Calcolo Numerico

A.A. 2013-2014

Esercitazione n. 3

21-03-2014

Grafica: plot 2D

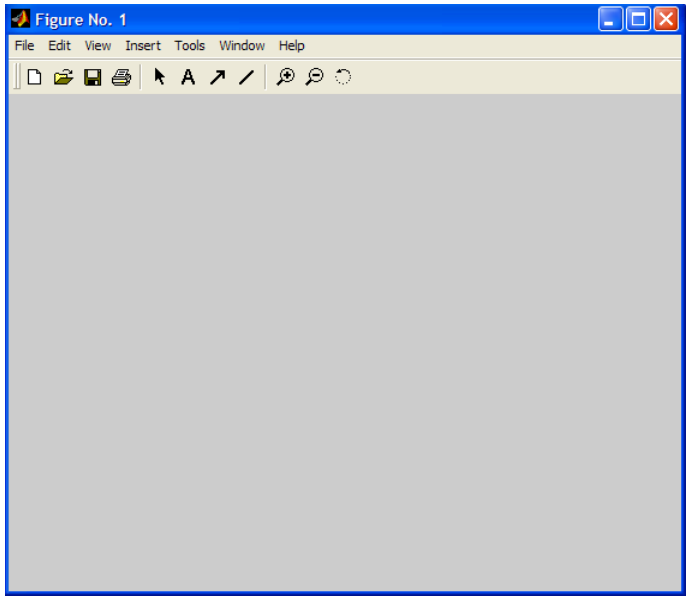
Grafica

- MATLAB è molto più di un semplice software in grado di eseguire calcoli numerici (anche se in forma molto sofisticata). Con MATLAB è possibile realizzare **grafici di funzioni** anche in più dimensioni e realizzare veri e propri programmi. In MATLAB è possibile
  - disegnare funzioni in 2D e 3D
  - rappresentare dati memorizzati in vettori e matrici in molti modi differenti
- Il comando **`plot(x,y)`** si usa
  - per **rappresentare punti nel piano**
  - per disegnare il **grafico di una funzione** **`y=f(x)`**
    - **`x e y`** devono essere vettori di ugual misura

Figure

- **`figure`**
  - apre la finestra grafica al quale viene associato un numero
- **`figure(n)`**
  - **`n`** è il numero associato alla finestra grafica
- **`id = figure`**
  - ad ogni figura è associato un identificativo **`id`** che consente di gestirne le proprietà grafiche

```
>> figure(1)

>> id = figure;
```



Plot - sintassi

```
plot(vettore_x, vettore_y, 'opzioni')
```

- **vettore_x** e **vettore_y** devono avere lo stesso numero di componenti, sono i vettori dei dati (ascisse e ordinate dei punti)
- **opzioni**: è una stringa opzionale che definisce il tipo di colore, di simbolo, di linea usato nel grafico, …

Esempi colore: **m** magenta, **r** rosso, **g** verde, **b** blu, **w** bianco, **k** nero, **y** giallo, **c** ciano

Esempi tipo di linea: **-** continua (default), **--** tratteggiata, **:** punteggiata, **-.** punto-linea

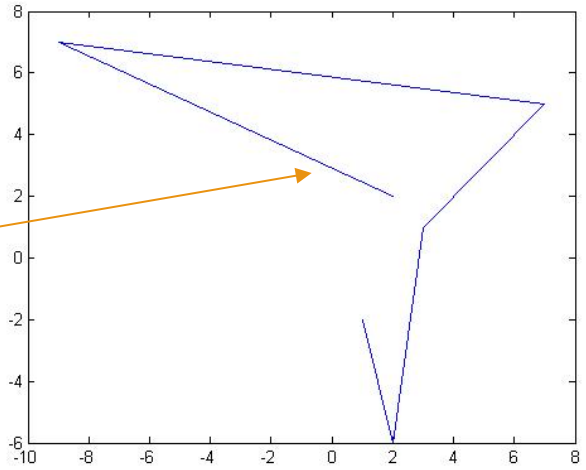
Esempi simbolo: **+** croce, **o** cerchietto, ***** asterisco, **x** ics, **.** punto, **s** quadratino, **d** diamante, **v** triangolo

