Funzioni e Operazioni Matriciali

Informatica (ICA) AA 2020 / 2021

Giacomo Boracchi

13 Novembre 2020

giacomo.boracchi@polimi.it

Array Vuoto

Un array vuoto si definisce così:

```
nomeVettore = []
```

Può essere una forma di dichiarazione di una variabile

```
» a = []
a =
     []

» whos a
  Name      Size            Bytes  Class     Attributes
  a           oxo               o   double
```

Cancellare Parti di un Vettore

Quando si assegna il valore [] ad un elemento di un vettore, il corrispondente elemento viene rimosso e il vettore ridimensionato: non si crea un 'buco'

```
>> a = [1 : 5]
a =
     1     2     3     4     5
>> whos a
Name  Size Bytes  Class    Attributes
  a    1x5 40     double
>> a(3) = []
a =
     1     2     4     5
>> whos a
Name  Size Bytes  Class    Attributes
  a    1x4  32    double
```

Cancellare Parti di una Matrice

L'array vuoto [] non è assegnabile a singoli elementi di matrici (non si possono "creare buchi")

```
>> m(1 : 3, 1:3) = 1
m =
     1     1     1
     1     1     1
     1     1     1
>> m(2 , : ) = 5
m =
     1     1     1
     5     5     5
     1     1     1
>> m(2,3)=[]
??? Subscripted assignment dimension
mismatch.
```

Cancellare Parti di una Matrice

È però assegnabile a intere righe o colonne di matrici, che vengono cancellate (ricompattando la matrice)

```
>> m(: , 2) = []
m =

     1     1
     5     5
     1     1
>> whos m
  Name        Size    Bytes Class    Attributes
  m           3x2        48 double
```

Memorizzazione degli Array

Gli array vengono salvati linearmente in memoria.

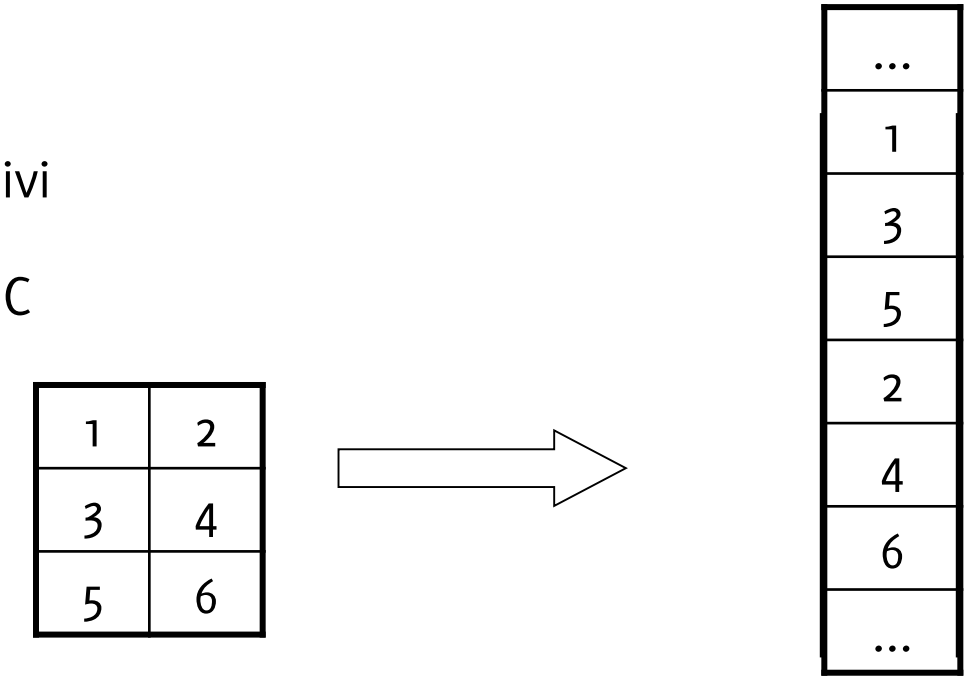
In particolare le matrici sono memorizzate

- per colonna: colonna 1, poi colonna 2, 3, etc.
- ogni colonna memorizzata per indici di riga crescenti

Array memorizzati in forma lineare nella RAM variando

- più velocemente i primi indici
- più lentamente quelli successivi

NB: opposto a quanto avviene in C



Array: forma *linearizzata*

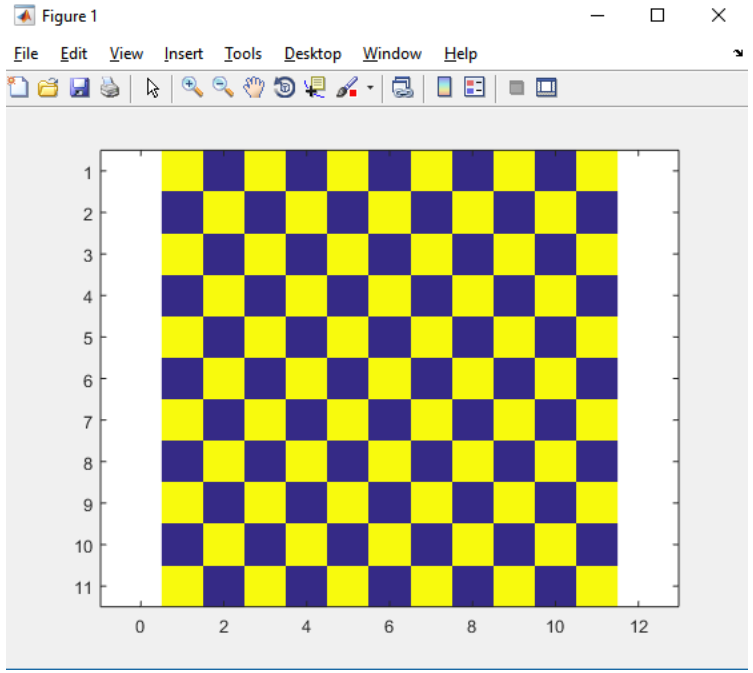
Si può accedere a un array a più dimensioni come se ne avesse una sola

