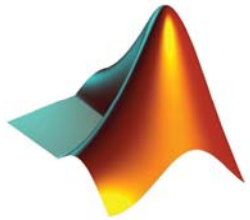
L'ambiente Matlab
per le applicazioni industriali
(Parte 1 – Panoramica)



Schema del corso

Ogni modulo è strutturato in modo indipendente e copre aspetti progressivamente più avanzati

Moduli:

1. Introduzione a Matlab come linguaggio
2. Grafica 2D e 3D
3. Applicazioni di calcolo con Matlab
4. Panoramica dei toolbox e loro impiego
5. Costruzione di una GUI in Matlab
6. Uso dei MEX files e del compilatore Matlab
7. Introduzione a Simulink
8. Risorse Matlab di pubblico dominio

Un pò di storia

- **Acronimo di Matrix Laboratory**
- **Creato alla fine degli anni '70 alla New Mexico University da Chris Molen (ed altri)**
- **Lo scopo iniziale era fornire un ausilio ai corsi di algebra lineare e di calcolo numerico per studenti senza conoscenze di programmazione**
- **Integra gradatamente librerie già disponibili (es. Linpack)**
- **Nel 1984 inizia la commercializzazione da parte di Mathworks**
- **Evolve successivamente come suite completa di supporto alla ricerca scientifica ed allo sviluppo di applicazioni ad elevato contenuto scientifico**

Un pò di storia

- **Oggi ha diffusione universale ed è maggiormente orientato alle applicazioni industriali mediante la realizzazione di parecchi toolbox specifici**
- **Esistono diverse soluzioni di acquisto parziale (a nessuno serve tutto…) che consentono di accedere alle sole parti utili**
- **Le prossima versione (in beta testing) sarà definitivamente object-oriented**
- **L'impiego didattico rimane attuale ma molto spesso è limitato al solo prodotto base (il motore di calcolo) usato in modalità interpretata**

Le versioni successive

Anno	Nome	Versione
1986	-	2
1987	-	3
1992	R7	4
1996	R8	5
2000	R12	6
2004	R14	7

Perché Matlab?

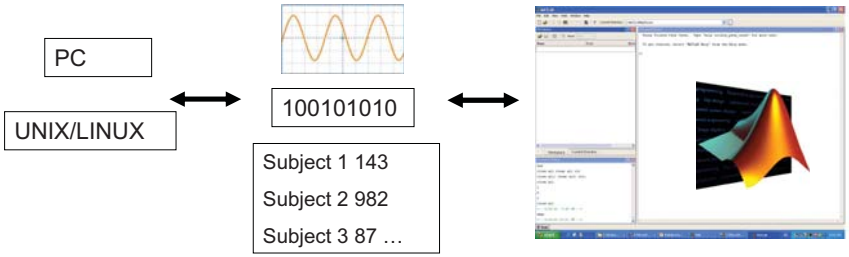
Acquisizione dati

- MATLAB fornisce primitive per acquisire ed analizzare dati da qualsiasi sorgente digitale

Perché Matlab?

Importazione dati

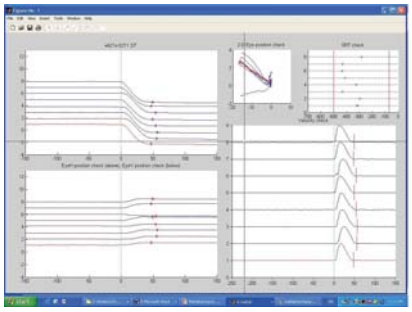
- Si possono importare in MATLAB dati in qualunque formato e da qualunque piattaforma



Perché Matlab?

Strumenti di analisi

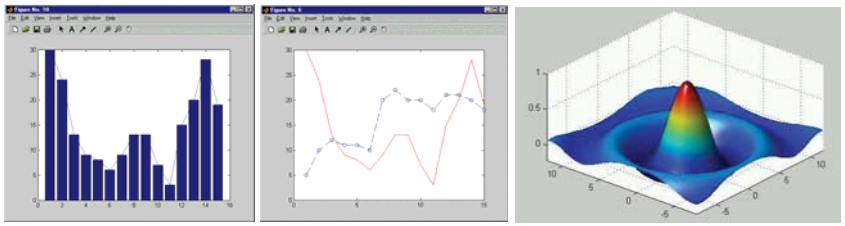
- Esistono librerie per compiere qualunque tipo di analisi sui dati
- E' possibile costruire qualunque tipo di rappresentazione grafica



Perchè Matlab?

Grafica N-dimensionale

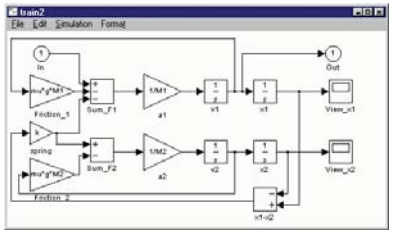
- Sono disponibili tutte le opzioni di grafica da 2 a 4 dimensioni
- Possibile controllo totale della formattazione e di qualunque elemento visivo

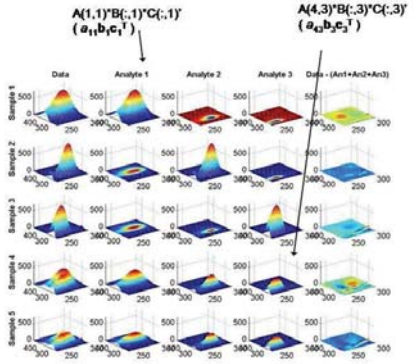


Perchè Matlab?

Modellazione dinamica

- E' possibile costruire modelli dell'interazione di sistemi dinamici complessi e verificarli su dati sperimentali





Perchè Matlab?

- **Disponibilità di sw**
  - **Matlab contiene un numero enorme di funzioni pronte per l'uso, organizzate in toolbox tematici e perfettamente commentate**
  - **Il sito di Mathworks è ricchissimo di esempi ed applicazioni pronte per l'uso**
  - **Sono disponibili in rete risorse di pubblico dominio che estendono ulteriormente questo insieme a specifici problemi applicativi**
  - **Le probabilità di trovarsi il lavoro già fatto sono parecchio elevate…**

Alcuni fatti …

- **Matlab serve solo ai matematici**
  - FALSO! anzi, è vero il contrario. Serve poco (o nulla) ai matematici, è invece pensato e sviluppato per la progettazione di sistemi, quindi per l'ingegneria
- **Matlab è lento**
  - FALSO! è lento solo se usato in modalità interpretata. Il compilatore genera eseguibili di discreta efficienza, specie se si usano bene le funzioni già ottimizzate.
- **Matlab costa troppo**
  - FALSO! Sì, il costo iniziale è oggi il vero limite alla diffusione di Matlab nell'industria. Solo le grandi aziende usano Matlab originale, le altre…si arrangiano. Ma rispetto ai costi umani di sviluppo sw, Matlab costa poco.

Alcune critiche …

- Non esistono versioni con le sole libreria di funzioni compilate (forse con .NET…)
- L'uso di (…) per compiti diversi è criticabile
- I nomi di funzione (parecchi…) sono tutti globali, con possibili conflitti risolti solo in base all'ordine
- Il comportamento variabile delle funzioni spesso confonde i programmatori (es. sum)
- Gli array che partono da 1 sono fonte infinita di errori quando si interfaccia Matlab con altri linguaggi
- Le diverse versioni di Matlab creano problemi di porting
- …

Uso di Matlab

Valutare espressioni utilizzando variabili

- Le espressioni immesse sono interpretate e valutate immediatamente dal sistema
- Le variabili sono identificativi usati per memorizzare valori
- Le variabili calcolate possono essere richiamate in seguito
- Le variabili non vanno dichiarate (dichiarazione = assegnazione)

Variabile = *Espressione*

Oppure

Espressione

N.B. i nomi delle variabili sono case-sentitive

Uso di Matlab

Lavorare con le matrici

- Matlab opera normalmente con un unico oggetto di base, una matrice rettangolare di numeri (array) indicizzata con (…,…)
- Una matrice è una struttura indicizzata da 2 valori: riga e colonna
- Il numero di righe o colonne è arbitrario
- Gli indici partono sempre da 1

Uno scalare è un singolo numero rappresentato da matlab come una matrice 1x1.

