Introduzione all'ambiente

MATLAB©

Approssimazione di funzioni e dati in 1D, 2D e 3D.

Rappresentazione di dati su griglie strutturate e no.

Lezione 3 *Paola Gervasio*

Caso monodimensionale: 1D

Esempio: un problema di climatologia.
La temperatura t in prossimità del suolo varia al variare della concentrazione K dell'acido carbonico e della latitudine L. In particolare, per $K = 1.5$, la temperatura al suolo subirebbe una variazione (Δt) dipendente dalla latitudine come descritto in tabella:

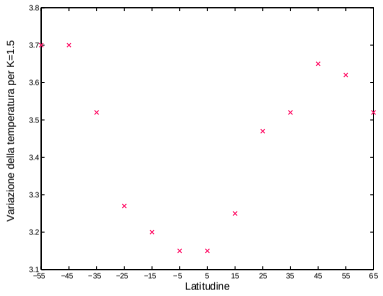
L	-55	-45	-35	-25	-15	-5	
Δt	3.7	3.7	3.52	3.27	3.2	3.15	
L	+5	+15	+25	+35	+45	+55	+65
Δt	3.15	3.25	3.47	3.52	3.65	3.62	3.52

Come valutare Δt a Roma (per $L = +42$)?

Si vuole costruire un modello ($\equiv$ una funzione) che descriva la legge $\Delta t = \Delta t(L)$ anche per le latitudini non riportate in tabella.

Passo 1.
Rappresentazione grafica dei dati del problema.

```
x=...;                                  (x vettore delle latitudini L )
y=...;                                  (y vettore dei Δt)
plot(x,y,'rh');
xlabel('Latitudine','Fontsize',14)
ylabel('Variazione della temperatura per K=1.5','Fontsize',14)
```



$(x_i, y_i) \in \mathbb{R}^2$
per $i = 1, ..., 13$

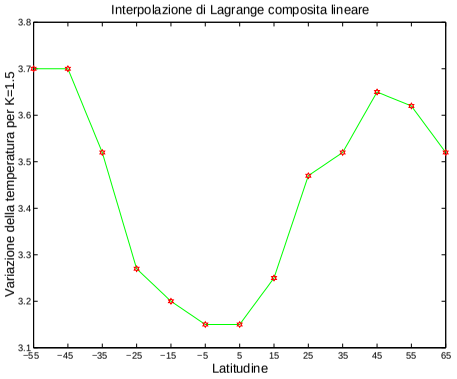
Passo 2. *Costruzione del modello*
1^a possibilità: Interpolazione lineare a tratti

L'idea è costruire una *"spezzata"* detta anche *"polinomio lineare a tratti"* che congiunga i punti (x_i, y_i) in maniera sequenziale.

```
>> x1=(-55:1:65)';          Si deve costruire un vettore x1
                            di punti "campione" piú fitti
                            rispetto a quelli di x
>> y1=interp1(x,y,x1);      Per valutare la spezzata nei
                            punti di x1
>> roma=interp1(x,y,42);
>> fprintf('Variazione di temperatura a Roma= %6.4e \n',roma)

>> hold on; plot(x1,y1)     Per disegnare la spezzata sullo
                            stesso grafico dei dati
```

Non vengono dati i coefficienti associati alla forma analitica della funzione. Il modello può solo essere valutato per punti.



Il polinomio lineare a tratti è continuo, ma con derivata prima discontinua nei nodi x_i.