Usando un unico indice si segue l'ordine della memorizzazione

```
>> a = [1 2 3; 4 5 6; 7 8 9; 10 11 12]
a =
     1     2     3
     4     5     6
     7     8     9
    10    11    12
>> a(3, 2)
ans =
     8
>> a(10)
ans =
     6
```

Esercizio

Scrivere una funzione che genera una matrice quadrata $n \times n$ di 0 ed 1 disposti «a scacchiera». La funzione controlla anche che n sia dispari



Tipo di Dato Logico

...e operazioni su vettori

Tipo di Dato Logico

È un tipo di dato che può avere solo due valori

- true (vero) 1
- false (falso) 0

I valori di questo tipo possono essere generati

- direttamente da due funzioni speciali (true e false)
- dagli operatori relazionali
- dagli operatori logici

I valori logici occupano un solo byte di memoria (i numeri ne occupano 8)

Esempi

```
>> a = true;
>> whos a
 Name  Size    Bytes  Class     Attributes
  a    1x1      1     logical
```

a è un vettore 1x1 che occupa 1 byte e appartiene alla classe “tipo logico”

```
>> a = 1>7
a =
     0
```

Operatori Relazionali

Operano su tipi numerici o stringhe.

Possono essere usati per confrontare

- due scalari
- due vettori aventi la stessa dimensione

Forma generale: a OP b

- a,b possono essere espressioni aritmetiche, variabili, stringhe (della stessa dimensione)
- OP: **==, ~=, >, >=, <, <=**

Esempi:

- **3<4** true(1)
- **3==4** false(0)
- **'A'<'B'** true(1)

Note

Come in C: non confondere == e =

- == è un operatore di confronto
- = è un operatore di assegnamento

La precisione finita può produrre errori con == e ~ =

- sin(0) == 0 → 1
- sin(pi) == 0 → 0
- eppure logicamente sono vere entrambe!!

Per i numeri piccoli conviene usare una soglia

- abs(sin(pi)) < = eps

Vettori e stringhe

Gli operatori relazionali tra vettori vengono applicati in maniera **puntuale**

Il risultato di un confronto tra v1 e v2 è un vettore v3 di tipo boolean, aventi le stesse dimensioni di v1 (e v2)

$$v3 = (v1 >= v2); \quad v3(i) = \begin{cases} 1, & \text{se } v1(i) \geq v2(i) \\ 0, & \text{se } v1(i) < v2(i) \end{cases}$$

Esempi:

```
>> [1 0; -2 1] < 0    [false false; true false] ([0 0; 1 0])
>> [1 0; -2 1] >= [2 -1; 0 0]      [false true; false true]
```

Si possono confrontare stringhe di lunghezza uguale

```
>> 'pippo'=='pluto'
 ans = [1 0 0 0 1]
```

Operatori Logici: Forma Generale

Operatori binari: **AND** (**&&**, oppure **&**), **OR** (||, oppure |), **XOR** (**xor**):

a OP1 b per la notazione simbolica
OP(a,b) per la notazione testuale

Operatori unari: NOT (~):

OP2 a

a,b possono essere variabili, costanti, espressioni da valutare, scalari o vettori (dimensioni compatibili)

Valori numerici di a, b vengono interpretati come logici:

- 0 come falso
- tutti i numeri diversi da 0 come vero

Richiamo, Tabelle di Verità

a	b	a && b	a || b	~a	**xor**(a, b)
0	0	0	0	1	0
0	1	0	1	1	1
1	0	0	1	0	1
1	1	1	1	0	0

Or esclusivo: vero quando è vera
solo uno delle due espressioni coinvolte
a XOR b == a OR b AND (~(a AND b))

&& vs & e || vs |

&& (**||**) **funziona con gli scalari** e valuta prima l'operando più a sinistra. Se questo è sufficiente per decidere il valore di verità dell'espressione non va oltre

- **`a && b`**: se **a** è falso non valuta **b**
- **`a || b`**: se **a** è vero non valuta **b**

& (**|**) **funziona con scalari e vettori** e valuta **tutti** gli operandi prima di valutare l'espressione complessiva

Esempio: **`a / b > 10`**

- se **b** è o non voglio eseguire la divisione
- **`(b~=0)&&(a/b>10)`** è la soluzione corretta: **`&&`** controlla prima **`b~=0`** e se questo è falso non valuta il secondo termine. Invece **`(b~=0)&(a/b>10)`** porterebbe ad una divisione per o quando **`b == 0`**

Esempi

"Hai tra 25 e 30 anni?"

```
(eta >= 25) & (eta <= 30)
```

Con i vettori:

```
Voto = [12, 15, 8, 29, 23, 24, 27]
 C = (Voto > 22) & (Voto < 25)
-> C = [0 0 0 0 1 1 0]
 D = (mod(Voto,2) == 0) | (Voto > 18)
-> D = [1 0 1 1 1 1 1]
 E = xor((mod(Voto,2)==0), (Voto>18))
-> E = [1 0 1 1 1 0 1]
```

Utile per contare quanti elementi soddisfano una condizione

```
nVoti= sum (Voto > 22 & Voto < 25)
```

Precedenze tra gli Operatori

Ogni espressione logica viene valutata rispettando il seguente ordine:

- operatori aritmetici
- operatori relazionali da sinistra verso destra
- NOT (**~**)
- AND (**&** e **&&**) da sinistra verso destra
- OR (**|** e **||**) e XOR da sinistra verso destra

Vettori Logici per Selezionare Sottovettori

I vettori logici possono essere usati per selezionare gli elementi di un array al posto di un vettore di indici

`nomeVettore(vettoreLogico)`

- vengono estratti gli elementi di **`nomeVettore`** alle posizioni per cui **`vettoreLogico`** vale 1

Per esempio

```
>> x = [6,3,9]; y = [14,2,9];
>> b = x<=y ; % b = 1     0     1
>> z = x(b)
z =
     6     9
```

Esempi

Inizializzare a con i numeri da -10 a 20 con passo 3

Visualizzare solamente i numeri maggiori di 10

Portare a zero tutti gli elementi negativi

Sommare 10 ai numeri minori di 10

Cambiare il segno a tutte le occorrenze di -7 o 17

Esempi

Inizializzare a con i numeri da -10 a 20 con passo 3

```
>> a = [-10 : 3 : 20]
```