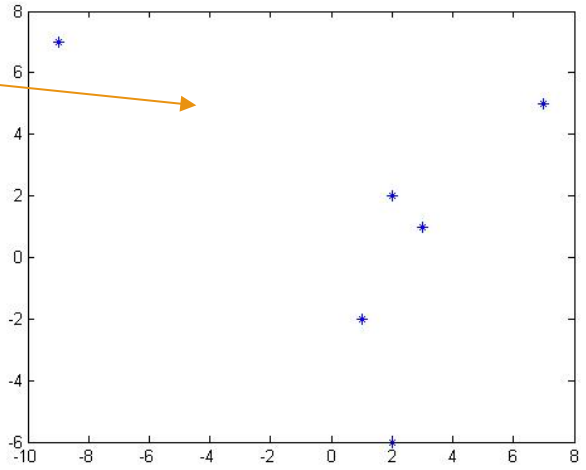
help plot per vedere quali sono le varie opzioni

Rappresentazione dei punti

- Per rappresentare dei punti nel piano

```
>> x = [1 2 3 7 -9 2];
>> y = [-2 -6 1 5 7 2];
>> plot(x,y)
>> figure(2)
>> plot(x,y,'*')

% realizza il grafico del
% vettore y rispetto ai propri
% indici
>> plot(y)
```





Rappresentazione dei punti

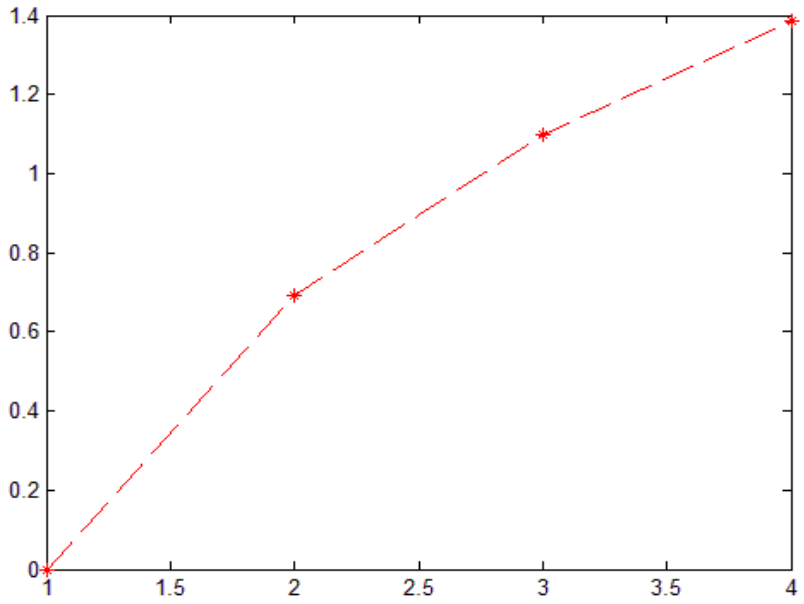
- La funzione **`plot(x,y)`** rappresenta graficamente la tabella di valori ottenuta raccordando con segmenti di retta nel piano cartesiano **`Oxy`** i vertici **`(x(i),y(i))`** in modo ordinato al variare di i da **`x(1)`** a **`x(end)`.**

- Se vogliamo evidenziare i vertici della poligonale possiamo utilizzare l'istruzione **`plot(x,y,'-g')`** che disegna i punti **`(x(i),y(i))`** in verde. La funzione non cambia, sono i parametri di ingresso che cambiano, il terzo parametro aggiunto serve a specificare lo stile della linea del grafico (pallini in verde)

Grafici di funzione

- Esempio: Siano **`x = [1 2 3 4]`** e **`y = log(x)`**. Disegnare **`y`** in funzione di **`x`** usando una linea tratteggiata rossa e marcando i punti della curva con asterischi
  - si creano due vettori x e y contenenti rispettivamente la successione di valori nell'intervallo ed i corrispondenti valori della funzione

```
>> x = 1:4;
>> y = log(x);
>> plot(x,y, 'r--*')
```

