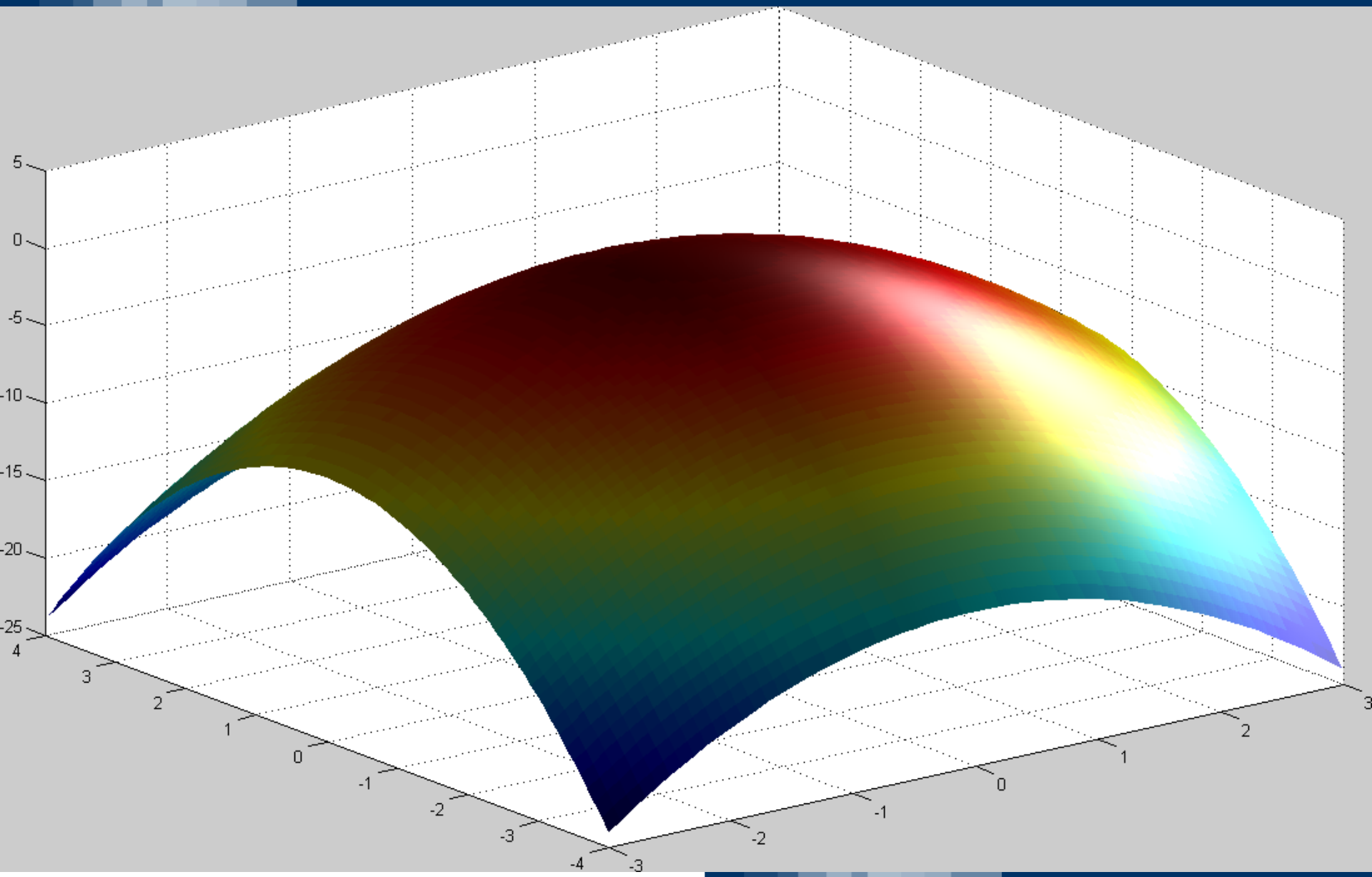

5
0
-5
-10
-15
-20
-25
4
3
2
1
0
-1
-2
-3
-4
-3
-2
-1
0
1
2
3

```
[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

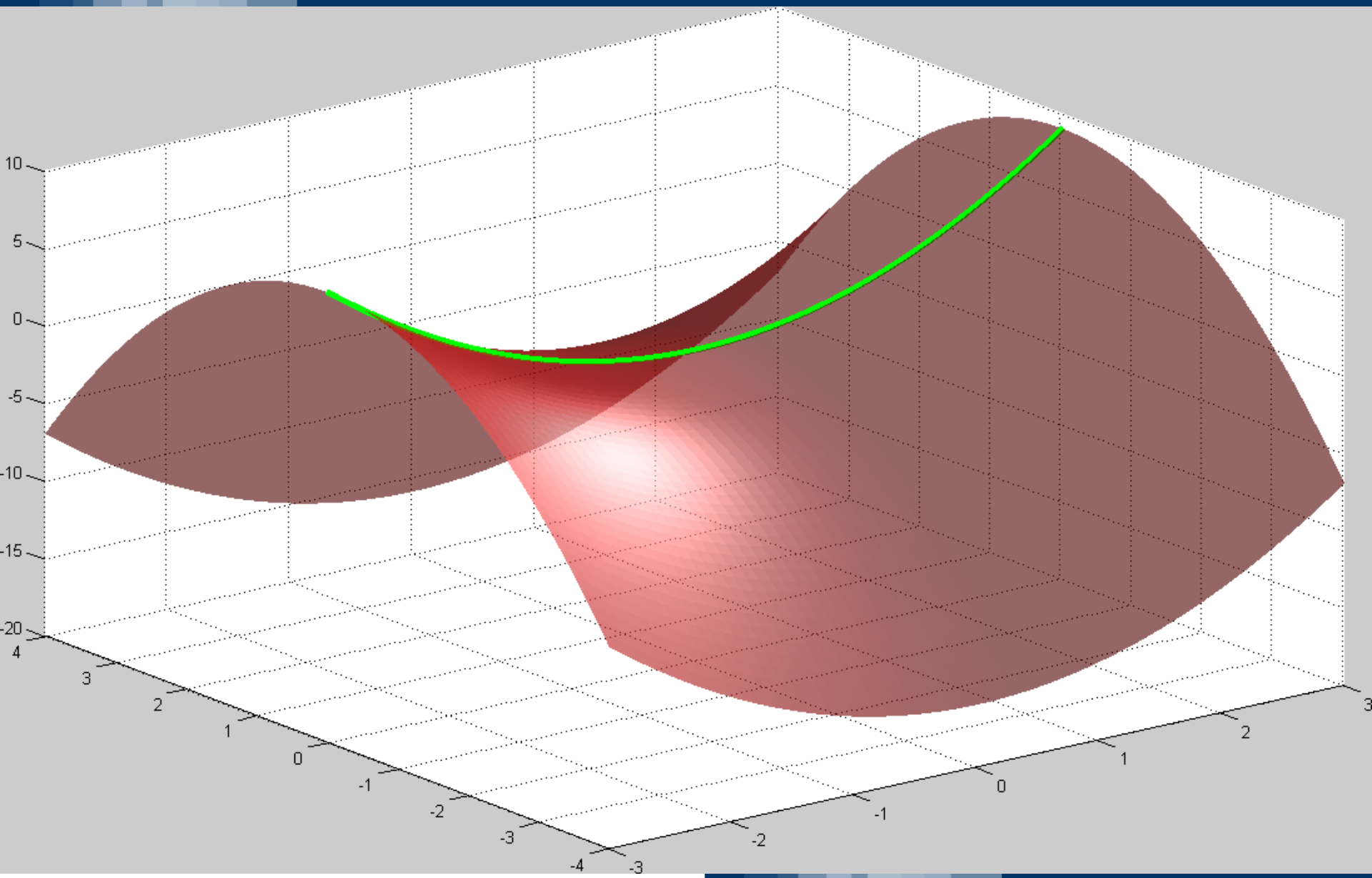
- plot3(), mesh(),altre funzioni grafiche quali surf() etc..

- 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

