Perché calcolo interattivo?

Nei programmi di calcolo interattivo le operazioni da eseguire possono venire immesse direttamente da tastiera ed eseguite istantaneamente.
Rispetto ai programmi prodotti con linguaggio compilato (come C, C++, fortran, etc) si ha il vantaggio di poter rapidamente capire quali sono le operazioni da effettuare sui dati, e poter visualizzare immediatamente il risultato, evitando di dover compilare ed eseguire ogni volta il programma da capo: questo porta a velocizzare le fasi di scrittura dei programmi.
Gli svantaggi risiedono nel fatto che i programmi prodotti risultano piu' lenti di quelli scritti con linguaggio compilato di basso livello, e nel fatto che solitamente non e' possibile produrre programmi in grado di funzionare al di fuori del programma di calcolo col quale sono stati prodotti.
Di solito questi strumenti, e relativi linguaggi, vengono utilizzati nella fase di prototipizzazione di un programma, o nella fase finale di visualizzazione dei risultati, mentre per le operazioni di calcolo su grosse quantità di dati rimane
sempre preferibile l'uso di un programma compilato.

Programmi di calcolo interattivo

Programmi intesi primariamente per il calcolo simbolico
Mathematica *Windows, Mac, Linux*
Maple *Windows, Mac, Linux*

Programmi intesi primariamente per il calcolo numerico:
Matlab - commerciale - Windows, Mac, Linux
Octave - gratuito - *Linux, (windows?)*
Scilab - gratuito - *Linux, Windows*
root - gratuito - *Linux, Mac, Windows*
paw - gratuito - *Linux*
Per Linux si trovano molte altre possibilita', come grace, R, etc..., etc….

Perché Matlab?

…un programma commerciale??????? Bleach!

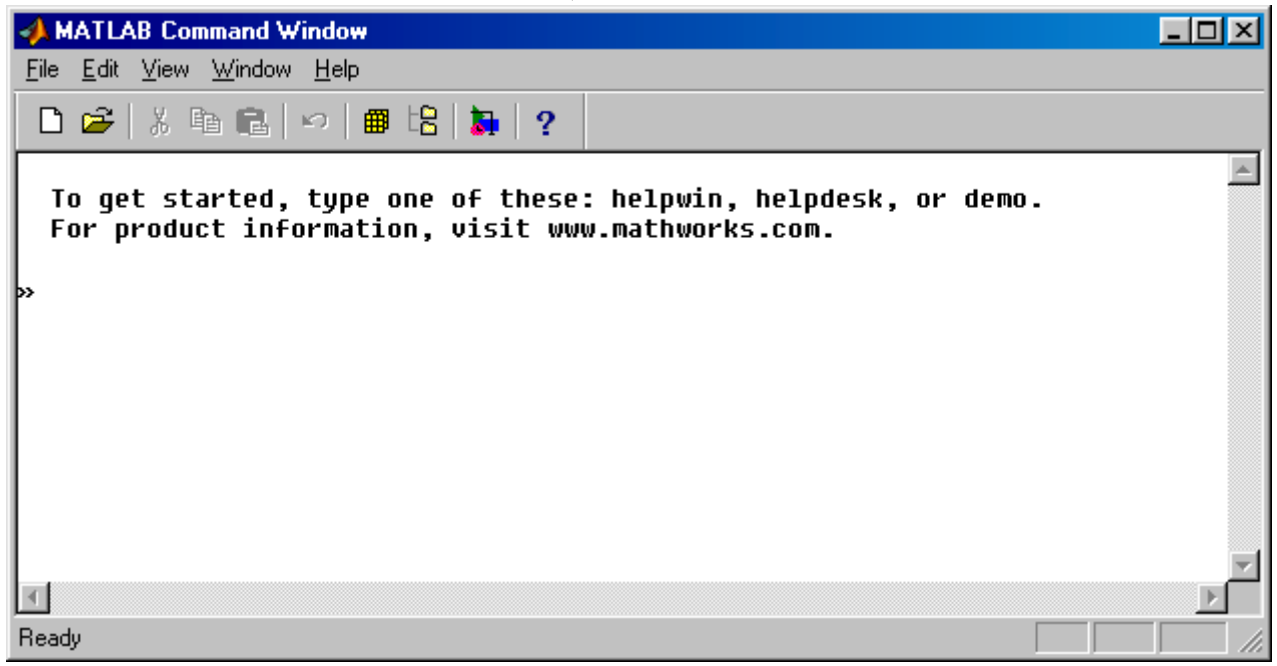
Ma:
- programma più potente
- Molto utilizzato sia nell'ambito scientifico che industriale
- Fornito di molte librerie aggiuntive (toolbox) contenenti funzioni specifiche a diversi tipi di analisi dei dati