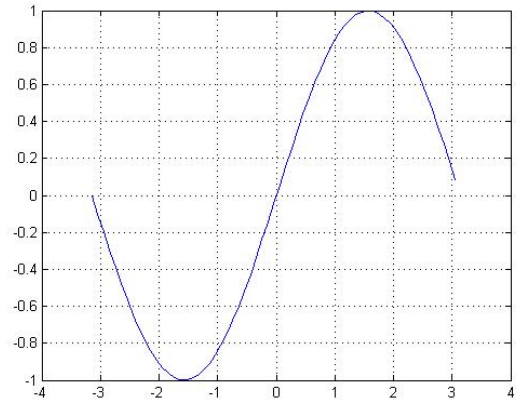


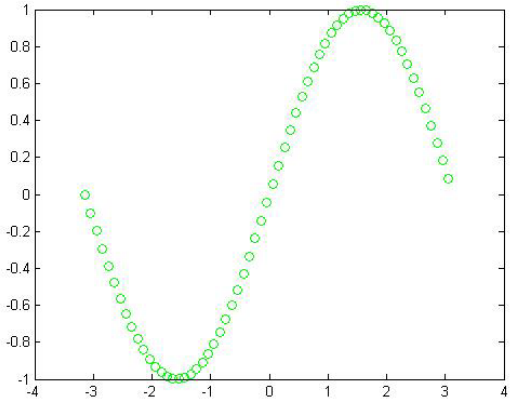
Se volessimo produrre un grafico privo di spigoli è sufficiente aumentare il numero di punti in modo che i segmenti di raccordo siano cosi piccoli da dire l'idea di una linea continua

Esempio

Per plottare la funzione **y=sin(x)**

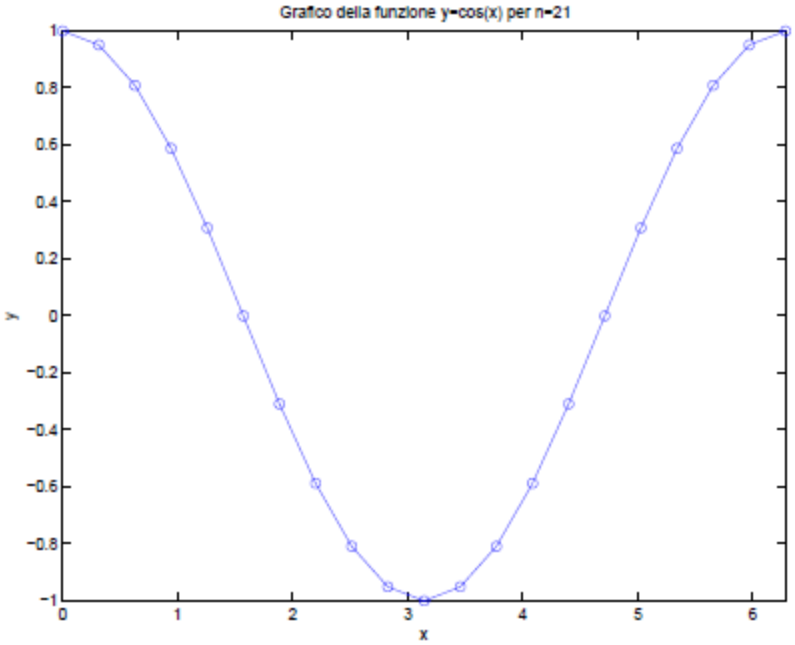
```
% definiamo l'intervallo in cui
% vogliamo disegnare la funzione
x = [-pi:.01:pi];

% definiamo la funzione
y = sin(x);

% disegniamo la funzione
plot(x,y)

% con un terzo parametro di input
figure(2)
plot(x,y, '°g')
```