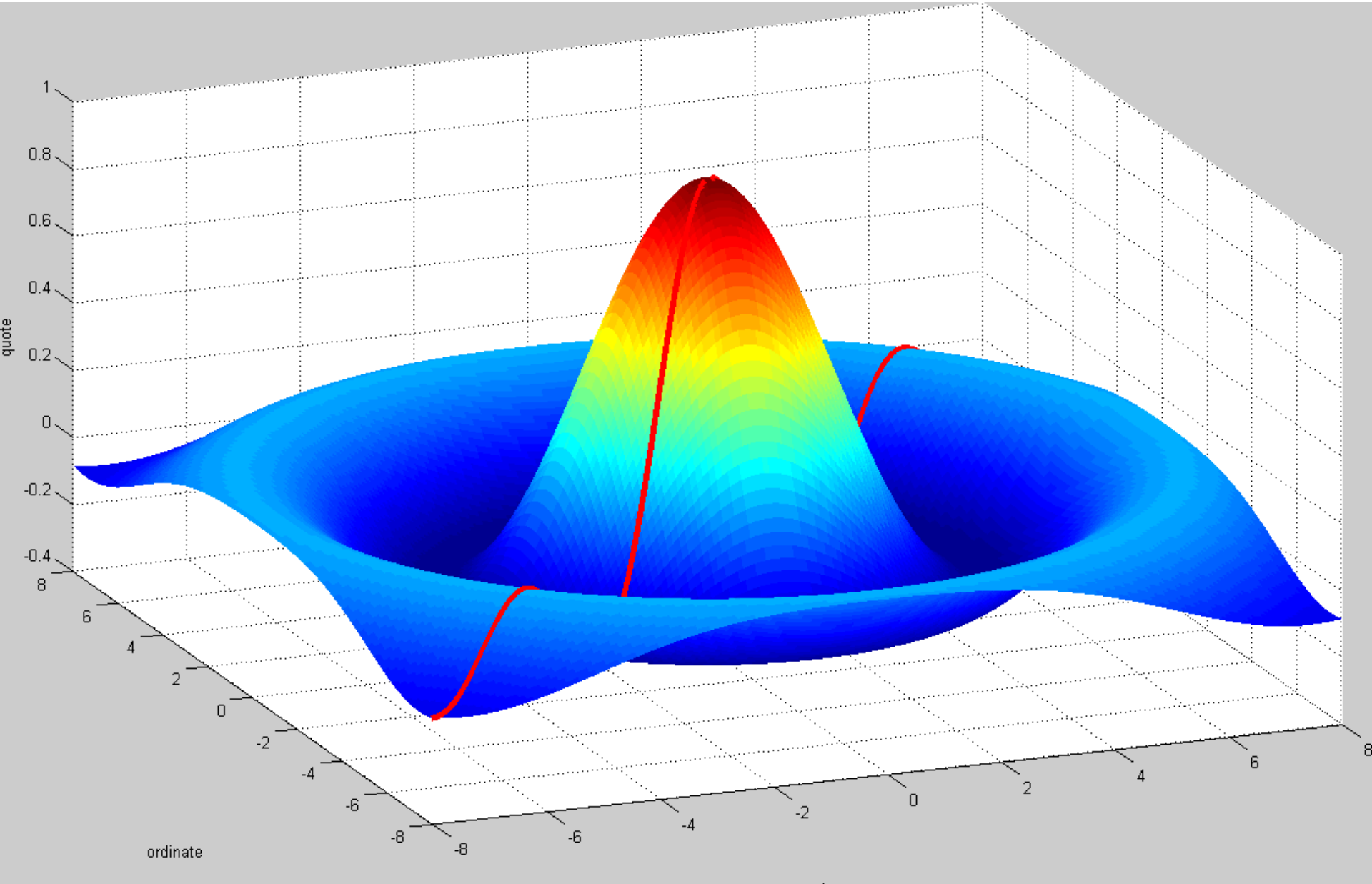
Disegnare la funzione

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

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)



Strutture in Matlab

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):
  - Assegnamento dei valori ai campi (e contestuale definizione dei campi)
  - Utilizzando la funzione struct()

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

- 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) = []

Creazione di una struttura mediante la funzione struct

`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

`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);
```