Un vettore è un array mono-dimensionale di numeri rappresentato come matrice n x 1 (vettore colonna) o 1 x n (vettore riga)

Si usano "," e ";" per separare gli elementi nella assegnazione

Operazioni con matrici

Una matrice vuota può essere creata con [ ]

L'operatore [ ] può essere usato per cancellare righe o colonne

Le funzioni size e ndims ritornano il numero di elementi e di dimensioni di una variabile matriciale

La struttura 2D è generalizzabile ad un numero arbitrario di dimensioni (matrici N-dimensionali)

L'operatore ':' consente di estrarre sottomatrici mediante la definizione di intervalli di righe o colonne

Operazioni con matrici

Tutte le normali operazioni di calcolo sono utilizzabili sulle matrici combinate con scalari

- **+ Addizione**
- **- Sottrazione**
- *** Moltiplicazione**
- **/ Divisione**
- **^ Elevamento a potenza**

Le operazioni con scalari sono sempre intese come applicate ai singoli elementi della matrice

Operazioni con matrici

Le operazioni tra matrici assumono particolari proprietà

Addizione e Sottrazione

Le dimensioni delle matrici devono essere uguali

Moltiplicazione

Ci sono due possibili implementazioni

*prodotto algebrico (righe x colonne) – operatore **

*prodotto elemento x elemento – operatore .**

I vincoli sono diversi nei due casi

Operazioni con matrici

Divisione di matrici

Ci sono due possibili implementazioni:

*divisione matriciale a sinistra o a destra – operatori / e *

divisione elemento per elemento – operatore ./

Divisione a sinistra (\) :

X = A\B è la soluzione di A*X = B

Divisione a destra (/) :

X = A/B è la soluzione di X*A = B

Potenza di una matrice

Ci sono due possibili implementazioni:

Serie di prodotti

Potenza elemento per elemento

Operazioni con matrici

Operazioni specifiche per matrici

La trasposta si forma scambiando tra loro le righe con le colonne –operatore ‘

inv	**inversa di una matrice**
det	**determinante di una matrice**
trace	**traccia di una matrice**
rank	**rango di una matrice**
zeros	**matrice nulla**
ones	**matrice unaria**
diag	**matrice diagonale**
eye	**matrice identità**

Esistono numerosissime altre operazioni specifiche per le matrici

Operazioni con matrici

- **min / max**
  - minimo / massimo lungo righe /colonne
- **sum / mean**
  - somma / media lungo righe / colonne
- **norm**
  - norma di una matrice
- **sort**
  - ordina le righe / colonne
- **reshape**
  - Altera la forma di una matrice ridistribuendo i valori

Operatori relazionali

Utilizzati per comparare matrici di uguale dimensione (o scalari)

<	**minore**
<=	**minore o uguale**
>	**maggiore**
>=	**maggiore o uguale**
==	**uguale**
~=	**non uguale**

La comparazione, a risultato TRUE o FALSE, avviene sempre tra elementi corrispondenti

Operatori relazionali

La funzione Find

La funzione *find* serve per trovare le posizioni di tutti gli elementi di una matrice che soddisfano una determinata condizione

```
>> find(A > 0)
```

Le posizioni trovate possono essere usate per indicizzare matrici

```
>> A(find(A>0)) = 1
```

La funzione ritorna anche le posizioni come vettori o come matrice

```
[x,y] = find(A > 0)        B = find(A > 0)
```

Gli scripts di Matlab

Raggruppano una serie di comandi senza eseguirli (modalità differita)

- **Matlab può eseguire sequenze di comandi contenuti in un file**
- **I file che contengono comandi Matlab devono avere estensione '*.m'**
- **Gli M-files si possono scrivere e salvare con l'apposito editor**
- **Gli M-files sono eseguibili al command prompt come un comando**
- **Gli M-files possono chiamare altri M-files**

N.B. Per essere eseguibile, un M-file deve avere il suo path settato nella configurazione di Matlab

Gli scripts di Matlab

Vantaggi degli M-files

- **Sviluppo facilitato dall'editor**
- **Possibili modifiche a valle dell'esecuzione**
- **Leggibilità/Portabilità – si possono aggiungere commenti tramite il simbolo '%' per facilitare la comprensione**
- **Salvare M-files è più efficiente che salvare il workspace**

Operatori Logici

- **Gli operatori logici sono usati per comparare due valori booleani**
- **Un valore booleano è una rappresentazione logica di vero o falso**
- **Convenzionalmente 1 e 0 sono usati per rappresentare vero e falso**
- **In Matlab quando si fa una comparazione booleana, il valore '*false*' o 0 rappresenta falso e ogni intero positivo o 'true' rappresenta vero**

Operatori Logici

&	**AND**
|	**OR**
~	**NOT**

- **L'operatore & (AND) ritorna TRUE solo se entrambi i valori A,B sono TRUE altrimenti ritorna FALSE**
- **L'operatore | (OR) ritorna TRUE se almeno uno dei valori A, B è TRUE, altrimenti ritorna FALSE**
- **L'operatore ~ (NOT) inverte la rappresentazione booleana**

Ordine di precedenza:

Nelle operazioni logiche multiple senza parentesi, l'operatore ~ (NOT) viene valutato per primo, seguito da & (AND) e da | (OR)

A&B|C = (A&B) | C **A|B&C = A | (B&C)**

Controllo di Flusso

- **Il controllo di flusso consente a Matlab di superare la semplice funzionalità di puro calcolo**
- **Con il controllo di flusso Matlab può essere usato come un linguaggio di programmazione ad alto livello orientato alla elaborazione di matrici**
- **Il controllo di flusso è implementato tramite statement condizionali e cicli**

Statement condizionali

If, Else, and Elseif

- Uno statement if valuta una espressione logica ed esegue un gruppo di comandi se questa è vera
- La list dei comandi condizionati termina con uno statement end
- Se l'espressione logica è falsa, tutti I comandi condizionati sono saltati
- L'esecuzione dello script riprende dopo lo statement end

```
if espressione_logica
    comandi
end
```

Statement condizionali

If, Else, and Elseif

- Lo statement else forza l'esecuzione dei comandi successivi se l'espressione logica originale è falsa
- Solo una delle due liste viene eseguita

```
if espressione_logica
    comandi 1
else
    comandi 2
end
```

Statement condizionali

If, Else, and Elseif

- Lo statement elseif nidifica una nuova struttura if dopo un else
- Sono una delle liste presenti viene eseguita

```
if espressione_logica_1
    comandi 1
elseif espressione_logica_2
    comandi 2
elseif espressione_logica_3
   comandi 3
end
```

Statement condizionali

Switch

- Lo statement switch agisce come una sequenza elseif
- Solo una lista di comandi viene eseguita

```
switch espressione (scalare o stringa)
   case valore 1
        comandi 1
   case valore 2
        comandi 2
   case valore n
        comandi n
end
```

Cicli

Ciclo For

Nel ciclo for la lista di comandi viene eseguita un numero fissato di volte.

```
for index = inizio:incremento:fine
    comandi
end
```

Se '*incremento*' non è definito, il default è 1

Ciclo While

Nel ciclo while la lista di comandi viene eseguita finchè la condizione rimane vera

```
while espressione_logica
    comandi
end
```

Modifiche al flusso

Uscita anticipata da un ciclo

- **Il comando *break* termina immediatamente un qualunque ciclo**
- **Quando Matlab incontra un break, l'esecuzione dello script continua dopo il ciclo**

Costrutti try / catch

- **In modo del tutto analogo al C, Matlab supporta il meccanismo try/catch, passando al blocco catch se si verifca un errore dentro al blocco try**
- **Nel caso in cui il blocco try contiene funzioni e l'errore si verifica dentro queste, si forza anche l'uscita dalle funzioni**

Funzioni

- **Costituiscono dei blocchi elementari di programmazione**
- **Consentono al codice di essere generico e riutilizzabile**
- **Dato un insieme di inputs, eseguono una serie di comandi e ritornano un output**
- **In Matlab, ogni funzione è un M-file**
- **E' prassi nominare il file come la funzione, cioè il file *funcname.m* contiene la funzione definita da:**

```
function outargs = funcname(inargs)
```

- **return termina il calcolo e ritorna al chiamante (opzionale alla fine del file)**

Aggiungere un Help

- **Inserire sempre alcune righe di commento tra la dichiarazione della funzione e la prima riga di codice**
- **Così si abilita automaticamente al funzione help**
- **Le righe di commento sono scandite da lookfor che trova tutte le occorrenza di un termine**

```
function [y] = cube(x)
% Calcola il cubo di x
y = x*x*x;
```

```
>> help cube
Calcola il cubo di x
```

Le variabili nargin/nargout

- **Matlab accetta dichiarazioni di funzioni con un numero variabile di argomenti di input / output**
- **nargin viene usata per sapere quanti sono gli input**
- **nargout viene usata per sapere quanti sono gli output**

```
function [y1, y2] = cube(x1, x2)
if nargin == 1
        y1 = x1*x1*x1;
        y2 = Nan;
elseif nargin == 2
        y1 = x1*x1*x1;
        y2 = x2*x2*x2;
end
```

```
>> [a b] = cube(2,3)
a = 8
b = 27
```