Esiste un clone freeware, con la medesima sintassi:
Chi possiede un computer domestico con s.o. Linux può ripetere a casa buona parte delle esercitazioni, e impratichirsi dei comandi fondamentali.

Avviare Matlab

Cliccare sull'icona 

Appare lo schermo:



Primi esempi:

```
»
» 10*12
ans =
   120
»
» sin(.3)
ans =
    0.2955

» log(.2)
ans =
   -1.6094

»
» ans+1
ans =
   -0.6094

»
```

Qui Matlab viene utilizzato come una calcolatrice: lavora direttamente sui numeri

```
» x=15
x =
    15

» y=3.14159
y =
    3.1416

» z=x^2+y^2
z =
  234.8696

» x=15;
» y=3.14159;
» z=x^2+y^2;
» z
z =
  234.8696
```

Qui invece vengono utilizzate le variabili simboliche x,y,z.
Per evitare l'effetto di eco, i comandi possono essere terminati con un punto e virgola ; .

Costanti

Alcune costanti variabili possiedono un valore predefinito: ad esempio pi=3.14159265358979

realmin=2.225073858507201e-308 è il più piccolo numero reale

realmax=1.797693134862316e+308 è il più grande numero reale

eps=2.220446049250313e-016 la minima differenza percentuale tra due numeri

i,I,j,J sono l'unità immaginaria.

Macro

Se si devono eseguire più comandi in una volta sola, è possibile creare una macro: una macro non è altro che un file contenente tutte le istruzioni che si desidera eseguire.
Per creare una macro, bisogna cliccare sul disegno della pagina bianca, il primo a sinistra nella sbarra degli strumenti.
Una nuova macro viene aperta all'interno dell'editore: si possono scrivere i comandi, nell'ordine in cui si desidera eseguirli, e poi salvare.
Per eseguire la macro, bisogna digitare il nome del file al prompt.
Ad esempio, se si dà alla macro il nome mymacro, il file salvato si chiamerà mymacro.m, e per eseguirlo bisognerà digitare
al prompt: mymacro.
Tutte le variabili create o modificate dalla macro saranno disponibili alla fine dell'esecuzione.

Funzioni

La differenza più vistosa tra una macro ed una funzione è costituita dalla sua sintassi: per eseguire una funzione bisogna dare un comando del tipo : **`y=func(x)`**, oppure, in modo più generale,

`[y1 y2 y3 …]=func(x1,x2,x3,…..)`

Le variabili x1, x2, x3 … sono dette variabili di ingresso: sono le uniche variabili sulle quali lavora la funzione. Le variabili y1, y2, y3…. sono dette variabili in uscita: vengono create o modificate dalla funzione, e contengono il risultato delle operazioni effettuate al suo interno. Tutte le altre variabili create dalla funzione vengono distrutte al termine dell'esecuzione.

Una funzione inizia con una riga del tipo:

`function [y1,y2,y3… ]=func(x1,x2,x3,….)`

e termina con l'istruzione:

`return`.