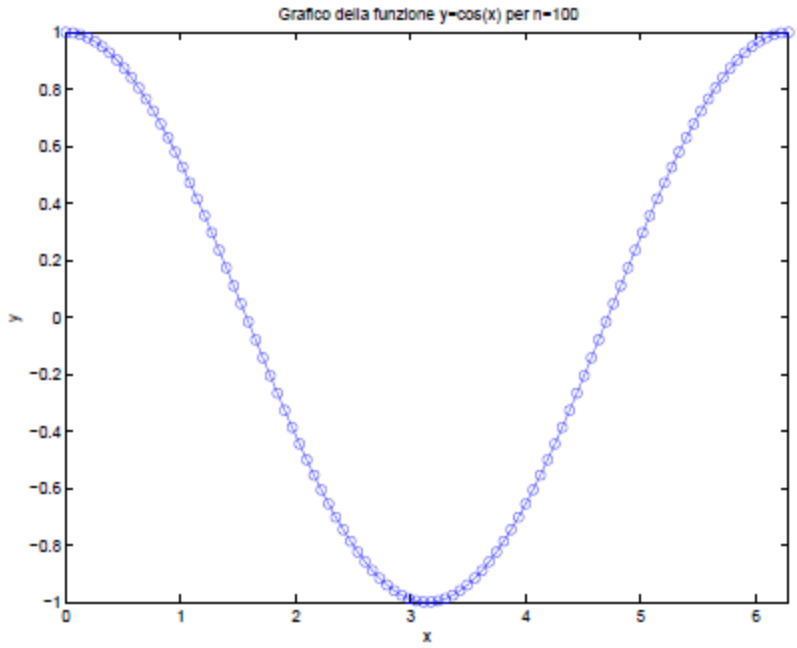




Esempio

Rappresentazione grafica della funzione **`y=cos(x) in [0, 2π]`**





Grafici sovrapposti

- Sia **x** il vettore delle ascisse e **y1 e y2** due vettori aventi la stessa lunghezza di **x**. Per visualizzare sulla **stessa finestra** si usa il comando **hold on** che funziona come un interruttore acceso/spento; tale comando fa sovrapporre tutti i grafici successivi nella stessa finestra grafica fino a quando non si digita il comando **hold off**

```
>> figure
>> plot(x,y1)
>> hold on
>> plot(x,y2)
>> hold off
```

- Equivalentemente si può usare la seguente istruzione che usa in modo automatico linee tipo differenti per i diversi grafici. (le opzioni si possono omettere)

```
>>  plot(x,y1,'opzioni',x,y2,'opzioni')
```

Grafici – etichette

- Esistono molte possibilità per personalizzare un grafico

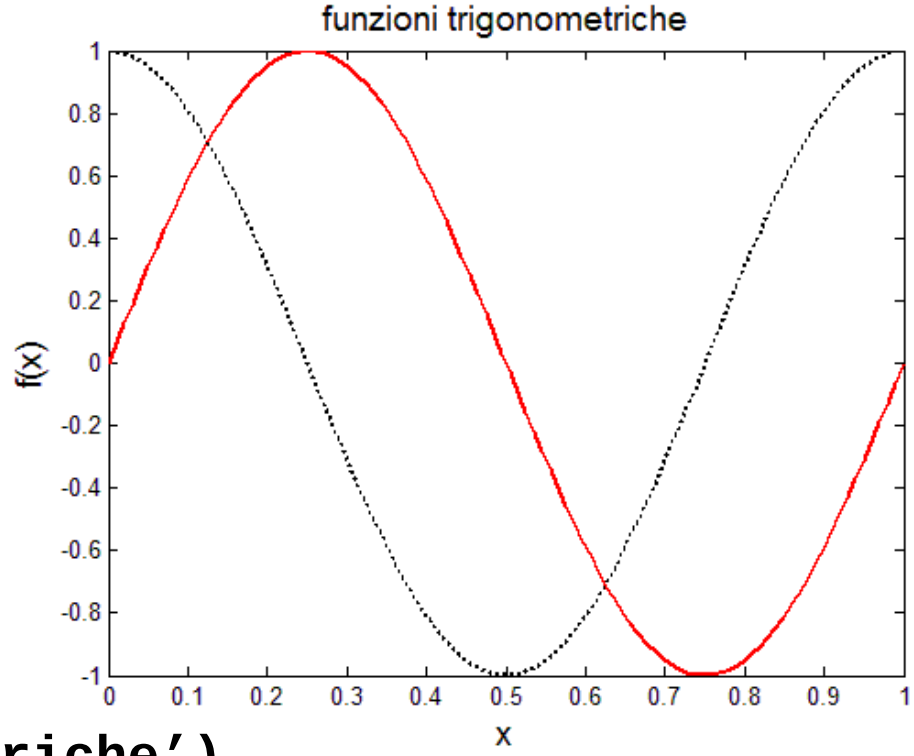
`title('stringa')`	titolo del grafico
`xlabel('stringa')`	etichetta per l'asse delle ascisse (asse x)
`ylabel('stringa')`	etichetta dell'asse delle ordinate (asse y)
`grid`	inserisce una griglia nel grafico
`legend('stringa1', 'stringa2',…)`	legenda
`axis([xmin xmax ymin ymax])`	regola la dimensione degli assi coordinati (determina il rettangolo nel quale si vogliono visualizzare i dati).
`axis('equal')`	usa la stessa scala sulle ascisse e le ordinate