Le liste varargin / varargout

- **Le variabili varargin e varargout individuano gli elenchi delle variabili di ingresso / uscita di una funzione**
- **Tecnicamente sono cell array, cioè semplici strutture seriali di tipo eterogeneo**
- **Si indicizzano con {…} anziché […] come gli array normali (omogenei)**
- **L'uso di varargin/varargout permette di costruire funzioni con argomenti opzionali**

Località nelle funzioni

- **Tutte le variabili interne ad una funzione sono locali, e quindi non visibili all'esterno**
- **Si usa global per renderle globali (pericoloso) oppure mlock per ottenere la persistenza**
- **Si possono creare anche funzioni locali, usate solo da un'altra funzione nello stesso file (quella principale)**

La funzione eval

- **La funzione eval(stringa) è una funzione molto speciale, perché valita la stringa in input come comando Matlab**
- **Essa permette quindi di costruire testi che sono a loro volta programmi e di eseguirli**
- **Una funzione derivata da questa è feval(func, X) la quale valuta la funzione func (passata come stringa) nel punto X**
- **Essa permette di operare quindi su funzioni come argomenti testuali, ma esiste un modo più generale per farlo …**

Puntatori a funzioni

- **Per passare una funzione come parametro basta premettere il simbolo @**

  retval = funcname(@funcarg, …)

- **All'interno di funcname, funcarg non è direttamente valutabile, ma occore feval**
- **Questo perché ciò che viene passato non è la funzione ma solo il suo puntatore**

```
function R = sumfunvet(FUN, X)
R = 0;
for i=1:length(X)
   R = R + feval(FUN,X[i]);
end
```

```
>> R = sumfunvet(@sin, X);
>> disp(R);
```

Importare dati da file

- **Il comando load importa dati da un generico file ASCII in una variabile Matlab (matrice)**

  Nome_variabile = load('filename')

- **Ci sono restrizioni sulla struttura del file, per cui funziona solo su file strutturati in modo regolare (es. tabelle)**
- **Molto potente ma di uso limitato**
- **Esiste il comando simmetrico save**

Spesso si ricorre ad una gestione manuale della importazione dei dati, che la rende una delle principali cause di errore nello sviluppo di codice Matlab

Gestione dei files

- **Sono supportate da Matlab quasi tutte le funzioni f* definite nella libreria <*stdio*> del C per gestire le operazione di I/O da file**
  - **fopen**
  - **fclose**
  - **fseek / ftell / frewind**
  - **fscanf**
  - **fprintf**
  - **fgetl**
- **Ogni file è identificato da un numero intero (file_id)**
- **Le funzioni fscanf e fprintf ammettono i tradizionali formati del C (%d, %f, %c, …)**

Importazione semplificata

- **Matlab supporta due tipi di importazione semplificata, eseguibili con un unico comando**
  - **File di testo** **textread**
  - **File Excel** **xlsread**
- **Matlab supporta anche l'inportazione da tastiera tramite il comando input**

```
>> nome_variabile = input('stringa di prompt');
```

Files binari interi

- **Le operazioni di I/O da file binari sono supportati da fread e fwrite previa definizione del formato**
  - 'schar' Signed character; 8 bits
  - 'uchar' Unsigned character; 8 bits
  - 'int8' Integer; 8 bits
  - 'int16' Integer; 16 bits
  - 'int32' Integer; 32 bits
  - 'int64' Integer; 64 bits
  - 'uint8' Unsigned integer; 8 bits
  - 'uint16' Unsigned integer; 16 bits
  - 'uint32' Unsigned integer; 32 bits
  - 'uint64' Unsigned integer; 64 bits

File binari floating point

- **I formati floatiing point supportati sono**
  - 'float32', 'single' Floating-point; 32 bits
  - 'float64', 'double' Floating-point; 64 bits
- **Per default Matlab memorizza tutti i dati numerici in double**
- **E' possibile specificare nella fopen diversi formati macchina per gestire codifiche particolari**
- **Per sapere quali valori sono rappresentabili con un certo formato si usano realmin e realmax**

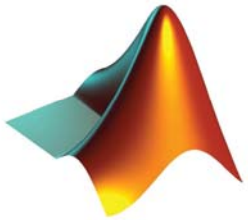
La funzione fread

- **Sintassi possibili:**
  - A = fread(fid)
  - A = fread(fid, count)
  - A = fread(fid, count, precision)
  - A = fread(fid, count, precision, machineformat)
  - [A, count] = fread(...)
- **Argomenti in input:**
  - Fid (identificativo numerico del file)
  - Count (X: legge X elementi, Inf: legge fino a fine file, [m,n]: legge q.b. a riempire una matrice m x n
  - Precision (specifica il formato dei dati in input)
  - MachineFormat (specifica la classe di formati binari utilizzata)

Stringhe di caratteri

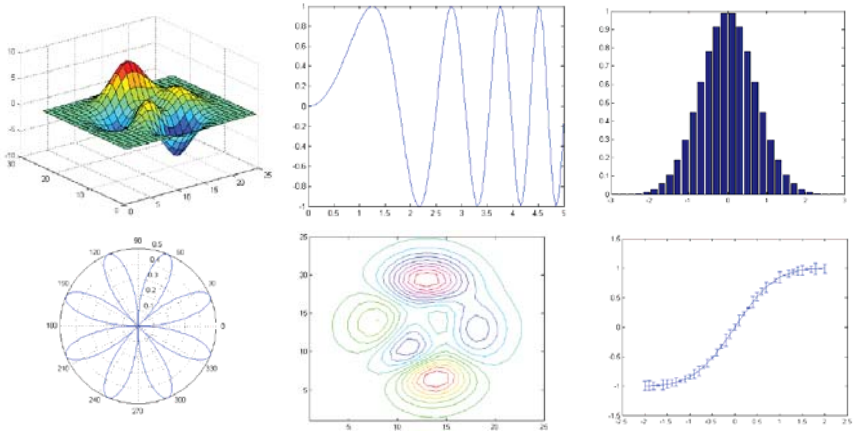
- **Una stringa in Matlab è un insieme di caratteri racchiuso tra apici ', rappresentato come vettore riga**
- **L'eventuale apice interno va scritto come doppio apice ''**
- **Sono supportate le funzionalità in stile C come strcmp, findstr, etc.. ma con parecchie varianti possibili**
  - pos = findstr('stringa di prova','a');
  - nella variabile pos si ottiene il vettore [7 16]

L'ambiente Matlab
per le applicazioni industriali
(Parte 2 – Grafica 2D e 3D)



Cosa si può fare

- **Matlab ha un ottimo motore per la generazione di grafici, in grado di produrre qualuqnue tipo di rappresentazione dei dati**



Generazione dei dati

- **Il motore grafico di Matlab non usa le funzioni ma solo array di numeri. Date quindi le funzioni**
  - **a=t^2**
  - **b=sin(2*pi*t)**
  - **c=exp(-10*t)**
  - **d=cos(4*pi*t)**
  - **e=2*t^3-4*t^2+t**
- **queste vanno calcolate in uno specifico intervallo campionato**
- **L'intervallo di definisce di norma con la sintassi inizio:passo:fine, oppure con linspace(inizio,fine,campioni)**

```
t=0:0.01:10;  %assegna il vettore delle ascisse
y=t.^2; %calcola il vettore delle ordinate
        % ma solo nell'intervallo specificato
```

La funzione plot()

- **La più semplice funzione di grafica è plot()**
- **Cosa succede scrivendo plot(y)?**
  - Matlab genera automaticamente una figura, disegna i punti corrispondenti ai dati y e li connette con linee
  - L'asse x non è corretto (Matlab usa gli indici come default)
- **plot(x,y) risulta simile ma ora l'asse x è corretto**
- **plot(x1,y1,s1,x2,y2,s2, …) plottano più funzioni con un solo comando**
- **Se x è una matrice, plot(x) disegna le colonne come tracce separate**

La funzione plot()

- **Se si plottano in sequenza a e b si vede solo b**
- **Matlab sostituisce ogni plot con il successivo, in assenza di altre istruzioni**
- **Per sovrapporre i due plot nella stessa figura bloccata si usa il comando hold on (hold off disabilita il blocco)**
- **Per avere I plot in figure diverse si usa il comando figure**

```
plot(t,a)
plot(t,b)
```

```
% Disegna a e b in un plot
plot(t,a);
hold on;
plot(t,b);
```

```
% Disegna due plot
plot(t,a);
figure;
plot(t,b);
```