Visualizzare solamente i numeri maggiori di 10

```
>> a(a > 10);
```

Portare a zero tutti gli elementi negativi

```
>> a(a < 0) = 0;
```

Sommare 10 ai numeri minori di 10

```
>> a(a < 10) = a(a < 10) + 10;
```

Cambiare il segno a tutte le occorrenze di -7 o 17

```
>> a(a == -7 | a == 17) = -a(a == -7 | a == 17);
```

NB qui non si può usare ||

Note

`nomeVettore` e **`vettoreLogico`** devono avere la **stessa dimensione**

Per **creare un vettore logico non basta** creare un **vettore di 0 e 1** (numeri), bisogna convertirlo con la funzione logical

```
>> ii = [1,0,0,0,1];
>> jj = (ii == 1); %oppure jj = logical(ii)
>> A = [1 2 3 4 5];
>> A(jj) → [1 5]
>> A(ii) → Subscript indices must either be real positive
     integers or logicals.
```

Una nota sul costrutto if

`espressione1` può coinvolgere vettori:

- in tal caso **`espressione1`** è vera solo se tutti gli elementi di **`espressione1`** sono non nulli

Esempio

```
v = input('inserire vettore: ');
if (v >= 0)
    disp([num2str(v),' tutti pos. o nulli']);
elseif(v<0)
    disp([num2str(v),' tutti negativi']);
else
    disp([num2str(v),' sia pos. che neg.']);
end
```

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 non nulli dell'array **x**. x può essere un'espressione logica.

Esempio

```
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]
```

Altro su Funzioni

Return

Non necessario in Matlab,

- I valori ritornati sono definiti dall'header della funzione

Tuttavia può essere usata per terminare l'esecuzione della funzione

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

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

Esercizio

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

Esercizio

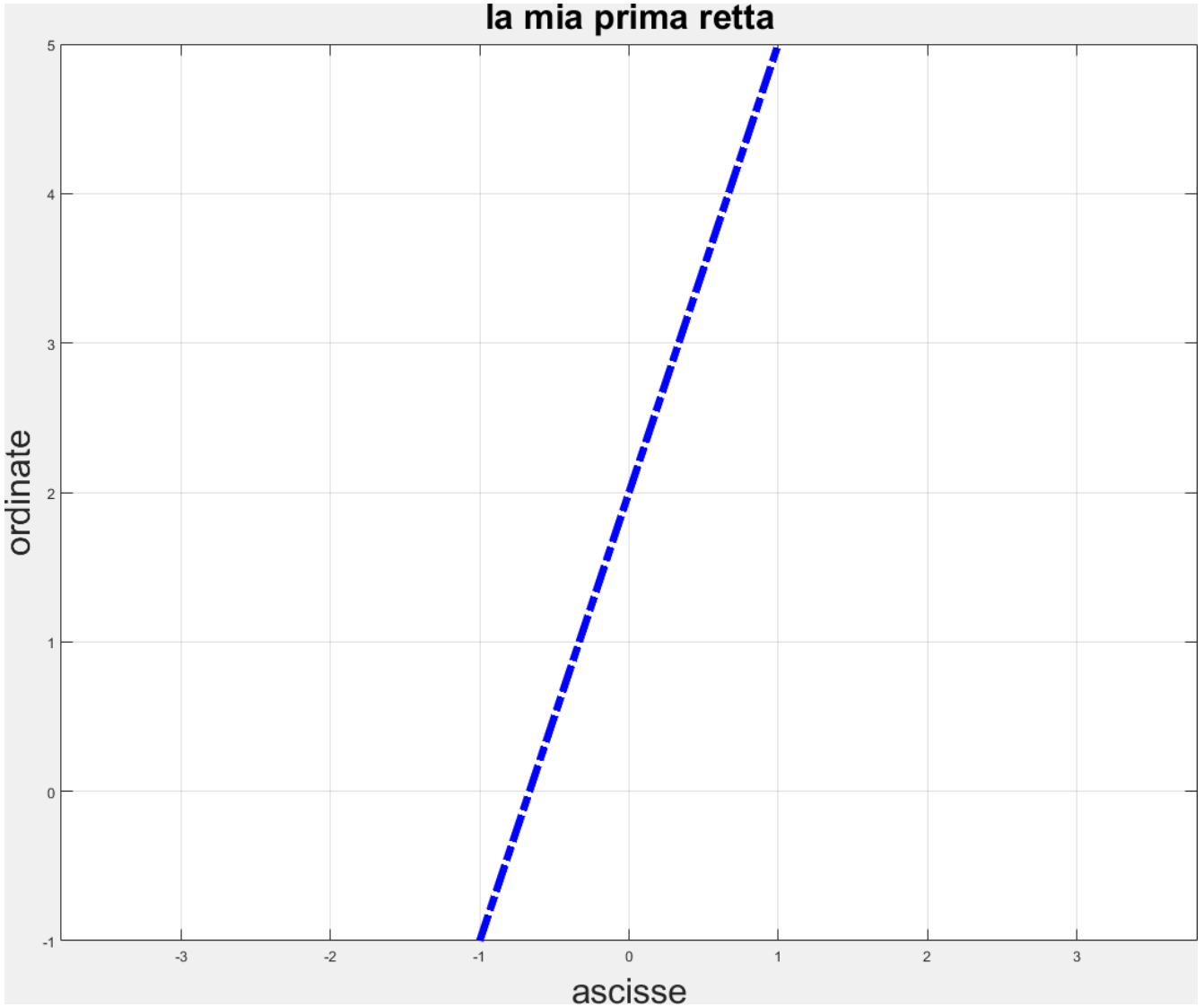
Scrivere una funzione che prende in ingresso due coefficienti m, q ed un vettore di punti xx e restituisce il vettore yy dei punti che stanno sulla retta $y = mx + q$ in corrispondenza a xx