2^a possibilità: Interpolazione polinomiale globale di Lagrange

Si costruisce un polinomio $p_{12}(x)$ di grado 12 che **"interpoli"** i dati, ovvero t.c.

$$p_{12}(x_i) = y_i \qquad i = 1, ..., 13$$

C'è un teorema che assicura che dati $(n+1)$ punti distinti $(x_i, y_i) \in \mathbb{R}^2$ esiste un solo polinomio

$$p_n(x) = c_1 x^n + c_2 x^{n-1} + ... + c_n x + c_{n+1} \qquad (1)$$

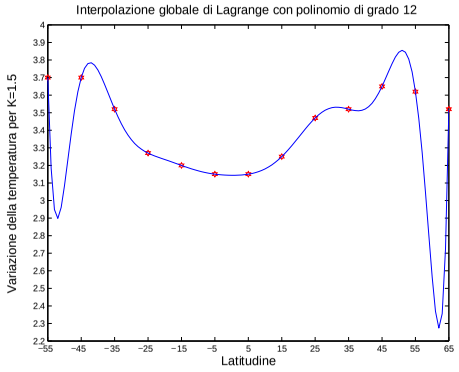
di grado n che "passa" per questi punti. Questo polinomio viene detto *polinomio interpolatore di Lagrange dei dati* (x_i, y_i).

`>> c=polyfit(x,y,12);` *Per costruire $p_{12}(x)$: `c` è un vettore riga di 13 elementi contenente i coefficienti c_i del polinomio di Lagrange (1).*

`>> y2=polyval(c,x1);` *Per valutare il polinomio $p_{12}(x)$ nei punti di `x1`*

```
>> roma2=polyval(c,42);
>> fprintf('Variazione di temperatura a Roma= %6.4e \n',roma2)
```

`>> figure(2);`
`>> plot(x,y,'hr',x1,y2)` *Per disegnare il polinomio $p_{12}(x)$ e i punti di interpolazione*



Osservazione 1.
Il polinomio passa per i punti dati, è C^∞, ma ha forti oscillazioni verso le estremità dell'intervallo (è un problema legato all'interpolazione globale di Lagrange quando i nodi x_i sono equispaziati).

Osservazione 2.
Dopo aver eseguito il comando `c=polyfit(x,y,12);`
Matlab stampa a video un messagio:

```
Warning: Matrix is close to singular or badly scaled.
         Results may be inaccurate. RCOND = 1.912183e-22.
> In /usr/local/matlab5/toolbox/matlab/polyfun/polyfit.m
        at line 52
```

Infatti, per calcolare il vettore dei coefficienti `c`, la function `polyfit` risolve un sistema lineare del tipo $A\mathbf{c} = \mathbf{b}$, la cui matrice A è "mal condizionata". Questo puó portare a risultati inaccurati.

Controllo quanto vale il polinomio p_{12} nei nodi `x` e ne faccio la differenza con i dati

```
>> yy2=polyval(c,x);  res2=y-yy2;
>> [x', res2']
res2 =
  -1.3767e-14
  -7.5939e-14
  -1.5099e-14
  -1.3323e-15
  -4.4409e-16
   2.6645e-15
  -4.4409e-16
            0
  -4.4409e-16
  -2.2204e-15
   2.2204e-15
   9.3259e-15
   8.9706e-14
```


`res2` viene detto *residuo* e fornisce una misura della bontà dell'approssimazione fatta. Qui abbiamo:

$$\max_{i=1,\dots,13} |\texttt{res2}_i| = \max_{i=1,\dots,13} |y_i - p_{12}(x_i)| = 8.9706e-14.$$

Il modello costruito non è buono: sia per le oscillazioni, sia per il fatto che in alcuni nodi il residuo è di due ordini maggiore della precisione di macchina (`eps=2.2204e-16`).

Osservazione 3. L'interpolazione polinomiale globale di Lagrange produce una funzione molto regolare, ma ci sono problemi di stabilità numerica e problemi di approssimazione. L'idea è di tornare verso una interpolazione composita.