Proprietà delle linee

- **Senza specifiche, tutti i plot sono fatti nel colore di default … blu**
- **Tutti gli attributi grafici sono modificabili selezionando le opzioni ammesse dal comando plot**

```
% linea rossa
Plot(t,a,'r');
Hold on;
% linea nera
Plot(t,b,'k');
% linea a punti verdi
Plot(t,c,'g.');
% linea a croci cyan
Plot(t,d,'cx')
% linea tratteggiata %
a cerchi magenta
Plot(t,e,'--om')
```

Line Style Specifiers

Specifier	Line Style
-	Solid line (default)
--	Dashed line
:	Dotted line
-.	Dash-dot line

Color Specifiers

Specifier	Color
r	Red
g	Green
b	Blue
c	Cyan
m	Magenta
y	Yellow
k	Black
w	White

Marker Specifiers

Specifier	Marker Type
+	Plus sign
o	Circle
*	Asterisk
.	Point
x	Cross
'square' or s	Square
'diamond' or d	Diamond
^	Upward-pointing triangle
v	Downward-pointing triangle

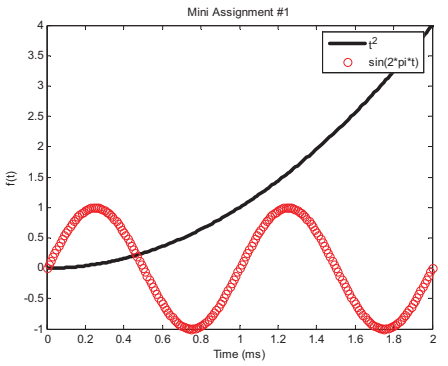
Etichette, Titolo e Legenda

- **Per aggiungere etichette agli assi x e y, si usano I comandi xlabel e ylabel**
- **Per aggiungere un titolo si usa il comando title**
- **Per aggiungere una legenda si usa il comando legend**

```
plot(t,a,t,b,'r',t,c,'--om');  %genera tutti i plot in un colpo
title('Random Plots')
xlabel('t(ms)');
ylabel('f(t)')
legend('Funzione 1','Funzione 2','Funzione 3');
```

Esempio

- **Disegnare a come linea nera spessa**
- **Disegnare b come una serie di cerchi rossi**
- **Etichettare gli assi, aggiungere titolo e legenda**

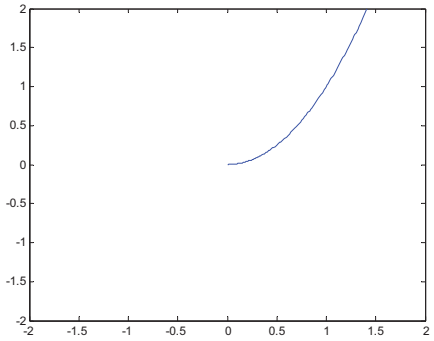



```
figure
plot(t,a,'k','LineWidth',3);
hold on;
plot(t,b,'ro')
xlabel('Time (ms)');
ylabel('f(t)');
legend('t^2','sin(2*pi*t)');
title('Mini Assignment #1')
```

Il comando axis

- **Il comando axis modifica l'intervallo visualizzato sul grafico ed aggiunge ulteriori controlli**
  - Axis([xmin xmax ymin ymax])
  - Axis equal;
  - Axis square;

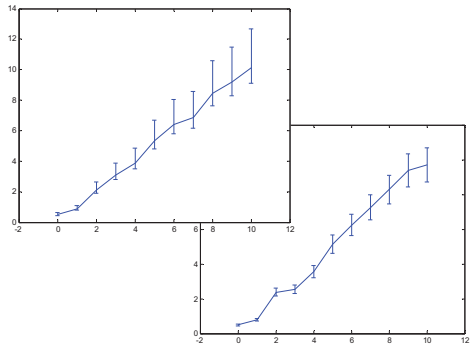
```
Figure;
Plot(t,a,'r');
Axis([-2 2 -2 2]);
```



Le barre di errore

- **Oltre ai punti, Matlab può facilmente disegnare anche gli intervalli di errore per ogni punto con il comando errorbar**
  - errorbar(x,y,e) oppure errorbar(x,y,elow,eup);
  - genera la curva e le barre di errore a y+/-e oppure (y-elow,y+eup)

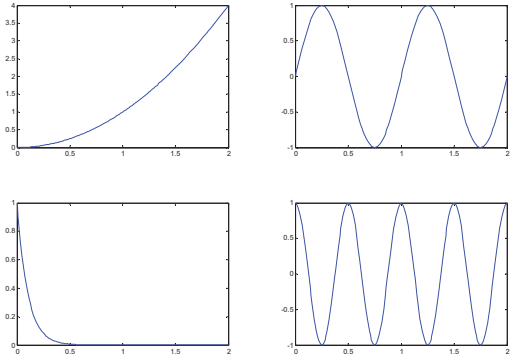
```
t2 = 0:1:10;
f = t2+(rand(1,length(t2))-0.5);
err1 = 0.1*f;
err2_l = 0.1*f;
err2_u = 0.25*f;
errorbar(t2,f,err1);
figure;
errorbar(t2,f,err2_l, err2_u);
```



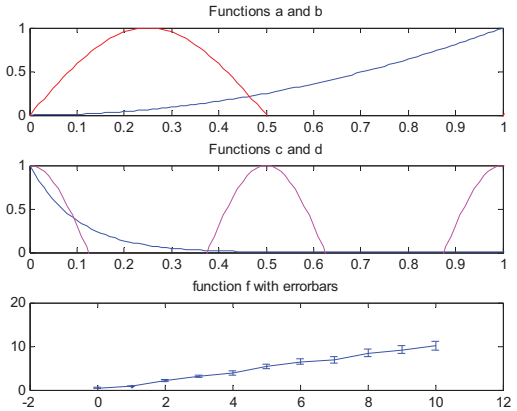
La funzione subplot()

- **Con subplot è possibile realizzare una struttura matriciale di grafici omogenei, che rende più agevole la comparazione**
- **Ogni subplot identifica un particolare grafico su cui agiscono le plot successive**

```
figure;
subplot(2,2,1)
plot(t,a);
subplot(2,2,2)
plot(t,b);
subplot(2,2,3)
plot(t,c);
subplot(2,2,4)
plot(t,d);
```



Esempio

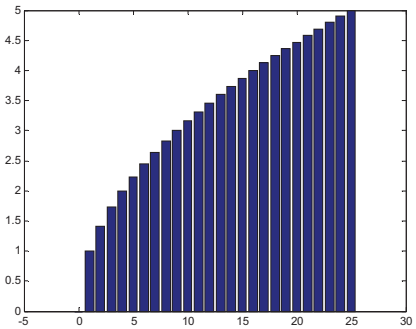



```
figure
subplot(3,1,1)
plot(t,a,t,b,'r');
axis([0 1 0 1]);
title('Functions a and b')
subplot(3,1,2)
plot(t,c,t,d,'m');
axis([0 1 0 1]);
title('Functions c and d')
subplot(3,1,3)
errorbar(t2,f,err1);
title('function f with errorbars')
```

Grafici a barre

- **Oltre ai grafici a linea, sono spesso utili rappresentazioni di dati tramite barre**
- **Il comando Matlab bar(x,y,lar) genera un grafico a barre dei dati x,y con larghezza specificata (default 1)**
- **Può essere usato insieme a tutte le altre opzioni viste finora**

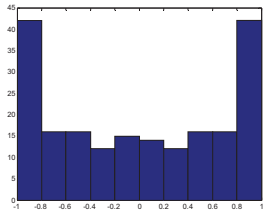
```
t = 0:1:25;
f = sqrt(t);
bar(t,f);
```

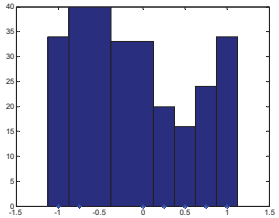


Istogrammi

- **La funzione Matlab hist(y,m) genera un istogramma del vettore di dati y distibuito su m accumulatori**
- **La variante hist(y,x) definisce anche i valori degli accumulatori x**
  - La funzione histc permette di definire i bordi degli accumulatori anzichè i centri
- **Può essere usata con tutte le opzioni grafiche viste finora**

```
hist(b,10);
Figure;
hist(b,[-1 -0.75 0 0.25 0.5 0.75 1]);
```