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

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

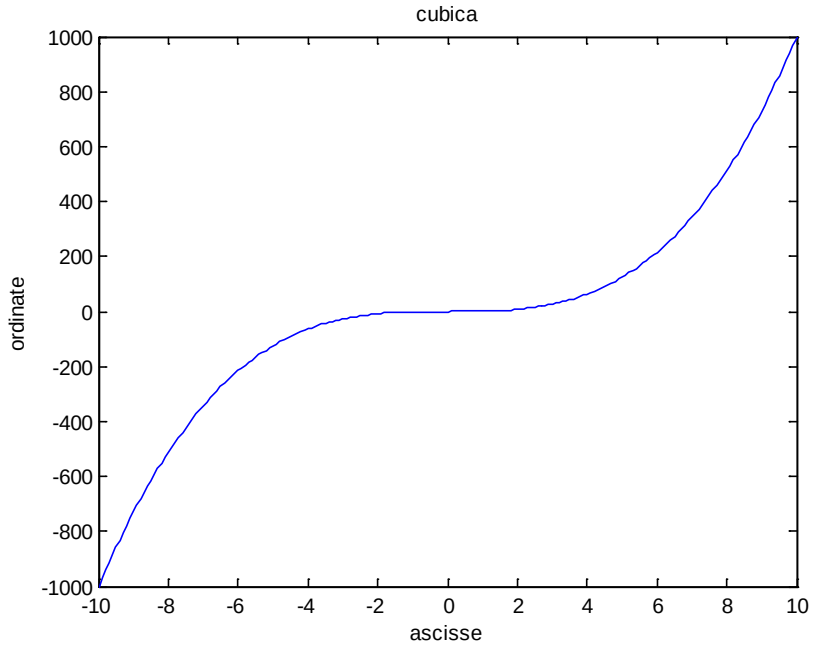
Esempi di script per invocare la funzione