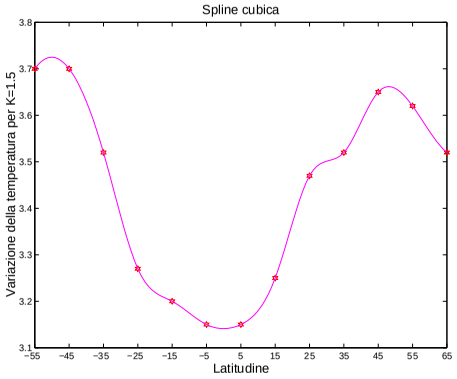
3^a possibilità: interpolazione con spline cubiche

Si costruisce una funzione che globalmente è di classe C^2 (continua con derivate prime e seconde continue) e su ogni intervallino è un polinomio di grado 3.

```
y3=interp1(x,y,x1,'spline');
roma3=interp1(x,y,42,'spline');
fprintf('Variazione di temperatura a Roma= %6.4e \n',roma3)
figure(3)
plot(....)
yy3=interp1(x,y,x,'spline');
res3=y-yy3
```

In questo caso il residuo ha tutte componenti nulle.

```
res3 =
   -55     0
   -45     0
   -35     0
   -25     0
   -15     0
    -5     0
     5     0
    15     0
    25     0
    35     0
    45     0
    55     0
    65     0
```



4^a possibilità: interpolazione globale di Lagrange su alcuni nodi

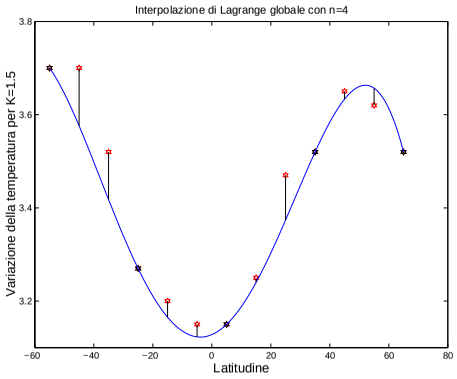
Costruisco due vettori ridotti di dati (5 dati) e costruisco il polinomio di interpolazione globale di Lagrange di grado 4:

```
xr=[-55,-25,5,35,65]'; yr=[3.7,3.27,3.15,3.52,3.52]';
cr=polyfit(xr,yr,4);
y4=polyval(cr,x1);
figure(4)
plot(x,y,'rh',xr,yr,'gh',x1,y4)
roma4=polyval(cr,42);
fprintf('Variazione di temperatura a Roma= %6.4e \n',roma4)
```


Calcolo il residuo:

```
>> yy4=polyval(cr,x); res4=y-yy4
```

```
res4 =
  -4.4409e-16
   1.2387e-01
   1.0226e-01
   4.4409e-16
   3.4733e-02
   2.6955e-02
   4.4409e-16
   1.0041e-02
   9.6091e-02
            0
   1.6461e-02
  -3.6996e-02
            0
```



5^a possibilità: approssimazione nel senso dei minimi quadrati

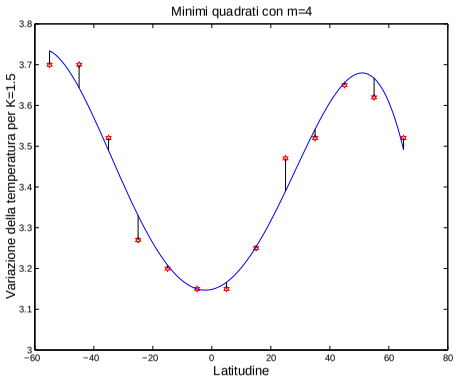
Cerco un polinomio di grado $m < n$ tale da minimizzare la quantità:

$$R(p_m) = \sum_{i=1}^{n+1} (p_m(x_i) - y_i)^2 \quad \forall p_m \in \mathbb{P}_m.$$

R non è altro che il quadrato della norma euclidea (o norma dei quadrati) del vettore residuo `res`. Scegliamo $m = 4$.

```
m=4; c_mq=polyfit(x,y,m);
y5=polyval(c_mq,x1);
figure(5)
plot(x,y,'hr',x1,y5)
roma5=polyval(c_mq,42);
fprintf('Variazione di temperatura a Roma= %6.4e \n',roma5)
yy5=polyval(c_mq,x);
```

```
>> res5=y-yy5
res5 =
  -3.4195e-02
   5.6881e-02
   3.0843e-02
  -6.0094e-02
  -8.1357e-03
   9.4024e-05
  -1.6450e-02
  -3.2326e-03
   7.9861e-02
  -2.1473e-02
  -5.9602e-03
  -4.6745e-02
   2.8607e-02
```



Calcolo R:

```
>> R=norm(res5)^2
R =
   1.9193e-02
```

La function `norm` calcola la norma euclidea di un vettore.