Esempio

```
function [S, p, d]=rettangolo(a,b)
% questa riga, preceduta dal simbolo del percento,
% e' un commento.
S=a*b;
p=2*a+2*b;
d=sqrt(a^2+b^2);
return
```

Questa funzione, dati i lati del triangolo a, b restituisce la superficie S, il perimetro p e la diagonale d.
Per usarla, dopo aver salvato in un file rettangolo.m , devo dare un comando del tipo:

```
[sup, per, diag]=rettangolo(A,B);
```

Si noti come i nomi utilizzati per le variabili possono anche essere diversi da quelli usati internamente
dalla funzione. Quello che importa e' solamente l'ordine.

Ciclo for

Un ciclo for è una serie di istruzioni del tipo:

for k=1:100
  operazioni….
end
Il principio è semplice: dapprima alla variabile k viene dato il valore 1, poi vengono eseguite le istruzioni, quindi a k
viene dato il valore 2, e le operazioni vengono di nuovo eseguite, così via fino ad arrivare ad 100.

Esempio:

```
function fatt=fattoriale(n)
fatt=1;
for k=1:n
   fatt=fatt*k;
end
return
```

Questa funzione calcola n!

Struttura if….else

```
if (condizione1)
istruzione 1
elseif (condizione2)
istruzione 2
else
istruzione 3
end
```

In questo schema dapprima viene verificata la condizione 1: in caso affermativo, l'istruzione 1 viene eseguita.
Se la condizione 1 non e' verificata, si passa al controllo della condizione 2: se questa è verificata si esegue l'istruzione 2.
Se nessuna delle due condizioni e' verificata, e solo in questo caso, si esegue l'istruzione 3.
Esempio: per evitare di calcolare i logaritmi di zero, o di un numero negativo

```
if (x>0)
   lx=log(x);
else
   lx=NaN;
%NaN= not a number….
end
```

Condizioni

`x>a` x maggiore di a
`x<a` x minore di a
`x>=a` x maggiore o uguale ad a
`x<=a` x minore o uguale ad a
`x==a` x uguale ad a
`x~=a` x diverso da a
L'espressione **`if (x)`** controlla se x è diverso da zero.

& and logico: **`(x>10)&(y>100)`**
| or logico : **`(x>10)|(y>100)`**
~ negazione: ~(x==a) corrisponde a (x~=a)

Ottenere aiuto

Se si conosce il nome della funzione, e si vogliono conoscere dettaglie sul suo funzionamento, esiste il comando help:

```
» help round

 ROUND  Round towards nearest integer.
    ROUND(X) rounds the elements of X to the nearest integers.

    See also FLOOR, CEIL, FIX.
```

Un comando utile per trovare una funzione sconosciuta è lookfor

Altrimenti, andando nel menù help->help window compare una finestra in cui tutti i comandi e le funzioni sono raggruppati per categorie, con una breve descrizione. Infine si può scegliere help desk: questo comando aprirà il browser con la documentazione completa di matlab: questa è la più esauriente, ma anche la più dispersiva.

Esercizi

Verranno proposti quattro esercizi:
1) calcolo del campo e del potenziale di una carica: si puo'
fare da linea di comando, senza creare nuovi files
2) modifica di una funzione di euroconvertitore: per impratichirsi della costruzione if-then
3) creazione di una funzione per il calcolo delle coordinate polari: per imparare a creare una funzione dal nulla
4) Calcolo della somma troncata di una serie: per imparare il ciclo for.

Vettori

```
» x=[1 4 8 16]
x =
    1     4     8
16

» y=[1; 4; 8; 16]
y =
     1
     4
     8
    16

» x(3)

ans =

     8

»
```

Posso creare un vettore riga oppure un vettore colonna, separando gli indici da un ;
Le singole componenti del vettore possono essere ottenute ponendo l'indice tra parentesi tonde:
x(3) è la terza componente del vettore x

```
» z=3*x
z =
     3    12    24    48

» z=x+5
z =
     6     9    13    21
```

Moltiplicare un vettore per un numero equivale a moltiplicare tutti gli elementi per lo stesso numero. Analogamente, aggiungere un numero vuol dire aggiungere lo stesso valore a tutte le componenti. Lo stesso si ottiene moltiplicando per o aggiungendo una variabile numerica.