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

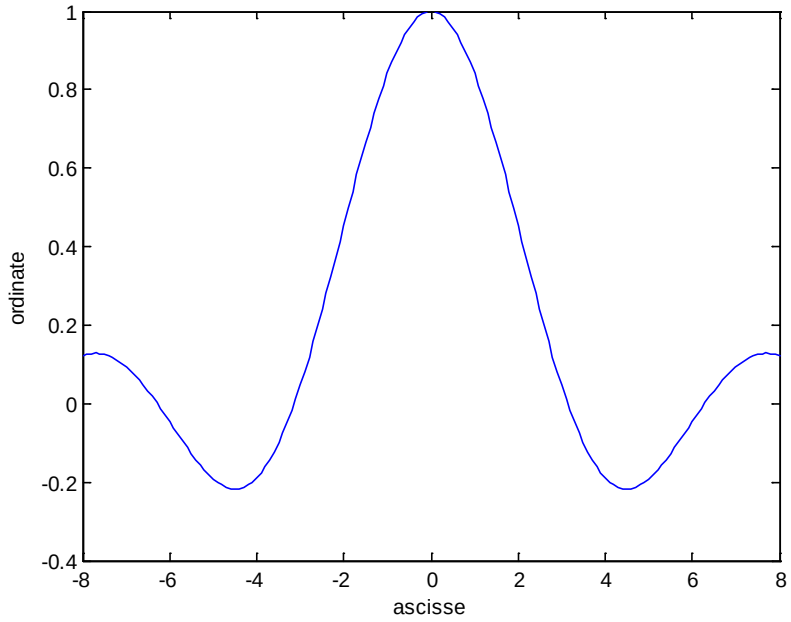

la mia prima retta
ordinate
ascisse
5
4
3
2
1
0
-1
-3
-2
-1
0
1
2
3

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');
```



Esempi

Definire una funzione *samplePolynomial* che prende in ingresso

- un vettore di coefficienti C
- un vettore che definisce un intervallo [a,b]

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

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

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

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

visualizzare, per ogni valore di x, la curva maggiore

Diagrammi a due dimensioni: ancora esempi

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

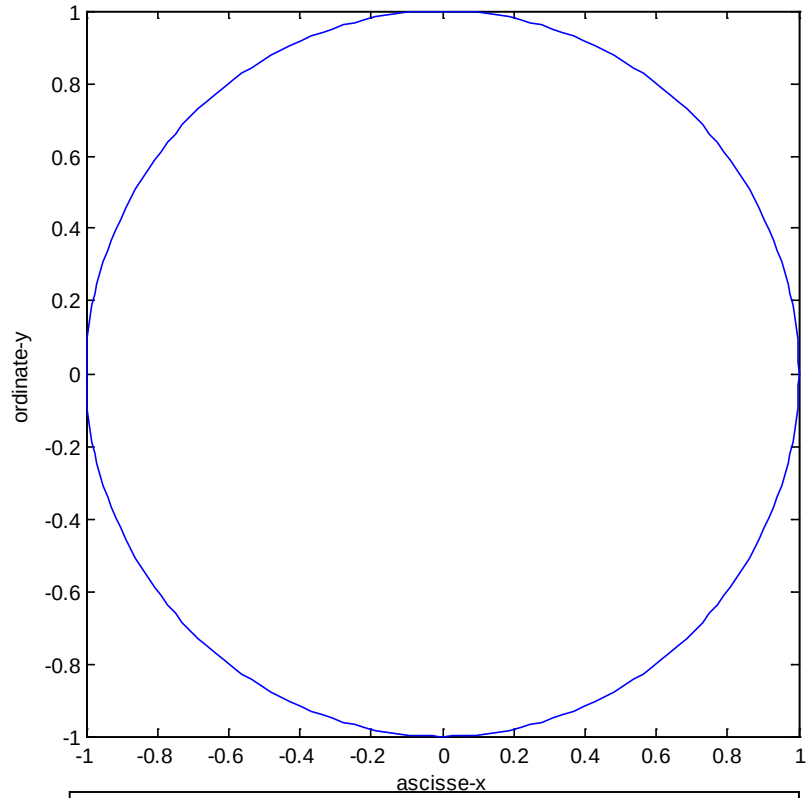
In `plot(x,y)` non necessariamente `x` contiene valori equispaziati e `y` non è necessariamente funzione di `x`. Sia `x` che `y` possono essere, ad esempio, funzioni di qualche altro parametro.

Che diagrammi disegnano i seguenti esempi?

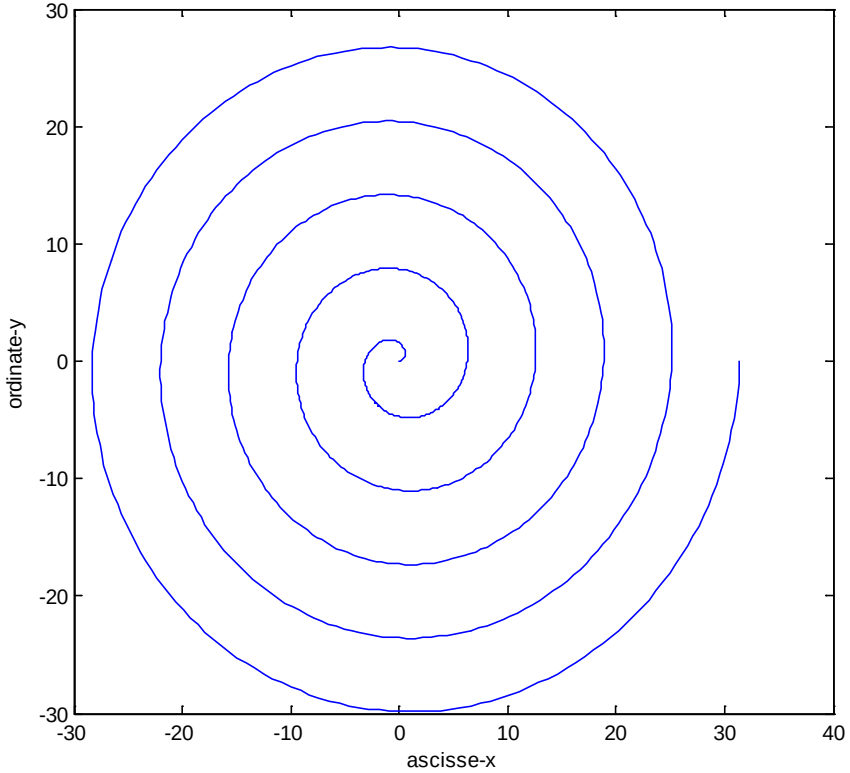
```
>> t=[0:pi/100:2*pi];
>> x=cos(t);
>> y=sin(t);
>> plot(x,y);
>> xlabel('ascisse-x');
>> ylabel('ordinate-y');
```

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

Esempi




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




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

Diagrammi lineari a tre dimensioni

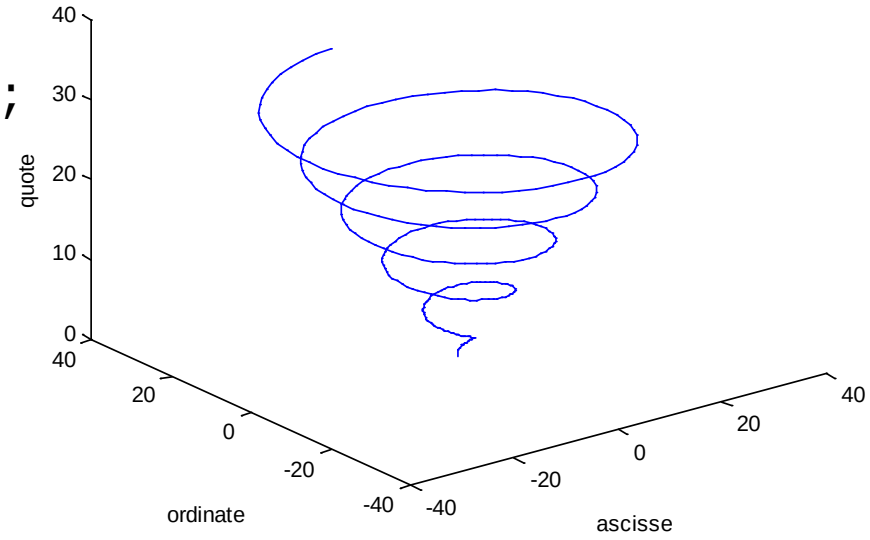
Generalizzazione del diagramma a due dimensione: insieme di terne di coordinate

`plot3(x,y,z)` disegna un diagramma cartesiano con `x` come ascisse, `y` come ordinate e `z` come quote

funzioni `xlabel, ylabel, zlabel, title`

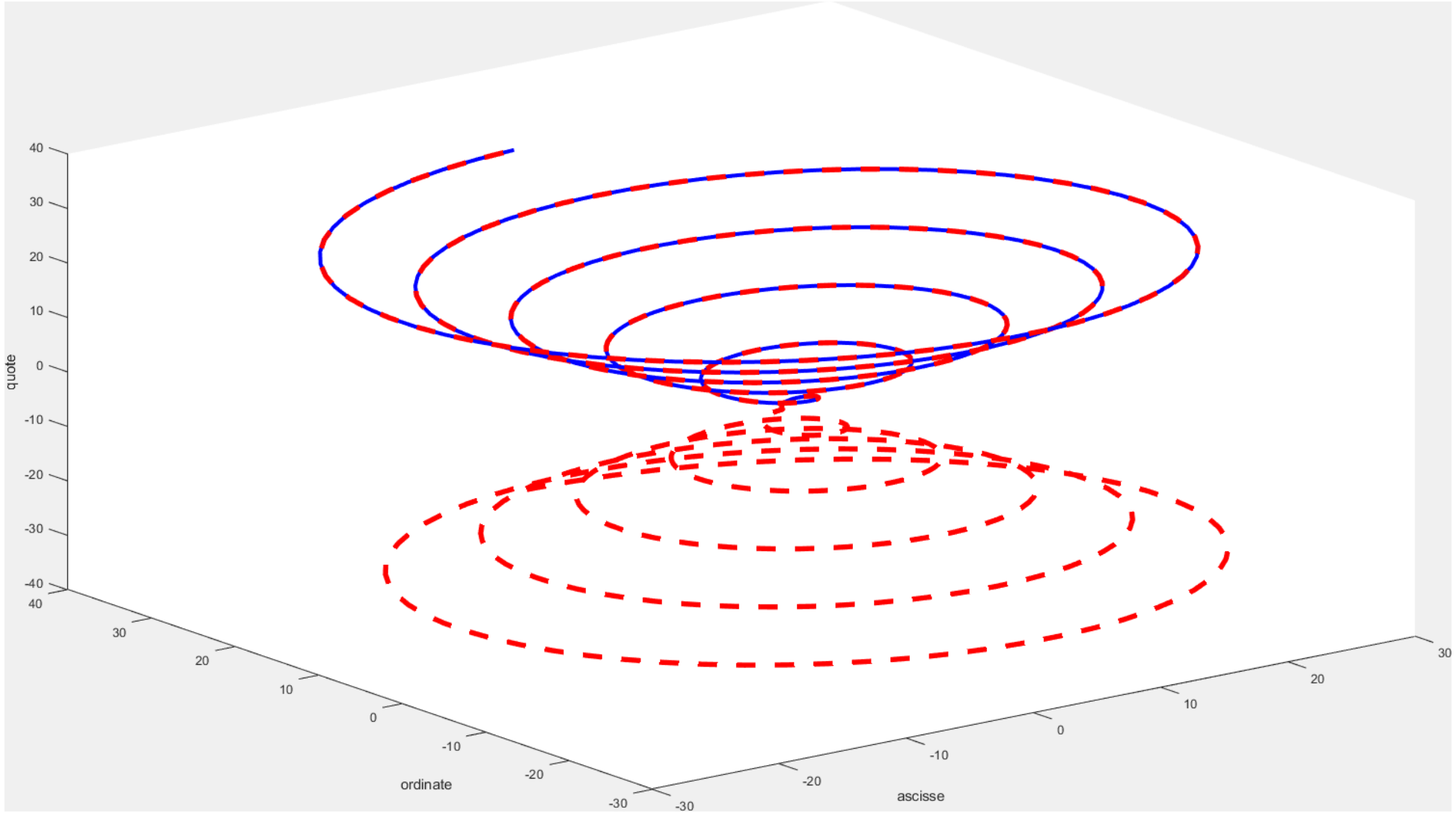
Esempio

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



Cosa fa?