Esercizio

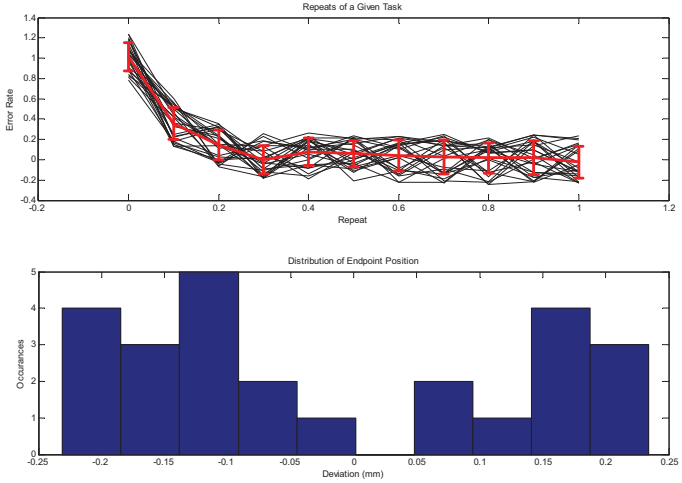
- **Generare un campioni di dati composto da N=25 istanze di una sequenza in parte deterministica (funzione) ed in parte casuale**
- **Creare mean_x come media di tutti i valori assunti in ogni punto di x**
- **Creare std_x come deviazione standard di tutti i valori assunti in ogni punto di x**
- **Graficare le sequenze originali e la media con barra di errore pari alla deviazione standard**
- **Graficare anche l'istogramma dei valori finali del processo**

```
t = 0:0.1:1
for(i=1:25)
      x(i,:) = exp(-10.*t) + 0.5*(rand(1,length(t))-0.5);
end
```

Svolgimento

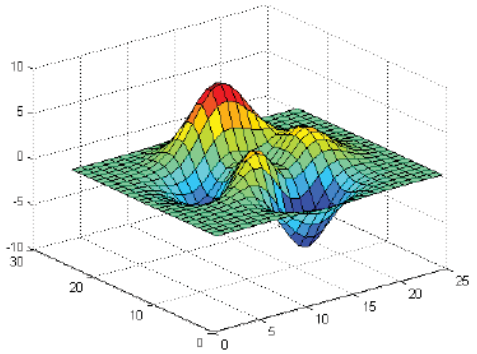
```
t = 0:0.1:1
for(i=1:25)
   x(i,:) = exp(-10.*t) + 0.5*(rand(1,length(t))-0.5);
end
mean_x = mean(x);
std_x = std(x);
figure
subplot(2,1,1)
plot(t,x,'k'); hold on;
errorbar(t,mean_x,std_x,'--r','LineWidth', 3);
title('Repeats of a Given Task')
xlabel('Repeat');
ylabel('Error Rate');
subplot(2,1,2)
hist(x(:,11),10)
title('Distribution of Endpoint Position');
xlabel('Deviation (mm)')
ylabel('Occurances')
```

Risultato



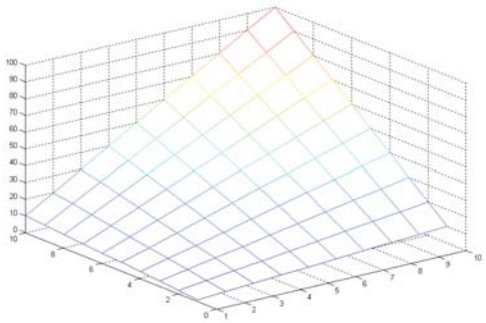
Grafici 3D

- **Matlab fornisce una vastissima scelta di opzioni per la grafica di dati in 3D**
- **Le funzioni di base utilizzate sono mesh, surf, contour pensate per visualizzare grafici del tipo Z=f(X,Y)**



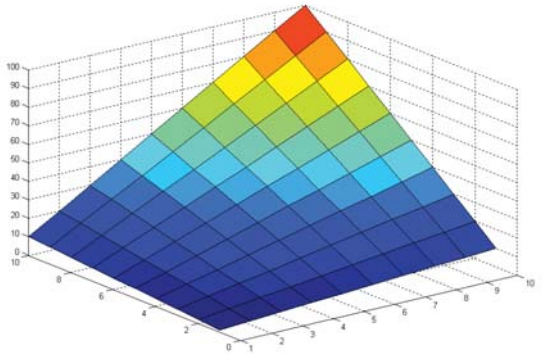
La funzione mesh

- **La funzione mesh connette una serie di punti discreti con una struttura reticolare (griglia o mesh)**
  - mesh(x,y,z) dove X(i) and Y(j) sono le posizioni dei nodi della griglia e Z(i,j) è il valore assunto in ogni nodo
  - mesh(Z) assume che X e Y siano 1..N e 1..M



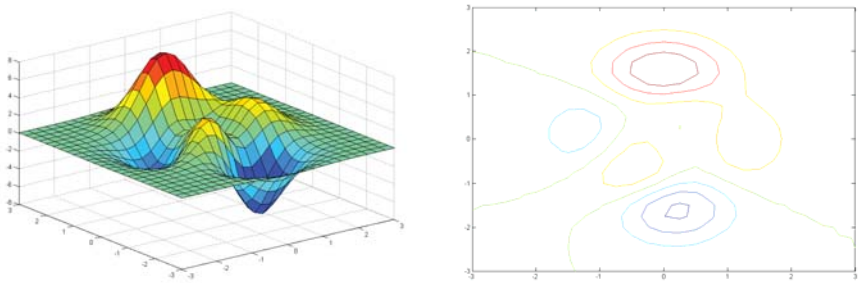
La funzione surf

- **E' concettualmente identica alla mesh, con l'unica differenza che la griglia è riempita con sfumature di colore**



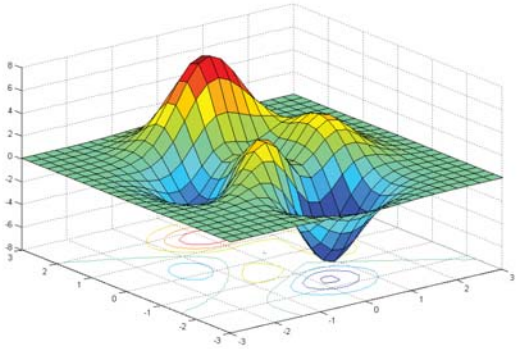
La funzione contour

- **Proietta i punti di uguale altezza in 3D (curve di livello) su un piano 2D sottostante**
- **Per il resto è analoga a surf o mesh – contour(x,y,z)**



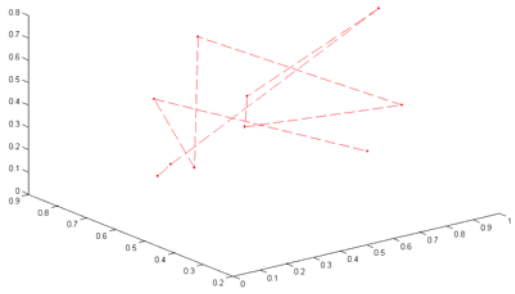
La combinazione meshc,surfc

- **Combina la visualizzazione della superfice o della griglia con il grafico delle curve di livello**
- **Per il resto è analoga a surf o mesh – meshc(x,y,z)**



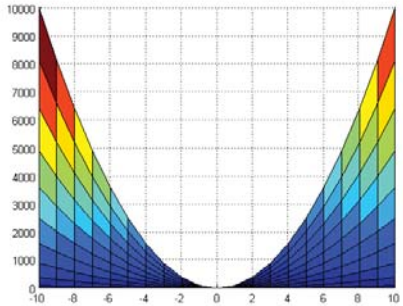
La funzione plot3

- **La funzione plot3 Disegna linee e punti nello spazio 3D, in modo analogo alla plot, ma ora per terne di dati (x,y,z) generiche**
- **Plot3(x,y,z) assume che i 3 vettori abbiano la stessa lunghezza**
- **Ammette le stesse opzioni grafiche di plot.**



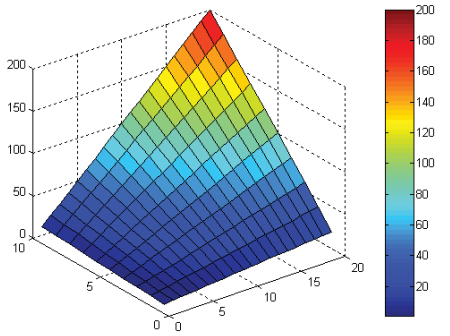
Gestione del punto di vista

- **E' possibile cambiare il punto da cui si osserva un grafico 3D con la funzione view**
  - view(az,el)
  - az = Azimut (rotazione attorno all'asse z)
  - el = Elevazione (rotazione rispetto al piano xy)
- **In modalità interattiva si usa rotate3D**