Creazione ed indicizzazione

```
» x=[1:10]
x =
    1    2    3    4    5    6    7    8    9   10
» y=[0:.3:2.]
y =
         0    0.3000    0.6000    0.9000    1.2000    1.5000    1.8000
»
» y(3:5)
ans =
    0.6000    0.9000    1.2000

» x=linspace(0,13,10)
x =
  Columns 1 through 7
         0    1.4444    2.8889    4.3333    5.7778    7.2222    8.6667
  Columns 8 through 10
   10.1111   11.5556   13.0000

» pp=[1 4 5 ];
» y(pp)
ans =
         0    0.9000    1.2000»
```

Crea un vettore con gli interi da 1 a dieci

Crea un vettore partendo dal primo valore ed incrementando del secondo fino a giungere al terzo.

Crea un vettore di 10 elementi equispaziati, con primo elemento 0 e ultimo elemento 13.

Indicizzazione di vettori: posso selezionare solamente un sottoinsieme del vettore.

Operazioni tra vettori

Queste operazioni sono una caratteristica del linguaggio usato da matlab: solitamente i linguaggi c, c++, fortran utilizzano una notazione più involuta.ma che si presta a minori equivoci

```
» x=[1 2 3 4 5 ];
» y=[2 4 6 8 10];
» z=x+y

z =

    3     6     9    12    15

» y=[2 4 6 8 10 12];
» z=x+y
??? Error using ==> +
Matrix dimensions must agree.

»
» z=sin(x)
z =
   0.8415    0.9093    0.1411   -0.7568   -0.9589
```

Posso sommare o sottrarre due vettori riga o colonna della stessa dimensione: matlab somma elemento per elemento. Posso calcolare espressioni come sin(x), exp(x), etc: matlab applica l'espressione aritmetica ad ogni singolo elemento

Prodotto tra vettori

Posso effettuare il prodotto di due vettori elemento per elemento se questi hanno la stessa dimensione: si indica col simbolo .*

```
» x=[1 2 3 4 5 ];
» y=[2 4 6 8 10];
»
» x.*y
ans =
    2     8    18    32
50
```

Se x e y hanno la stessa lunghezza, ma sono un vettore riga ed uno colonna, posso effettuare una ordinaria moltiplicazione riga * colonna utilizzando il simbolo *

```
» x=[1 2 3 4 5 ];
» y=[2; 4; 6; 8; 10];
» x*y
ans =
  110
```

Posso anche effettuare il "prodotto esterno": in questo caso ottengo una matrice: nella pratica, basta porre prima il vettore colonna e poi il vettore riga.

$$M_{ij} = y_i x_j$$

```
» y*x
ans =
    2     4     6     8    10
    4     8    12    16    20
    6    12    18    24    30
    8    16    24    32    40
   10    20    30    40    50
```

Altre operazioni

```
» x=[0 1 -2 4 5 13 -10];
» length(x)
ans =
     7
 » [m i]=max(x)
m =
    13
i =
     6
» [m i]=min(x)
m =
   -10
i =
     7
» pt=find(x<0)
pt =
     3     7
»
```

La funzione length ritorna la lunghezza del vettore.

Le funzioni max e min restituiscono il valore massimoe minimo e l'indice a cui questi valori corrispondono.

La funzione find(condizione) restituisce un vettore contenente gli indici per cui la condizione è verificata….

Statistica

```
» x=[0 1 -2 4 5 13 -10];
» mean(x)
ans =
   1.5714
» std(x)
ans =
   7.0441
» sum(x)
ans =
   11
» prod(x)
ans =
    0
» median(x)
ans =
    1
»
```