```
figure(2),
t = 0: 0.1 : 10*pi;
plot3(abs(t).*sin(t), abs(t).*cos(t), t, 'b-','LineWidth',
3);
hold on
t = [-t(end : -1 : 1), t];
plot3(abs(t).*sin(t), abs(t).*cos(t), t, 'r--','LineWidth',
4);
xlabel('ascisse');
ylabel('ordinate');
zlabel('quote');
hold off
```


quote
40
30
20
10
0
-10
-20
-30
-40
ordinate
40
30
20
10
0
-10
-20
-30
ascisse
-30
-20
-10
0
10
20
30

Superfici

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

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

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

`xx` e `yy` identificano una griglia in corrispondenza del quale per `zz` rappresenta il valore della funzione in corrispondenza di quell'ascissa e di quell'ordinata

Funzione meshgrid

Le due matrici, xx, e yy, si possono costruire, mediante la funzione `meshgrid(x,y)`

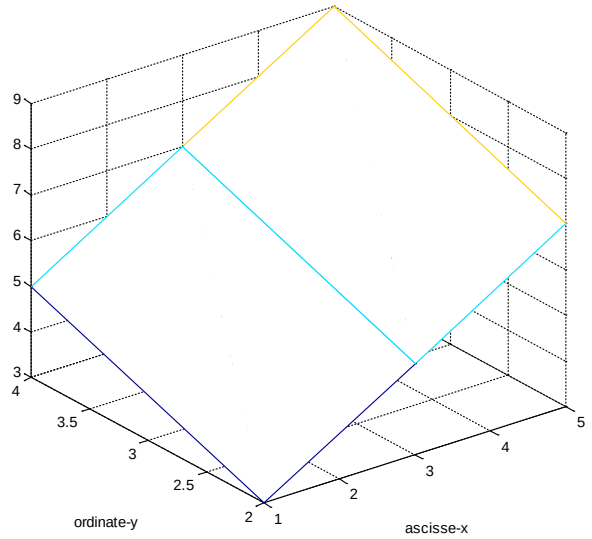
`[xx,yy]= meshgrid(x,y)`

- `x e y` sono due vettori
- `xx` e `yy` sono due matrici entrambe di legth(y) righe e length(x) colonne
- la prima, `xx`, contiene, ripetuti in ogni riga, i valori di `x`
- la seconda, `yy` , contiene, ripetuti in ogni colonna, i valori di `y` trasposto

Superfici: esempi

Disegnamo z=x+y

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




```
›› xx
xx =
     1     3     5
     1     3     5
```

```
›› yy
yy =
     2     2     2
     4     4     4
```

Punti di coordinate (x,y)…

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