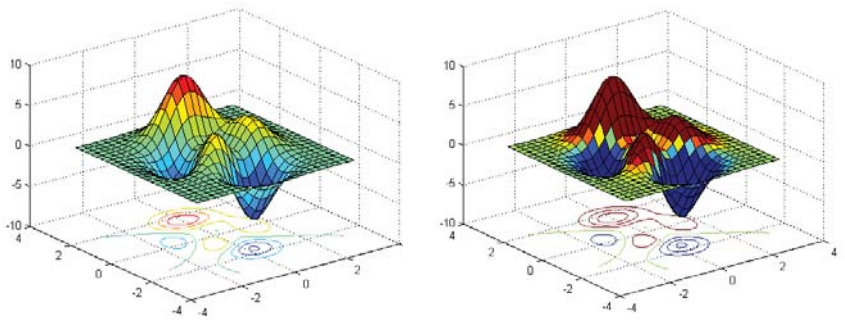
La colorbar

- **Nei grafici 3D è spesso utile affiancare al grafico una barra di colore che indica la corrispondeza tra colori e valori di altezza z, usando la fuznioen colorbar**
- **colorbar('vert') inserisce la colorbar in verticale**
- **colorbar('horiz) inserisce la colorbar in orizzontale**



Il comando caxis

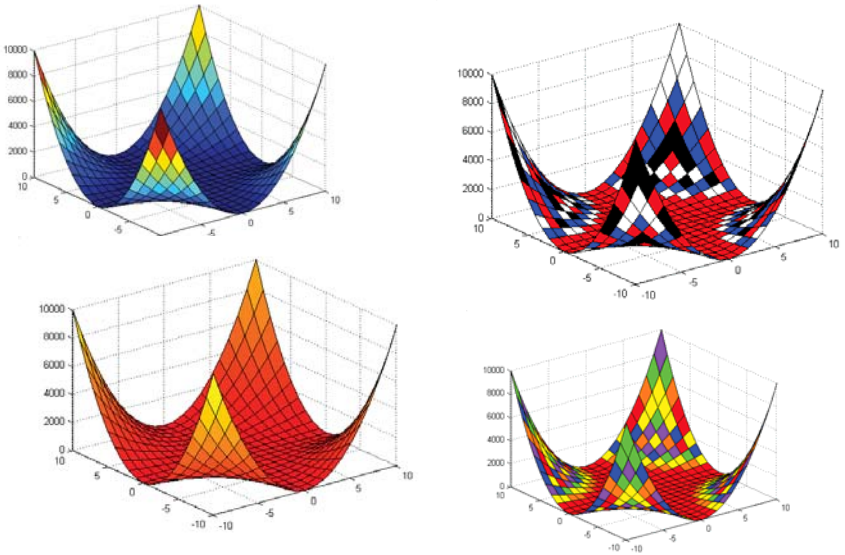
- **Il comando caxis permette di definire l'intervallo da visualizzare a colori**
- **caxis([min max]) sceglie i colori dalla mappa corrente e li assegna all'intervallo. I valori fuori dall'intervallo prendono colori fissi**



Gestione della mappa di colori

- **Matlab fornisce numerose opzioni per personalizzare la mappa di colori utilizzata in un grafico**
  - colormap(*colormap_name)* assegna una nuova mappa ad un grafico. Questa può essere una di quelle già disponibili oppure una realizzata *ad hoc*
  - hsv, jet, autumn, vga, summer, …
  - Consultare l'help graph3D per altre informazioni
  - Usare la funzione colormapeditor per costruire la propria mappa di colore

Esempi



Grafica per immagini

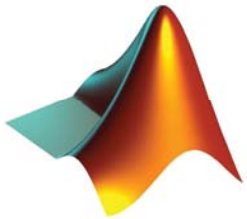
- **Una immagine non è altri che una matrice di numeri dotati di significato pittorico**
- **Viene visualizzata con diversi comandi**
  - image visualizza una matrice di indici della colormap corrente
  - imagesc scala i valori per sfruttare tutta la colormap
  - imshow usa una colormap di default (grigi) o rappresentazioni true color per immagini a colori
- **Uso legato al toolbox di image processing**

Altre funzionalità grafiche

- **Grafici speciali (polari con polar, a torta con pie , etc…)**
- **Effetti di sfumatura (shading)**
- **Sorgenti di illuminazione (light)**
- **Effetti di trasparenza (alpha)**
- **Stampa del grafico (print)**

Per ognuna di queste funzionalità facciamo riferimento all'help ed alle risorse disponibili in rete

L'ambiente Matlab per le applicazioni industriali (Parte 3 – Applicazioni numeriche)



Fitting di dati

- **In generale il fitting si pone l'obiettivo di adattare un generico modello matematico a dei dati sperimentali**
- **Tramite il modello è possibile**
  - Ricostruire dati mancanti (interpolazione)
  - Stabilire delle tendenze (trend analysis)
  - Fare previsioni (estrapolazione)
- **Nel caso più semplice i dati sono rappresentati da due vettori X (input) e Y (output), ed il modello descrive una relazione fra questi, del tipo Y=f(X), con f sconosciuta**
- **Molto spesso f è un polinomio**

Polinomi

- **Matlab dispone di un metodo particolarmente efficace per rappresentare i polinomi tramite vettori**
- **La variabile P=[a b c] rappresenta il polinomio $Y=aX^2+bX+c$**
- **I polinomi sono particolarmente utili per l'analisi dei dati perché sono un semplice modello relazionale tra X e Y**
- **I problemi di fitting con polinomi sono esprimibili con equazioni molto semplici**
- **Il valore del polinomio P nel punto X si calcola con**

  Y=polyval(P,X)

Operazioni tra polinomi

- **I polinomi consentono di effettuare numerose operazioni che continuano a produrre polinomi**
  - Addizione
  - Sottrazione
  - Moltiplicazione – funzione conv

    c = conv(a,b)
  - Divisione – funzione deconv

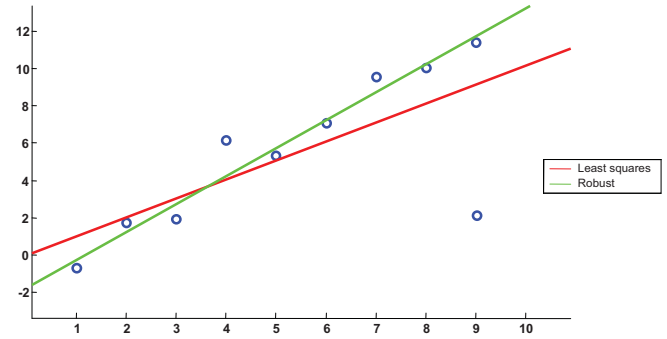
    [q r]=deconv(a,b)
  - Potenza
  - Derivazione – funzione polyder
  - Integrazione –funzione polyint

Radici di un polinomio

- **Un polinomio di grado N ha sempre N radici (complesse)**
- **Un polinomio è quindi individuato univocamente dalle sue radici, cioè i valori per cui si annulla (reali o complessi)**
  - R=root(P) estre le radici del polinomio P
  - P=poly(R) crea un polinomio con radici R
- **Le funzioni root e poly possono quindi essere intese una come l'inversa dell'altra (per radici calcolabili esattamente)**
- **Se N>4 non esistono soluzioni esatte e i dati generati da root sono ottenuti attraverso soluzioni numeriche**

Fitting

- **In generale una operazione di fitting può essere di tipo <u>semplice</u> (tutti i dati sono validi) oppure di tipo <u>robusto</u> (alcuni dati non sono validi e vanno scartati)**
- **Il fitting semplice spesso è quindi inadeguato**



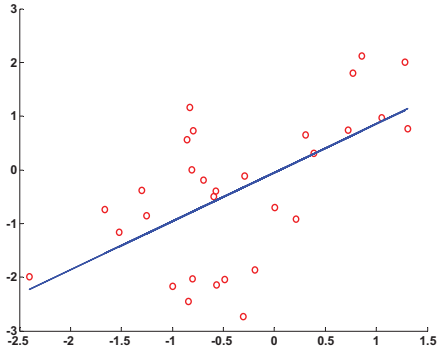
Fitting

- **La tecnica tradizionale di fitting è quella dei minimi quadrati, in cui si cerca un polinomio P di grado N che minimizza la distanza tra tra i dati Y ed i valori P(X)**
- **P = polyfit(X,Y,N)**
- **Se N=1 il fitting è lineare, se N=2 è parabolico, etc. etc.**
- **Il tutto (come sempre accade in Matlab) senza scrivere una riga di codice (o quasi), e senza errori**
- **La finestra di plot fornisce questo ed altri metodi attivabili interattivamente**