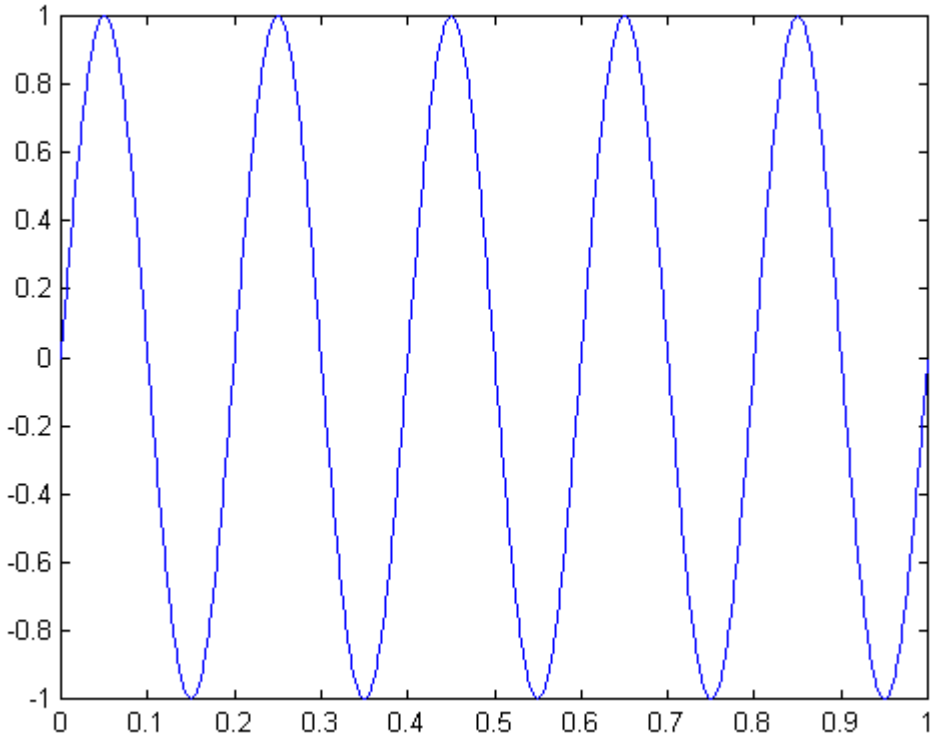
Le funzioni mean, std, cov, median calcolano la media, la deviazione standard, la varianza e la mediana dei dati contenuti in x.
Le funzioni sum(x) e prod(x) calcolano la somma e il prodotto degli elementi.

Ad esempio: sqrt(sum(x.^2)) calcola il modulo del vettore x
prod([1:n]) calcola il fattoriale di n.

Plot

```
» t=[0:.001:1];
»
» x=sin(5*t*2*pi);
» plot(t,x)
»
```

Questi comandi creano un vettore t tra 0 ed 1 secondo, a passi di 1 ms, poi un vettore contenente una sinusoide di frequenza 5 Hz, ed infine creano un grafico con t in ordinata e x in ascissa.



Altri esempi….

Plot(t,x,'.') stampa una serie di punti

plot(t,x,'r') stampa una linea rossa (g, verde, m magenta, y yellow, k nero, etc…)

plot(t,x,,t,y,'r') stampa due grafici nella stessa figura, x e y in funzione di t.

plot3(x,y,z) grafico tridimensionale di x,y,z

semilogx(t,x) Usa una scala logartimica nelle ascisse

semilogy(t,x) Usa una scala logaritmica nelle ordinate

loglog(t,x) Usa una scala bilogaritmica

Matrici

Un vettore con due indici costituisce una matrice ad esempio:

```
» m=[1 2 3 4; 3 5 9 11; 4 8 12 16]
m =
     1     2     3     4
     3     5     8    11
     4     8    12    16

»
» m(2,3)
ans =
     9
» m'
ans =
     1     3     4
     2     5     8
     3     8    12
     4    11    16
```

Crea una matrice 3 x 4

L'elemento 2, 3 della matrice (seconda riga, terza colonna)

L'operatore apice ' permette di trasporre una matrice.

I vettori riga non sono altro che matrici con la prima dimensione uguale ad uno, e cosi' i vettori colonna sono matrici con la seconda dimensione uguale ad 1. Le matrici di uguale dimensione si possono sommare, sottrarre, moltiplicare per un numero o sommare ad un numero con le stesse regole dei vettori.