```
›› zz
zz =
     3     5     7
     5     7     9
```

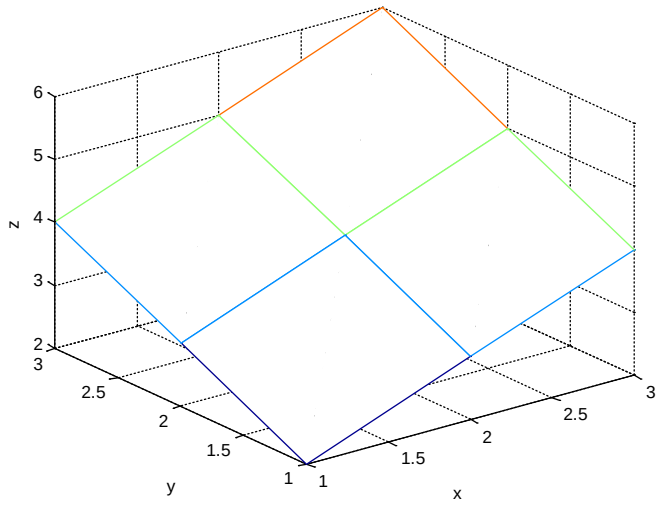
…hanno coordinate (x,y,z)

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

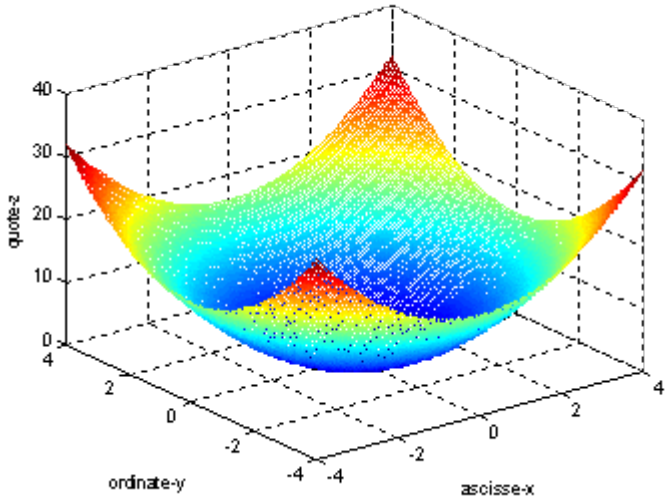
(NB: z=x+y)

Superfici: esempi (2)

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




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



Hold on

Le superfici vengono visualizzate su un grafico 3D.

È quindi possibile aggiungere degli elementi in sovraimpressione utilizzando la funzione

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

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

Esempio, disegnare in sovraimpressione al paraboloide precedente la curva $\begin{cases} z = x^2 \\ y = 0 \end{cases}$

dove `x=[-4:0.05:4];`

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

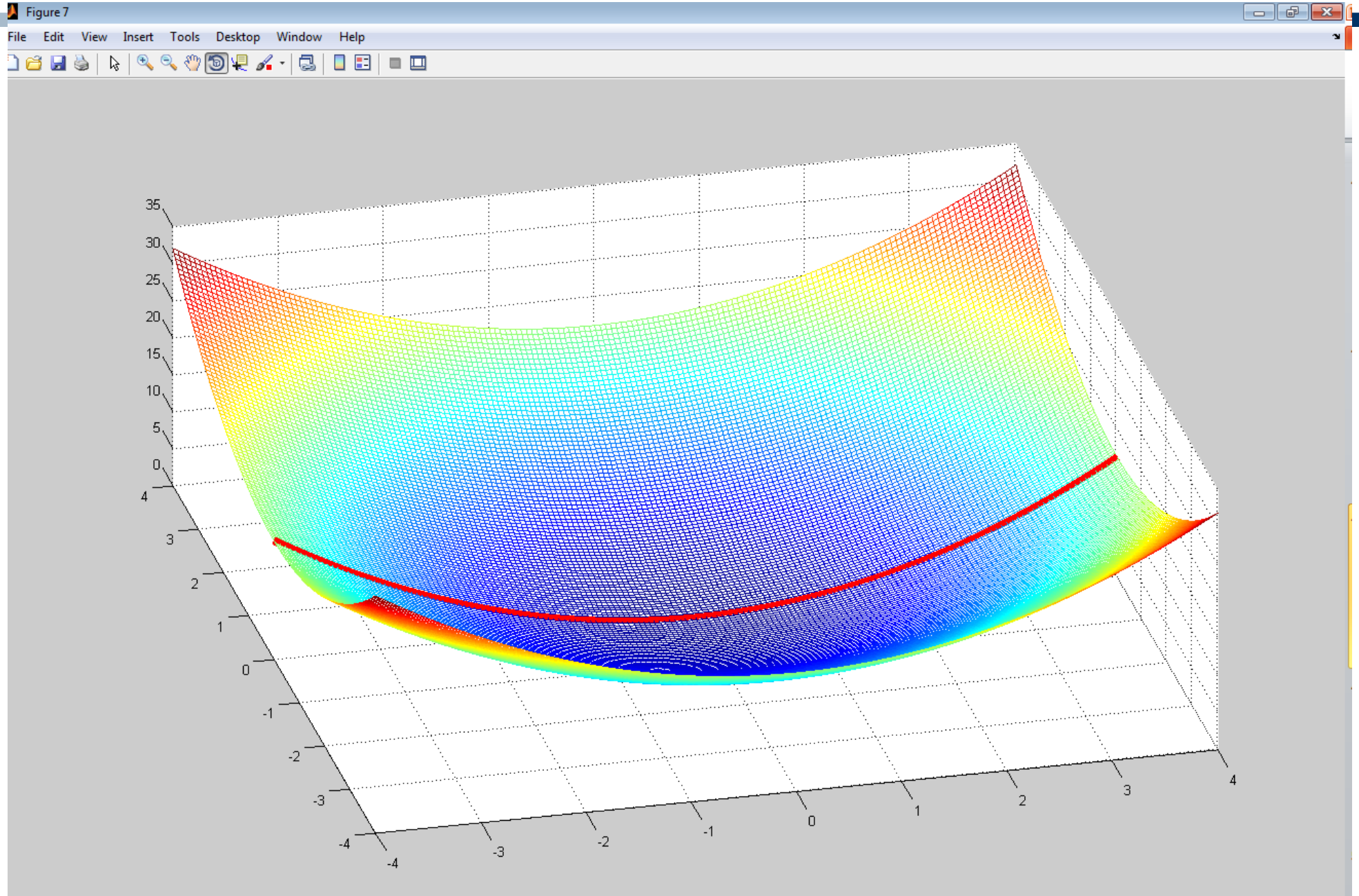
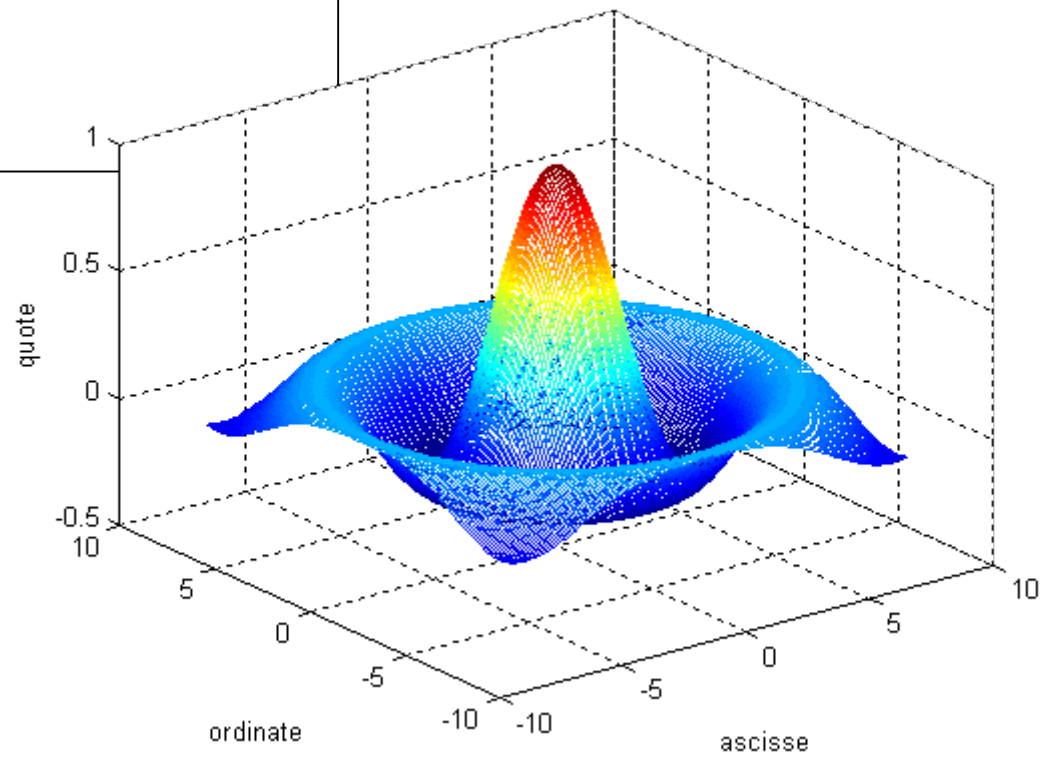


Figure 7
File Edit View Insert Tools Desktop Window Help
35
30
25
20
15
10
5
0
4
3
2
1
0
-1
-2
-3
-4
-4 -3 -2 -1 0 1 2 3 4

Superfici: esempi (3)

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



Strutture in Matlab

Struct vs Array

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

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

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

Esempio: variabile per contenere anagrafica di impiegati

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

Creazione di una struct

Creazione di una struttura :

Utilizzando la funzione struct()

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

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

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

Accedere ai campi di una `struct`

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