Per il polinomio p_4 ottenuto dall'interpolazione globale di Lagrange su 5 nodi si ha:

```
>> R=norm(res4)^2
R =
   3.8708e-02
```


Caso bidimensionale: 2D

Si estendono i concetti del caso monodimensionale.
1. e 4. *Interpolazione globale di Lagrange:* non c'è una function di Matlab prefatta $\Rightarrow$ bisogna scrivere il codice.

2. *Interpolazione lineare composita di Lagrange:* function `interp2`

3. *Spline cubiche:* function `interp2`

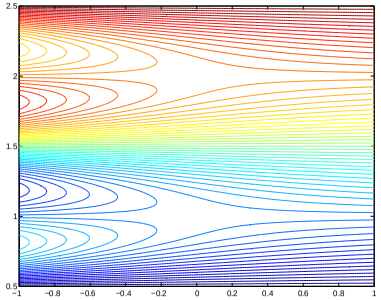
5. *Approssimazione nel senso dei minimi quadrati:* non c'è una function di Matlab prefatta $\Rightarrow$ bisogna scrivere il codice.

$\Rightarrow$ vediamo 2. e 3.

Esempio. La funzione

$$\psi(x,y) = y - \frac{1}{2\pi} e^{2x(10-\sqrt{100+\pi^2})} \cdot \sin(2\pi y),$$
$$(x,y) \in \Omega = (-1,1) \times (0.5, 2.5) \subset \mathbb{R}^2$$

rappresenta le linee di corrente di un flusso laminare oltre una griglia di ostacoli circolari posta perpendicolare alla direzione del moto del fluido.

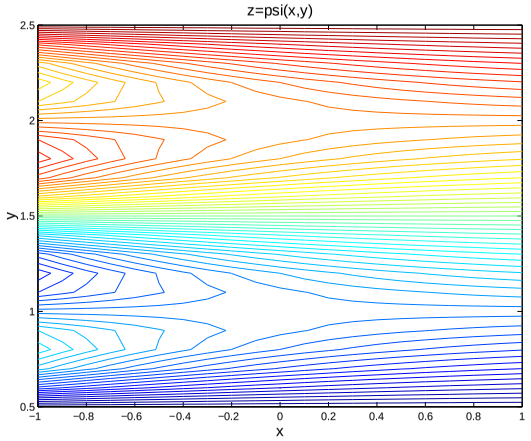


Supponiamo di conoscere la funzione $\psi(x, y)$ su una mesh di punti di Ω equispaziati in entrambe le direzioni x e y, con distanza pari a 0.1. A partire da questi dati vogliamo costruire un interpolazione con funzioni bi-lineari a tratti e con funzioni spline.

Rappresentazione dei dati

1. Costruzione della mesh `[x,y]` in Ω con una spaziatura pari a 0.1
2. Definizione e valutazione della funzione $z = \psi(x, y)$
3. Disegno della $z = \psi(x, y)$ con `mesh`
4. Disegno delle 61 linee di livello della superficie $z = \psi(x, y)$:

```
>> contour(x,y,z,61)
```

Interpolazione bi-lineare a tratti

Il polinomio bi-lineare a tratti è una funzione continua su Ω e ristretta ad ogni rettangolino della mesh è un iperboloide (funzione lineare in entrambe le variabili x e y).

Si definisce una mesh piú fitta su cui valutare la funzione: `[x1,y1]` ottenuta con spaziatura pari a 0.01.