Indicizzazione

```
» m=[1 2 3 4; 3 5 9 11; 4 8 12 16]
m =
    1    2    3    4
    3    5    9   11
    4    8   12   16
» m(2,:)
ans =
    3    5    9   11
» m(:,2)
ans =
    2
    5
    8
» [i j]=find(m>10)
i =
    3
    2
    3
j =
    3
    4
    4
»
```

Il primo indice indica la riga, il secondo la colonna.

Si può estrarre una singola riga o una singola colonna utilizzando i due punti :

La funzione find restituisce due vettori di indici, uno per le righe e l'altro per le colonne.

Operazioni con matrici

```
» a=[1 2 3 ; 2 3 4]
a =
     1     2     3
     2     3     4

» b = [1 2 ; 3 4 ; 5 6 ]
b =
     1     2
     3     4
     5     6

» a*b
ans =
    22    28
    31    40

» b*a
ans =
   5   8  11
  11  18  25
  17  28  39
```

Posso moltiplicare una matrice (*m x n*) con una (*n x l*) Il risultato sarà una matrice (*m x l*)

Posso moltiplicare una matrice (*m x n*) per un vettore colonna di lunghezza *n*: il risultato sara' un vettore colonna di lunghezza *m*

Posso moltiplicare un vettore riga di lunghezza *n* per una matrice di dimensioni (*n X m*): il risultato sarà un vettore riga di lunghezza *m*.

L'operatore .* (punto seguito da asterisco) permette di effettuare la moltiplicazione elemento per elemento di matrici di uguali dimensioni. Allo stesso modo funzionano ./ e .^

M^2 indica invece il quadrato di una matrice effettuato moltiplicando riga per colonna.

Operazioni su matrici

```
» a=[1 2 3; 1 4 9; 1 8 27]
a =
     1     2     3
     1     4     9
     1     8    27

» rank(a)
ans =
     3

» det(a)
ans =
    12

» inv(a)
ans =
    3.0000   -2.5000    0.5000
   -1.5000    2.0000   -0.5000
    0.3333   -0.5000    0.1667
```

Data una matrice a, rank(a) da' il rango della matrice, nei limiti delle approssimazioni numeriche

Per matrici quadrate:
det(a) da' il determinante.
trace(a) da' la traccia
inv(a) da' l'inverso

Metodi Montecarlo

I metodi Montecarlo permettono di risolvere problemi mediante l'uso di numeri casuali, o meglio pseudocasuali.
Una sequenza di numeri pseudo casuali è una sequenza generata da un processo del tipo:

$$x_n = f(x_{n-1})$$

Nonostante la sequenza sia deterministica e periodica, i numeri che compongono la sequenza appaiono totalmente scorrelati tra di loro, e risultano indistinguibili anche mediante test raffinati da una autentica sequenza di numeri casuali.
La funzione f è in realtà un algoritmo complicato e fortemente non lineare.
La maggior parte dei generatori di numeri pseudocasuali permette di estrarre numeri compresi tra 0 ed 1 con distribuzione uniforme. Tramite semplici algoritmi è possibile ottenere una sequenza di numeri pseudocasuali con distribuzione di probabilità qualsiasi.
Es: se x è distribuita in modo uniforme tra 0 e 1, $y=x^2$ sarà distribuita secondo la funzione $1/y^{1/2}$

Generatori di Matlab

```
» x=rand([1 7])
x =
   0.1038    0.6960    0.7177    0.2568    0.4222    0.8566    0.0687
```

La funzione rand(v) genera una matrice di numeri casuali di dimensioni v(1), v(2) distribuiti uniformemente tra 0 e 1.

```
» x=randn([1 7])
x =
  -0.0376    0.3273    0.1746   -0.1867    0.7258   -0.5883    2.1832
» y=3+2.*x
y =
   2.9247    3.6546    3.3493    2.6266    4.4516    1.8234    7.3664
»
```

La funzione randn genera numeri distribuiti secondo una gaussiana di media zero e larghezza 1.
Per ottenere numeri distribuiti in modo normale con media e larghezza qualunque, basta moltiplicare per la larghezza desiderata ed aggiungere la media.
Nell'esempio, y è distribuita con larghezza 2 e media 3.