`set, get`

Esempio

Esempio: Sia x un vettore di 10 elementi equidistanti e contenuti nell'intervallo [0,1]. Definire i vettori y = sin(2πx) e z = cos(2πx). Disegnarli sullo stesso grafico usando tratti diversi e colori diversi nel rettangolo [0 1]x[-1 1]. Etichettare gli assi rispettivamente con 'x' e 'f(x)' e dare il seguente titolo 'funzioni trigonometriche'.

```
>> x = linspace(0,1,10);
>> y = sin(2*pi*x);
>> z = cos(2*pi*x);
>> figure
>> plot(x,y,'r',x,z,'b:')
>> axis([0 1 -1 1])
>> xlabel('x')
>> ylabel('f(x)')
>> title('funzioni trigonometriche')
```



Sottografici

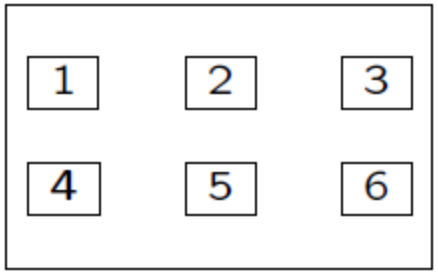
Spesso ci si pone il problema di disegnare diversi grafici separati in una stessa finestra. L'obiettivo può essere raggiunto facilmente utilizzando la funzione **subplot** la cui sintassi è

```
subplot(Righe, Colonne, Sottofinestra)
```

subplot(m, n, k) suddivide la finestra dei grafici in una matrice mxn e attiva la k-ma sottofinestra. Il successivo comando **plot** disegna il grafico nella finestra attivata

Esempio: **subplot(2,3,4)**

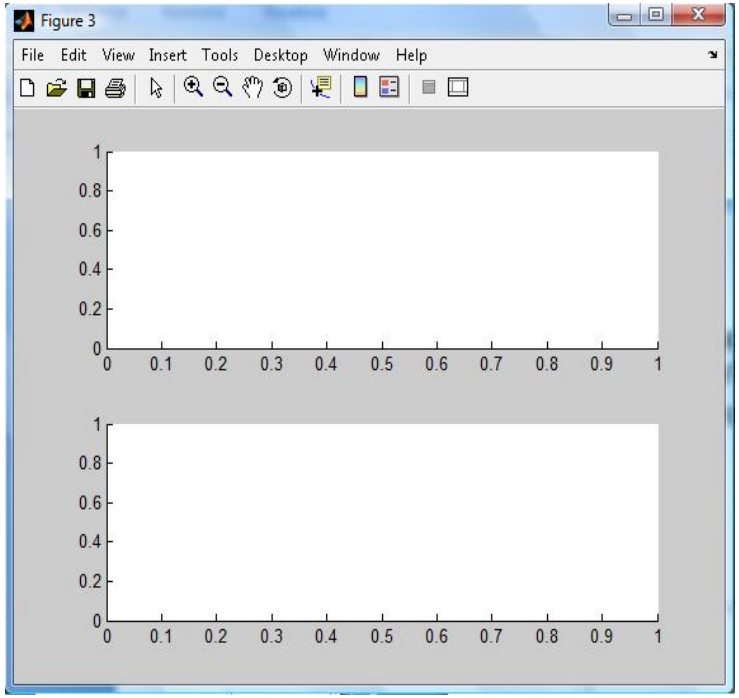
Tale istruzione suddivide la finestra in una matrice 2 × 3 di sottofinestre ed attiva la quarta sottofinestra grafica. Le sottofinestre sono numerate come segue