Per valutare e disegnare il polinomio bi-lineare a tratti:

```
>> z1=interp2(x,y,z,x1,y1);
>> figure(2); contour(x1,y1,z1,61)
```

N.B. Non si è ottenuto molto di piú...

Spline bi-cubiche

La spline bi-cubica è una funzione di classe $C^2(\Omega)$ e, ristretta ad ogni rettangolino della mesh, è un polinomio di grado 3 rispetto ad x ed y.

```
>> z2=interp2(x,y,z,x1,y1,'spline');
>> figure(3); contour(x1,y1,z2,61)
```

N.B. Il risultato *qualitativo* è buono: le linee di livello sono rappresentate in maniera regolare.

Per avere una stima *quantitativa* della bontà dell'approssimazione ottenuta si puó calcolare la differenza tra la ψ e la spline, nei nodi della mesh piú fitta `[x1,y1]`.

```
>> x=x1;y=y1; psi1=eval(psi);
>> res2=psi1-z2;figure(3);mesh(x1,y1,res2)
```

L'errore maggiore è concentrato alle estremità dell'intervallo in y. Per ridurre l'errore bisogna aumentare il numero di dati, od usare approssimazioni di grado superiore, o infine, usare mesh non uniformi concentrando i dati là dove serve.

Per calcolare il valore massimo assunto dal residuo:

```
>> err=norm(res2,inf)
err =
   2.7794e-02
```

`norm(v,inf)` calcola la norma "infinito" di un vettore:

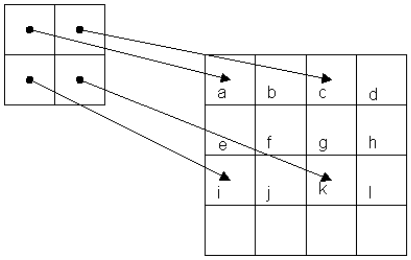
$$\|\mathbf{v}\|_\infty := \max_{i=1,\ldots,n} |v_i|.$$

Interpolazione di immagini

Data un'immagine in formato jpeg di $m \times n$ pixel, la si vuole portare ad una dimensione maggiore (ad esempio $2m \times 2n$) perdendo in definizione il meno possibile.

Ingrandimento con semplice duplicazione dei pixel.

Ingrandimento con semplice duplicazione dei pixel:




```
>> doppia_im
```

Il risultato non è buono.
Alternativa: a=A, c=B, b=ricostruzione del colore a partire dai colori A e B, ad esempio:
b=(A+B)/2; e=(A+C)/2; ecc.

Formato RGB di memorizzazione di una immagine

Una immagine di $m \times n$ pixel è memorizzata in un array `A=(m,n,3)` o equivalentemente in 3 matrici `R, G, B` di dimensione `(m,n)`. R=red; G=green; B=blue.
I valori numerici assunti dagli elementi di queste matrici sono numeri interi in [0,255]. La combinazione dell'intensità dei tre colori base nella posizione `(i,j)` delle tre matrici fornisce il colore finale del pixel `(i,j)`.

Per leggere un'immagine `.jpg` da disco:
`A=imread ('nomefile', 'formato')`

```
>>A=imread('LogoCRS4','jpg');
```

```
>> image(A)

>> whos
  Name          Size          Bytes  Class

  A           75x140x3        31500  uint8 array
```

Salviamo le istruzioni in un m-file.

```
A=imread('LogoCRS4','jpg');
image(A)
[m,n,dim]=size(A);
R=A(:,:,1); G=A(:,:,2); B=A(:,:,3);
% per stabilire un fattore di ingrandimento
fattore=2;
% Dimensioni (in pixel) della nuova matrice
m1=(m-1)*fattore+1;  n1=(n-1)*fattore+1;
```


N.B. `interp2` lavora solo su dati `double`. Bisogna fare una conversione da `uint8` a `double` e poi i risultato deve essere riconvertito in `uint8` per disegnare l'immagine.