```
nomeStruct.nomeCampo;
```

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

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

Creazione di una struttura campo per campo

Esempio: creo una struttura studente

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

Accesso ai campi come nel C con l'operatore .

`nomeStruttura.nomeCampo`

Es

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

Creazione di una struttura campo per campo

Esempio: la struttura studente

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

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

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

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

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

Aggiunta di campi

Aggiunta di un campo

```
%facciamo riferiemento alla definizione di studente delle slide precedenti
studente(2).esami = [20 25 30];
```

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

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

Creazione di una struttura mediante la funzione struct

Consente di preallocare una struttura o un array di strutture

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

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

Creazione di una struttura mediante la funzione struct

Consente di preallocare una struttura o un array di strutture

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

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

Esempio array di strutture:

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

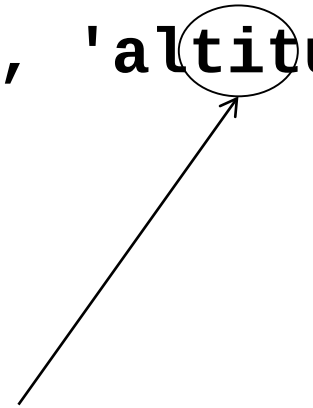
- s è un array 1x5 in cui ogni elemento ha attributi x e y
- solo il quinto elemento di s viene inizializzato con i valori x=10 e y=3
- gli altri elementi vengono inizializzato con il valore di default: [] (array vuoto)

Creazione di una struttura mediante la funzione struct

Consente di preallocare una struttura o un array di strutture

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

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



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

Array di strutture innestati

Un campo di una struttura può essere di qualsiasi tipo

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

Esempio

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

corso è un array di strutture

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

Esercizio

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

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

Soluzione

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

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

Array di Strutture

In Matlab gli array di strutture vengono gestiti allo stesso modo dell'array numerici e delle stringhe

- È possibile estendere l'array mediante assegnamento
  - Es: **`s(7) = s(2);`**
- È possibile estrarre sotto-vettori mediante indicizzazione
  - Es **`t = s(goodIndexes);`**
- È possibile rimuovere elementi da un array di strutture con l'assegnamento al vuoto
  - Es **`s(badIndexes) = []`**

Array di Strutture: vincoli

Attenzione: valgono i vincoli degli array:

Tutti gli elementi di un array di strutture **devono essere omogenei**

In paticolare, **tutte** le strutture nello stesso array devono avere:

- Lo stesso numero di campi
- Tutti i campi con lo stesso nome

(viene tollerato invece un diverso ordinamento dei campi)

```
>> s = struct('a', 10, 'b', 11)
>> t = struct('c', 10, 'a', 11)
>> s(2) = t
Subscripted assignment between dissimilar structures.
```

Array di Strutture: vincoli

Nota bene: non è necessario che il contenuto dei campi sia dello stesso tipo!

```
s = struct('a', 'pippo', 'b', ones(3))
t = struct('a', ones(3,1), 'b', [])
s(2) = t;
 >> s(1)
    a: 'pippo'
    b: [3×3 double]
>> s(2)
    a: [3×1 double]
    b: []
```

Se necessario quindi si creano campi vuoti (i.e. uguali a `[]`) struttura prima di concatenare una struttura in un array di strutture diverse

Array di Strutture: particolarità

È possibile accedere rapidamente a tutti i valori di un campo in un array di strutture

```
>> s(1) = struct('a', 'pippo', 'b', 3)
>> s(2) = struct('a', ones(3,1), 'b', 4)
>> s.b
```

L'ultimo comando restituisce

```
ans =
     3
ans =
     4
```

..e quindi non concatena automaticamente un array!

Array di Strutture: particolarità

Il motivo è che le dimensioni potrebbero non essere consistenti! Si pensi ad esempio

```
>> s.a
ans =
    'pippo'
ans =
     1
     1
     1
```

Questi non risultano concatenabili!

Array di Strutture: particolarità

Tuttavia, è possibile **forzare il concatenamento in un array** «a proprio rischio e pericolo», consapevoli che **questo potrebbe sollevare errori**

```
>> v = [s.b]

v =

     3     4

>> v = [s.a]

v =

     3     4

Error using horzcat

Dimensions of matrices being concatenated are not
consistent.
```

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

```
%
%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
%
```