Primo Esercizio

L'esercizio consiste nella realizzazione e nel test di un programma per effettuare il fit del minimo chi-quadro.

La funzione **`crea_dati`** e' un semplice programma di tipo Montecarlo, che crea una serie di dati legati da una relazione lineare cui viene aggiunto un errore di misura simulato la cui entità percentuale può essere scelta a piacere.

Il file **`fit_retta`** contiene gia' un programma che permette il fit dei dati con il metodo dei minimi quadrati.

Il programma **`test_fit`** permette un test del fit effettuato.

Compito dell'esercizio sara' di utilizzare i programmi dati, verificare il fit, e modificare il programma di fit per trasformarlo in un fit del minimo chi2.

Il programma Montecarlo

Nel caso proposto il programma crea_dati crea un vettore x di N elementi equispaziati tra xmin ed xmax, calcola la nuova quantita' y come y=ax+b, e poi aggiunge alla quantita' y un errore di misura casuale. L'errore di misura viene supposto distribuito secondo una gaussiana di larghezza err percento del valore y calcolato in precedenza.
In uscita si ottengono oltre ai vettori x e y il vettore contenente la larghezza della gaussiana:

```
[x y s ]= crea_dati(xmin,xmax,N,a,b,err);
```

Si puo' verificare immediatamente il risultato effettuando un plot.
Ad esempio:

```
[x y s]=crea_dati(0,1,1000,7,3,10);
plot(x,y,'.');
```

Listato della funzione crea_dati

```
function [x, y, s]= crea_dati(xmin,xmax,N,a,b,err);
% creo il vettore delle ascisse
x=linspace(xmin,xmax,N);
% questa riga crea un vettore della stessa dimensione di x formato da
% numeri distribuiti
% secondo una gaussiana di media 0 e larghezza 1.
r=randn(size(x));
% calcola il vettore y
y=a*x+b;
% calcola la precisione con cui viene effettuata ogni misura
s=err*y/100;
% aggiunge ad y un errore di misura proporzionale alla larghezza
y=y+s.*r;
return
```

Formule per i minimi quadrati

La funzione da minimizzare e'

$$q = \sum_{i=1}^{N} \left(y_i - a x_i - b \right)^2$$

Derivando si ottiene:

$$\sum_{i=1}^{N} x_i y_i - a \sum_{i=1}^{N} x_i^2 - b \sum_{i=1}^{N} x_i = 0$$

$$\sum_{i=1}^{N} x_i - a \sum_{i=1}^{N} x_i - bN = 0$$

In forma matriciale:

$$MA = V \qquad A = \begin{pmatrix} a \\ b \end{pmatrix} \quad M = \begin{pmatrix} \sum x^2 & \sum x \\ \sum x & N \end{pmatrix} \quad V = \begin{pmatrix} \sum xy \\ \sum y \end{pmatrix}$$

Con soluzione formale:

$$A = M^{-1} V$$

Listato del programma di fit

```
function [a, b]=fit_retta(x,y,sigmay);
% effettua il fit di una retta col metodo dei minimi quadrati
% y= ax + b
% in questa versione non sfrutta l'informazione data da sigmay

% pongo a e b uguale a zero....
a=0;
b=0;
% dapprima controlliamo che x e y abbiano la stessa lunghezza.....
% altrimenti pone a e b uguali a zero.
if (length(x)~=length(y))
   return
end
…….. Continua…….. ->
```

Fase di inizializzazione…….

```
%calcola la somma di x
sx=sum(x);
% calcola la somma di y
sy=sum(y);
% calcola la somma di x^2
sx2=sum(x.*x);
sxy=sum(x.*y);
N=length(x);
% crea la matrice
M=[sx2 sx ; sx N];
% crea il vettore
v=[sxy ; sy];
% controlla il rango di M
if (rank(M)<2)
      return
end

   Minv=inv(M);
   soluzione=Minv*v;
   a=soluzione(1);
   b=soluzione(2);
return
```

Crea la matrice, M e il vettore V,
e moltiplica V per l'inverso di M.
Il rango di M deve essere uguale a 2.