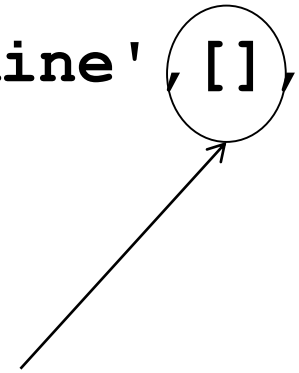
- s è un array 1x5 in cui ogni elemento ha attributi x e y
- solo il quinto elemento di s viene inizializzato con i valori x=10 e y=3
- gli altri elementi vengono inizializzato con il valore di default: [] (array vuoto)

Creazione di una struttura mediante la funzione struct

`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

Uso dei dati nelle strutture

- Notazione simile al C:

```
studente(2).nome
studente(2).esami(2)
unNome = studente(1).nome
studente(2).indirizzo=studente(1).indirizzo
%mean calcola la media degli elementi di un
array
mean(studente(2).esami)
```

Manipolazione array di strutture

E' possibile estrarre tutti valori di un campo assume in tutti gli elementi di un array di strutture.

```
studente(1).nome = 'Giovanni';
studente(1).media = 25;
Studente(2).nome = 'Pippo';
studente(2).media = 30;
```

`a = [studente.media]` → `a = [25 30]`

Le parentesi quadre sono necessarie per trasformare i dati in un array

`a = [studente.nome]` → `a = [GiovanniPippo]`

Array di strutture innestati

- Un campo di una struttura di strutture può essere di qualsiasi tipo (come in C)
- 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));
```

Soluzione con find

```
%% acquisizione dei rilievo
nRilievi = input('quanti rilievi intendi inserire? ');
for ii = 1 : nRilievi
    rilievo(ii).lat = input(['latitude ' , num2str(ii) , ' rilievo ']);
    rilievo(ii).long = input(['long ' , num2str(ii) , ' rilievo ']);
    rilievo(ii).altezza= input(['altezza ' , num2str(ii) , ' rilievo ']);
end

%% seleziona i rilievi richiesti
idx=find([rilievo.lat] >= 10 & [rilievo.lat]<=100 & [rilievo.long]>=30 &
[rilievo.long]<=60);
for ii = idx % stampa
    fprintf('Rilievo %d, Alt=%g, Lat=%g, Long=%g\n', ii, rilievo(ii).altezza,
rilievo(ii).lat, rilievo(ii).long);
end
aa = [rilievo.altezza];
altezzaMedia = mean(aa(idx));
```

È più semplice scorrere i rilievi selezionati se si usa la funzione find per selezionare le posizioni rispetto al vettore logico

Modificare i campi di un'array di strutture

È possibile estrarre tutti gli elementi che appartengono allo stesso campo di un array di strutture

- *Es* [rilievi.altezza]

Tuttavia non è possibile modificare con una sintassi simile i valori sui campi di tutti gli elementi dell'array.

- *Es* NON è possibile fare
  [rilievi.altezza] = [rilievi.altezza] + 10
  per sommare 10 a tutte le altezze di rilievi perché la parte a sx dell'uguale non è una variabile!

Occorre quindi procedere elemento per elemento.

Si potrebbe ad esempio definire una funzione come segue

Modificare i campi di un'array di strutture

```
function rilievi= sommaAlt(ril, offset)
for ii = 1 : length(ril)
    ril(ii).alt = ril(ii).alt + offset;
end
```

Che somma il valore di `offset` al valore del campo `alt` in ogni elemento dell'array `ril` (che si assume abbia tale campo)

Osservazione Importante

Tuttavia la funzione `sommaAlt` non può modificare «da sola» una variabile `rilievi` nel workspace.

È necessario a sovrascrivere `rilievi` il parametro restituito da `sommaAlt`

Occorre quindi fare la seguente chiamata

```
rilievi = sommaAlt(rilievi, 10)
```

Attenzione: questo non **vale** solo per le strutture ma **per ogni funzione**, visto che **il workspace locale** della funzione **e quello principale** sono distinti e **comunicano solo tramite i parametri attuali**

Esercizio, la roulette

```
% Scrivere un programma per simulare il gioco della roulette
% la roulette possiede 38 numeri (da 1 a 36, lo zero e il doppiozero)
% 0 e 00 non sono ne pari ne dispari (vince il banco)
%
% il banco inizialmente possiede 5000 euro
% i giocatori possiedono inizialmente 5000 euro
%
% 1) assumere ad ogni giocata che il giocatore 1 punti 5 euro su pari
% o dispari con la stessa probabilità
% se vince, giocatore1, ottiene 2 volte la posta,
% se perde il banco incassa il valore giocato.
%
%   Mostrare la variazione dell'ammontare del banco e del
%   giocatore all'aumentare delle giocate fino a che o il
%   giocatore perde il banco viene sbancato
%
```

% hints
% - utilizzare la funzione rand() per generare numeri uniformemente distribuiti in [0,1]. Riscalarli quindi in [0 , 38] e approssimarli
% - utilizzare un array di strutture per contenere i giocatori (è possibile aggiungere ulteriori campi alle strutture)
% - utilizzare, dove possibile, funzioni da voi sviluppate

Extensions

```
%
%2) aggiungere un secondo giocatore che punta sempre 1 euro
sul 15 (se esce 15 vince 36 volte la posta)
%
%3) aggiungere un terzo giocatore che usa la seguente strategia:
%   egli punta sempre sul pari e inizialmente punta un euro.
%   se vince ricomincia a puntare un euro sempre sul pari
%   se perde raddioppia la puntata sempre sul pari,
%   se non ha abbastanza soldi punta tutto quello che possiede
%
```