Dobbiamo costruire i vettori `x1` e `y1` ed interpolare facendo il passaggio dai dati in formato `uint8` ai dati in formato `double` e viceversa:

```
h=1/fattore;
x1=(1:h:n); y1=(1:h:m)';
R1=uint8(interp2(double(R),x1,y1));
G1=uint8(interp2(double(G),x1,y1));
B1=uint8(interp2(double(B),x1,y1));
A1=uint8(zeros(m1,n1,3));
A1(:,:,1)=R1; A1(:,:,2)=G1; A1(:,:,3)=B1;
figure(2); image(A1)
% per  salvare l'immagine su disco
nome=input('Inserisci nome figura \n','s');
imwrite(A2, [nome,'.jpg'],'jpg')
```


Utilizziamo `interp2` per ingrandire l'immagine con la ricostruzione dei pixel intermedi: o con una interpolazione lineare composita, o con spline cubiche.

Parentesi sull'utilizzo di `interp2`

```
z1=interp2(z,x1,y1);
```

dove `z` è la matrice dei dati da interpolare,
`x1` è un vettore riga contenente i nuovi punti (lungo la `x`) in cui si vuole interpolare,
`y1` è un vettore colonna contenente i nuovi punti (lungo la `y`) in cui si vuole interpolare,
`z1` è la nuova matrice, risultato dell'interpolazione nei nodi di coordinate `(x1,y1)`.

Ingrandimento con interpolazione bi-lineare a tratti.

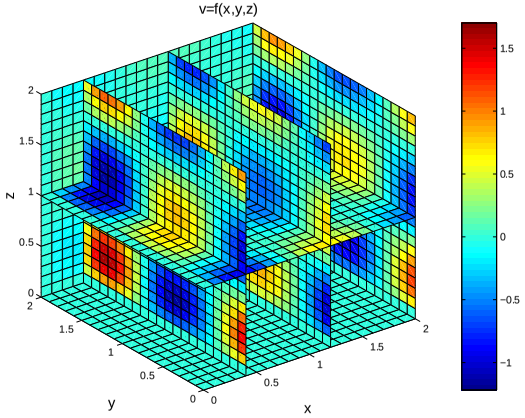
Se si vuole costruire un'interpolazione con spline, si devono sostituire le chiamate ad interp2 con:

```
R1=uint8(interp2(double(R),x1,y1,'spline'));
G1=uint8(interp2(double(G),x1,y1,'spline'));
B1=uint8(interp2(double(B),x1,y1,'spline'));
```


duplicazione


bilineare


spline



Interpolazione 3D

Interpolazione composita lineare e spline cubiche:
1D $\Rightarrow$ 2D $\Rightarrow$ 3D. `interp3`.
Per la visualizzazione grafica: function `slice`.

$f(x,y,z) = \sin(4x)\cos(4y)\sin(4z) \quad$ in $\Omega = (0,2)^3 \subset \mathbb{R}^3$

```
[x,y,z]=meshgrid(0:.5:2);
f='sin(4*x).*cos(4*y).*sin(4*z)'; v=eval(f);
[x1,y1,z1]=meshgrid(0:.1:2);
v1=interp3(x,y,z,v,x1,y1,z1);
figure(1); slice(x1,y1,z1,v1,[0.4 1.2 2],[2],[0 1])
colorbar;
v2=interp3(x,y,z,v,x1,y1,z1,'spline');
figure(2);slice(x1,y1,z1,v2,[0.4 1.2 2],[2],[0 1])
```

Gli ultimi 3 argomenti del comando `slice` sono array che contengono le coordinate dei piani su cui si vuole visualizzare la funzione.

Rappresentazione di dati su griglie strutturate e no

Consideriamo il caso di funzioni con dominio $\Omega \subset \mathbb{R}^2$.