Esempio di fitting

- **Tracciare il miglior fitting lineare (retta ai minimi quadrati) per un insieme di dati assegnato**

```
[V1, V2] = textread('testdata2.txt','%f%f','headerlines',1)
P = polyfit(V1,V2,1);
Y = polyval(P,V1);
close all
figure(1)
hold on
plot(var1,var2,'ro')
plot(var1,Y)
```



Fitting robusto

- **Il fitting robusto permette di tenere conto della presenza di possibili dati errati (outliers), che vanno eliminati dal processo**
- **I metodi di fitting robusto sono contenuti nel toolbox "Statistics"**
- **Ad esempio P = robustfit(X,Y) esegue questa operazione**
- **Il comando robustdemo fornisce una valida presentazione di come questo processo lavora**
- **Ogni toolbox possiede in generale uno o più demo (scritti in Matlab e con sorgenti visionabili) che illustrano le principali funzionalità**

Fitting di ordine superiore

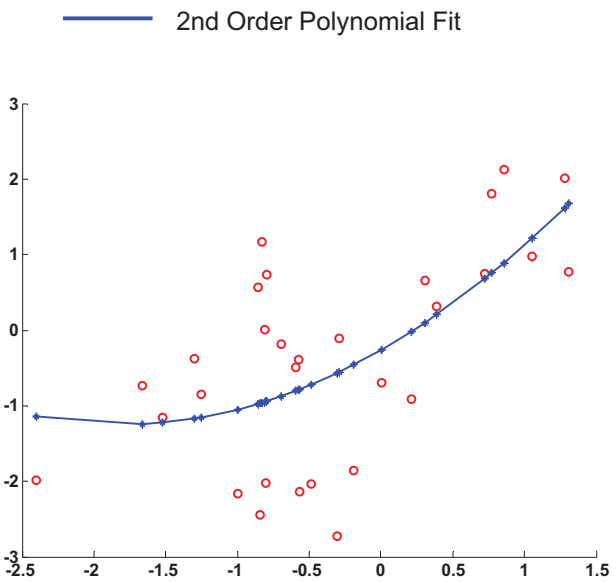
- **Attraverso polyfit si possono calcolare polinomi ai minimi quadrati di ordine arbitrario**

$$y = p_1 x^n + p_2 x^{n-1} + \ldots + p_n x + p_{n+1}$$

- **Spesso è interessante confrontare la curva risultante al variare di N, in modo da stabilire quale valore di N è più appropriato**
- **Questa è la tipica operazione banale in Matlab perchè coinvolge solo diverse chiamate alla polyfit e funzioni plot**

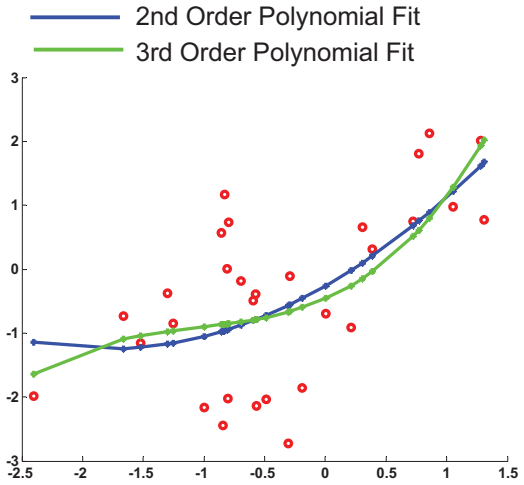
Esempio di fitting quadratico

```
% Lettura dei dati
[V1, V2] = textread('testdata2.txt','%f%f','headerlines',1)
% Calcolo del fitting con un polinomio di secondo grado
P2 = polyfit(V1,V2,2);
Y2 = polyval(P2,V1);
% Plot del risultato
close all
figure(1)
hold on
plot(V1,V2,'ro')
% Modifica alla grafica del polinomio
[sortval, sortind] = sort(V1)
plot(sortval,Y2(sortind),'b*-')
```



Aggiunta di fitting cubico

```
% Calcolo del fitting con un polinomio di terzo grado
P3 = polyfit(V1,V2,3);
Y3 = polyval(P3,V1);
% Aggiunta del grafico
plot(sortval,Y3(sortind),'g*-')
```



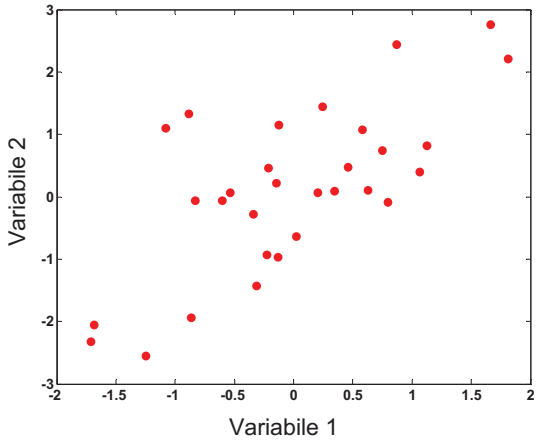
Stabilire la bontà di un fitting

- **La parte difficile di un fitting non è quasi ma il calcolo dei coefficienti del modello, ma stabilire se il risultato è o non è adeguato**
- **Stabilire la qualità di un fitting richiede esperienza, conoscenza del problema, ma soprattutto una visione "statistica" del problema**
- **La statistica c'entra perchè gli errori sono variabili casuali, e la loro gestione è quindi un problema di tipo statistico**
- **Matlab ovviamente aiuta ad affrontare questo problema attraverso diverse funzioni specifiche**

Correlazione tra dati

- **Disponendo di diversi dati strutturati in modo analogo (vettori della stessa lunghezza) si possono impostare test per stabilire se esiste una correlazione tra questi**
- **Matlab dispone di diversi test possibili come il coefficiente di crosscorrelazione**
  - R = corrcoeff(X)
- **Se X è una matrice NxM, contenente M dati misurati N volte, R è una matrice MxM che quantifica l'esistenza di una relazione lineare tra coppie di dati**

Esempio




```
% Calcolo cross-correlazione

R = corrcoef([V1,V2])

R = 1.0000    0.7051
    0.7051    1.0000
```

Interpolazione

- **Nel fitting si usano di solito tutti i dati a disposizione per definire il modello**
- **Nella interpolazione, si è interessati a calcolare il valore "mancante" in un punto Xi, che quindi ci si aspetta dipendere solo dai dati "vicini"a Xi**
- **Non è quindi necessario fittare tutti i dati X per ottenere il solo valore di Y nel punto Xi**
- **Esistono allora diversi modelli di interpolazione che applicano un determinato modello solo in modo "locale"**

Interpolazione

- **Matlab fornisce una funzione di uso generale per l'interpolazione di dati in forma vettoriale**
- **Yi = interp1(X,Y,Xi,'metodo')**
- **Attraverso il parametro 'metodo' si possono selezionare**
  - 'linear' Interpolazione lineare
  - 'splines' Interpolazione con splines cubiche
  - 'cubic' Interpolazione con cubiche di Hermite
  - 'nearest' Interpolazione nearest neighbour
- **Il vettore X deve essere crescente, e se non lo fosse, occorre quindi precedere l'interpolazione con un ordinamento dei dati tramite sort**

Esempio

```
%esempi di interpolazione 1D
%generazione dei dati reali
X=-pi:0.01:pi;
Y=sin(X);
%dati campionati
XC=-pi:1.0:pi;
YC=sin(XC);
%calcolo diversi interpolatori
YN=interp1(XC,YC,X,'nearest');
YL=interp1(XC,YC,X,'linear');
YS=interp1(XC,YC,X,'spline');
%grafica
figure
plot(XC,YC,'ok');
hold on
plot(X,YN,'g',X,YL,'r',X,YS,'b');
```

Interpolazione 2D

- **L'interpolazione si estende direttamente anche a dati 2D (superfici)**
- **Zi=interp2(X,Y,Z,Xi,Yi,'metodo')**
- **X,Y,Z sono tutte matrici della stessa dimensione che descrivono un reticolo regolare**
- **Esistono anche interpolazioni 2D pensate per disegnare superfici note solo su punti sparsi. In questi casi non sarebbe difatti possibile il normale plot tramite le funzioni mesh e surf**
- **Zi=griddata(X,Y,Z,Xi,Yi,'metodo')**
- **X,Y,Z sono vettori che descrivono i dati sparsi**