Formule per il minimo χ^2

La funzione da minimizzare e'

$$\chi^2 = \sum_{i=1}^{N} \frac{(y_i - a x_i - b)^2}{\sigma_i^2}$$

Derivando si ottiene:

$$\sum_{i=1}^{N} \frac{x_i y_i}{\sigma_i^2} - a \sum_{i=1}^{N} \frac{x_i^2}{\sigma_i^2} - b \sum_{i=1}^{N} \frac{x_i}{\sigma_i^2} = 0$$

$$\sum_{i=1}^{N} \frac{x_i}{\sigma_i^2} - a \sum_{i=1}^{N} \frac{x_i}{\sigma_i^2} - b \sum_{i=1}^{N} \frac{1}{\sigma_i^2} = 0$$

In forma matriciale:

$$MA = V \qquad A = \begin{pmatrix} a \\ b \end{pmatrix} \quad M = \begin{pmatrix} \sum \frac{x^2}{\sigma^2} & \sum \frac{x}{\sigma^2} \\ \sum \frac{x}{\sigma^2} & \sum \frac{1}{\sigma^2} \end{pmatrix} \quad V = \begin{pmatrix} \sum \frac{xy}{\sigma^2} \\ \sum \frac{y}{\sigma^2} \end{pmatrix}$$

Con soluzione formale:

$$A = M^{-1} V$$

La funzione di test.

Per effettuare il test, si può calcolare il chi2 delle varie misure:

```
function chi2=chi2(x,y,s,a,b)
% questo vettore contiene i valori di y che mi aspetterei in base al
fit.
yfit=a*x+b;
% questo valore contiene la differenza tra y e yfit
deltay=y-yfit;
% calcolo il chi2 effettuando il quadrato dei singoli elementi di
%deltay e di s e dividendo
% elemento per elemento, e poi sommando il vettore ottenuto.
chi2=sum((deltay.^2)./(s.^2));
return
```

Secondo esercizio

Si consideri un esempio semplice: un cannone con alzo θ che spara proiettili con velocità v_0 ha una gittata data dalla formula:

$$d = \frac{v_0^2 sin(2\theta)}{g}$$

Supponiamo che la velocità del proiettile sia in realtà distribuita con larghezza nota intorno al valore v0, e che anche l'alzo sia indeterminato di un certo valore: in questo caso il punto di impatto risulterà pure indeterminato intorno alla posizione teorica.
Il calcolo dell'indeterminatezza è facile nel caso di angoli diversi da 45 gradi: per quel valore infatti la gittata ha un massimo, e le solite approssimazioni lineari non sono più valide.
Per calcolare la distribuzione del punto di impatto si può allora provare ad utilizzare il metodo Montecarlo.

Ci porremo il problema di riuscire a calcolare il punto di impatto medio di un proiettile sparato da un cannone con alzo pari a 45 ± 5 gradi con velocità v=20 ±0.2 m/s,
e la percentuale di colpi che cadono entro un metro dal punto di impatto medio.

Procedura

1) si genera un vettore contente un insieme pseudocasuale di angoli distribuiti secondo una gaussiana di media 45 e larghezza 5, da convertire successivamente in radianti

2) Si genera un vettore contenenti velocità casuali con media 20 e larghezza 0.2

3) si calcola il vettore delle gittate corrispondenti a ciascun lancio

4) si calcola la media e la deviazione standard delle gittate e la si confronta col valore teorico

5) si esamina la distribuzione delle gittate tramite il comando hist(x,n) che genera a partire dal vettore x un istogramma con n canali

6) si determina, tramite il comando find, il numero di proiettili che cadono entro un metro dal punto di impatto teorico.