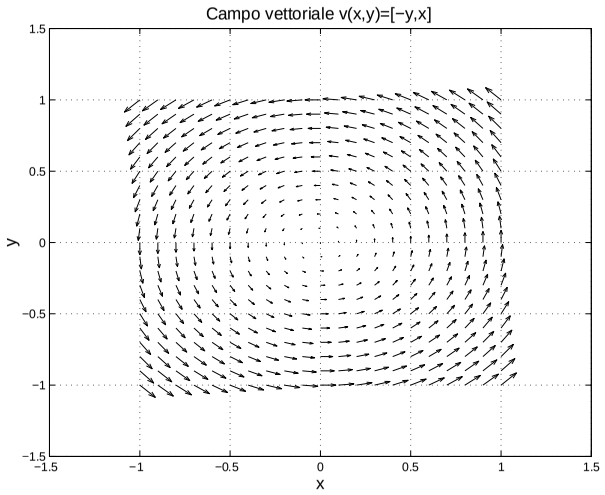
Le *griglie strutturate* sono generate con `meshgrid` ed i comandi di rappresentazione grafica su tali griglie sono:
`surf, mesh, contour, ..., quiver`

```
>> quiver(x,y,v1,v2);
```

disegna un campo vettoriale $\mathbf{v}(x,y) = (v_1(x,y), v_2(x,y))$.

Esempio: $v_1(x,y) = -y,\ v_2(x,y) = x$

```
>> [x,y]=meshgrid(-1:.1:1); v1=-y; v2=x;
>> quiver(x,y,v1,v2); grid
```



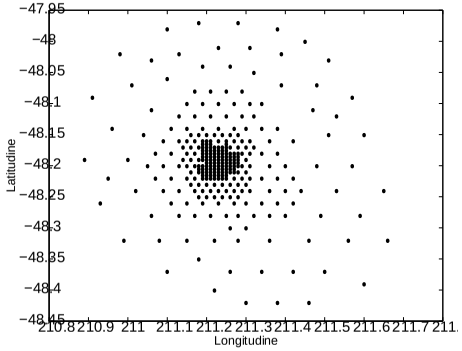
Griglie non strutturate: i nodi NON sono distribuiti su una griglia cartesiana e/o non sono equispaziati.

Vengono cotruiti due vettori `x` e `y` contenenti le coordinate dei punti.

Esempio disegnare la superficie di un monte partendo da misurazioni sperimentali delle altitudini in punti che non corrispondono ad una griglia cartesiana.

```
>> clear
>> load seamount
>> whos
  caption   x         y         z
>> plot(x,y,'.','markersize',12)
>> xlabel('Longitudine'); ylabel('Latitudine')
```


Sono rappresentati in 2D i punti (x, y) in cui sono state fatte le misurazioni.



Non posso visualizzare la superficie $z = z(x, y)$ con `mesh` o altra function vista in precedenza.

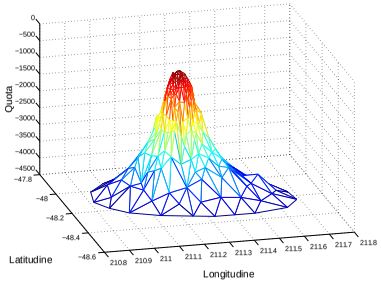
Alternativa: costruire una mesh di triangoli (sempre possibile quando si hanno piú di 3 punti nel piano) e disegnare una superficie partendo da essi.

```
>> tri=delaunay(x,y);
```

`tri` *è una matrice contenente le informazioni per generare i triangoli della mesh*

```
>> trimesh(tri,x,y,z);
```

Disegna la superficie $z(x, y)$ *a partire dalla mesh non strutturata.*



Oppure, generare una mesh strutturata ed interpolare i dati lí sopra.
La function `griddata` costruisce l'interpolatore lineare composito su una griglia strutturata a partire da una griglia non strutturata.

```
>> [x1,y1]=meshgrid(210.8:.01:211.8,-48.5:.01:-47.9);
>> z1=griddata(x,y,z,x1,y1);mesh(x1,y1,z1)
```

oppure, per disegnare le linee di livello con la quota indicata:

```
>> [c,h]=contour(x1,y1,z1); clabel(c,h);
```