Esempio

```
% interpolazione di dati 2D sparsi
% genero 100 dati vettoriali a caso tra -4 e 4
N=100;
X=rand(N,1)*8-4;
Y=rand(N,1)*8-4;
Z=sin(X).*cos(Y);
% voglio graficare questa superficie
% creo allora un reticolo 2D tra -4 e 4
[X1,Y1]=meshgrid(-4:0.1:4,-4:0.1:4);
% e ci interpolo i dati
Z1=griddata(X,Y,Z,X1,Y1,'linear');
% grafica delle superficie
figure(1)
surf(X1,Y1,Z1);
% grafica della superficie vera
figure(2)
Z2=sin(X1).*cos(Y1);
surf(X1,Y1,Z2);
```

Sovra/sottocampionamento

- **Un caso particolare di interpolazione molto frequente nella pratica è quello in cui:**
  - Dato un vettore di N dati X
  - Si vuole ottenere un vettore di M dati Y
  - Con le stesse caratteristiche "statistiche"
- **Se M>N si parla di sovracampionamento, altrimenti di sottocampionamento**
- **Essendo un problema molto comune, Matlab ha già pronta la soluzione, basata sulla trasformata di Fourier**

  Y=interpft(X,M)

Sovra/sottocampionamento

- **Il metodo utilizzato è basato sulla teoria di Shannon, secondo la quale, sotto opportune condizioni, una sequenza discreta di dati (segnale campionato) porta tutta l'informazione del segnale continuo**
- **Il segnale può in questi casi essere quindi arbitrariamente ricampionato in altri punti**
- **Funziona bene per dati variabili in modo lento (smooth), altrimenti può produrre valori non adeguati**
- **Approccio generalizzabile a dati N-dimensionali**
  - ad esempio le immagini A=imresize(B,[m n])

Smoothing

- **Le operazioni di smoothing hanno lo scopo di ridurre gli errori presenti nei dati**
- **Esistono metodi di smoothing lineare (media o polinomi) o non lineare (mediana)**
- **Matlab fornisce implementazioni pronte di tutti quanti (sia nel modulo base che nei vari toolbox)**
- **Alcuni esempi**
  - filter filtraggio lineare convolutivo
  - medfilter1 filtraggio mediano

Esempio

```
%esempi di smoothing 1D
close all;
%generazione dei dati reali rumorosi
X=0:0.01:pi;
amp=input('ampiezza rumore ...');
Y=sin(X)+amp*(rand([1 length(X)])-0.5);
%calcolo diversi tipi smoothing a passo N
N = 11;
F = ones([1 N])/N;
YF=filter2(F,Y);
YM=medfilt1(Y,N);
%grafica
figure
plot(X,Y,'k');
hold on
plot(X,YF,'g','LineWidth',5);
hold on
plot(X,YM,'r','LineWidth',5);
```

Calcolo su funzioni

- **Matlab mette a disposizione molte funzionalità per operare su funzioni definite dall'utente per cercare:**
  - **Zeri, con la funzione fzero**
  - **Minimi o massimi con la funzione fmin**
- **Queste funzioni si appoggiano a librerie di calcolo numerico integrate appositamente in Matlab**
- **La funzione utente (definita in un M-file) viene passata come parametro**
  - X = fzero(@funzione,X0)
  - X = fmin(@funzione,X1,X2)
- **Ammettono entrambe un numero elevato di opzioni che condizionano la qualità e la precisione della ricerca**

Esercizio

- **Un problema di marketing**
  - Una azienda deve decidere il prezzo di prodotto che massimizza il profitto sapendo che:
  - profitto=(prezzo-costo)*venduto
  - Il venduto è funzione del prezzo
  - Il costo è funzione del venduto
  - Queste funzioni sono espresse in forma parametrica in base a modelli econometrici
- **Gli economisti aziendali dicono:**
  - modello del costo c(v)=cf+cv*exp(-v/vrif)
  - modello del venduto $v(p)=k*(prif/p)^a$
- **L'analista deve valutare il prezzo ottimale in funzione dei parametri significativi del problema**

Esercizio

- **Intervistando la produzione si desume:**
  - Costo fisso cf = 100
  - Costo variabile cv = 100
  - Il costo variabile si riduce parecchio se si vendono almeno 1000 pezzi (vrif)
- **Intervistando il marketing si desume:**
  - Se il prodotto costasse 150, se ne venderebbero almeno 3000 pezzi
  - Se costasse di più se ne venderebbero di meno (!).
  - Con successiva richesta emerge che se costasse il doppio se ne venderebbero ¼.
- **L'analista decide che ora ha abbastanza dati per partire con il calcolo e si mette al lavoro con il suo Matlab**

Esercizio

- **Suggerimenti per la soluzione:**
  - Utilizzare una funzione che calcola il profitto per un prezzo assegnato
  - Provare un approccio semplice basato sul campionamento di un ragionevole intervallo di prezzo. Calcolare il profitto per ogni valore di prezzo e trovare il massimo con max
  - Più raffinato: massimizzare direttamente la funzione profitto nell'intervallo di prezzo selezionato, passandola come funzione alla fmin (o similare)

Soluzione

Operazioni tra insiemi

- **Matlab consente anche di operare tra dati simbolici anziché numerici, intesi come elementi di insiemi**
- **A=[1 2 3 4] è un insieme di 4 elementi**
- **Gli elementi possono essere qualunque cosa (es. caratteri), quindi gli insiemi sono array di celle**
- **La funzione più potente che opera su un insieme è unique. Dato A essa ritorna un insieme B ordinato e con ogni elemento contenuto una sola volta**
  - A=[1 2 5 2 1 3 3 7]
  - B=unique(A)
  - [1 2 3 5 7]

Operazioni tra insiemi

- **Unione (elementi che stanno in almeno uno)**
  - C=union(A,B)
- **Intersezione (elemendi che stanno in entrambi)**
  - C=intersect(A,B)
- **Differenza (elementi che stanno solo nel primo)**
  - C=setdiff(A,B)
- **Appartenenza (presenza e posizione di un elemento dato)**
  - index=ismember(A,b)
- **Dimensione (numero di elementi)**
  - b=length(A)

Esercizio

- **Sfruttando le funzioni degli insiemi, realizzare uno script Matlab che risolve uno schema di sudoku (almeno quelli semplici)**
- **Il sudoku prevede la definizione di una matrice 9x9 in cui:**
  - Ogni riga contiene tutti i simboli da 1 a 9
  - Ogli colonna contiene tutti i simboli da 1 a 9
  - Ognuna delle 9 sottomatrici 3x3 contiene tutti i simboli da 1 a 9
- **All'inizio la matrice contiene già un simbolo in alcune posizioni (quante e dove a seconda della complessità)**
- **Il vincolo costruttivo prevede che debba esistere sempre una ed una sola soluzione compatibile con la condizione iniziale**

Esercizio (seguito)

- **Alcuni suggerimenti per la soluzione**
  - Usare lo zero per rappresentare le posizioni vuote
  - Usare find per trovare tutte le posizioni vuote
  - Usare gli insiemi per trovare i valori ammessi o non ammessi in una certa posizione
  - Assegnare una posizione solo quando c'è certezza sul valore
  - Gestire il possibile fallimento della strategia
- **Vincolo: non utilizzare cicli for (se possibile)**

Soluzione

Equazioni differenziali

- **Spesso ci si trova di fronte a sistemi il cui modello prevede l'uso di equazioni differenziali (o anche sistemi)**
- **Matlab mette a disposizione soluzioni numeriche di vario livello per integrare direttamente queste equazioni a partire da condizioni iniziali note**
- **Si prevede sempre la riduzione alla forma normale**
  - x' = f(t,x)
- **La funzione incognita x(t) può essere scalare o vettoriale (sistema)**

Equazioni differenziali

- **La sintassi di tutti i metodi (es. ode45) è la stessa:**
- **[t x]=ode45(@funzione,Tint,x0)**
  - [t,x] vettori di uscita per x(t)
  - @funzione punta alla funzione che descrive f
  - Tint è l'intervallo dove calcolare il risultato
  - X0 è la condizione iniziale
- **Il risultato è in una forma pronta per essere graficata con plot**
- **Esistono poi numerose parametrizzazioni definibili tramite il comando odeset('parametro',valore,….)**

Esercizio

- **Studiare l'evoluzione di due popolazioni descritte dalle equazioni di Volterra**
  - x'=(a-by)x
  - y'=(cx-d)y
- **Assegnare i parametri a,b,c,d e le condizioni iniziali x0,y0 provando varie combinazioni**
- **Capire il significato delle varie situazioni che si producono**

Soluzione