Sottografici

Esempio

```
>> subplot(2,1,1)
% divide la finestra grafica in 2
% sottofinestre poste una sotto
% l'altra e attiva la prima
>> plot(x1,y1);
>> title('grafico1')
>> subplot(2,1,2);
% attiva la seconda finestra
>>plot(x2,y2); title('grafico2')
```



fplot

```
fplot(funzione,limiti_assi,'opzioni')
```

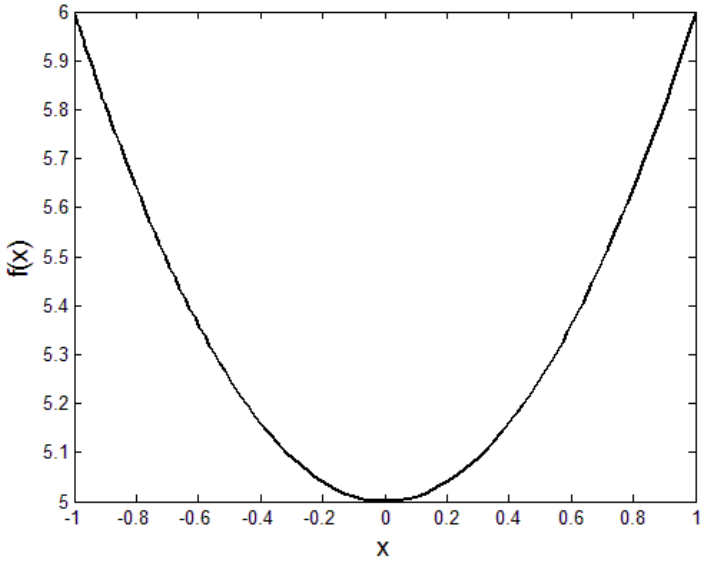
- **`funzione`** puntatore ad una funzione

```
>> f = @(x)[x^2+5]
    f =
        @(x)[x^2+5]
```

- **`limiti_assi : [xmin xmax]`** è un vettore contenente i limiti inf e sup dell'intervallo in cui si vuole visualizzare la funzione;

Esempio:

```
>> fplot(f,[-1 1])
>> xlabel('x'), ylabel('f(x)')
```



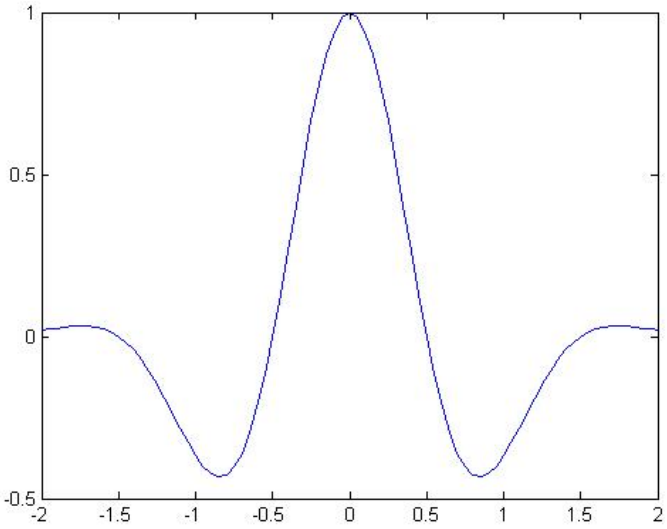
Max

- Trovare il max della funzione

$$f(x) = e^{-x^2}\cos(\pi x)$$

nell'intervallo [-2,2]

```
>> x = [-2:.05:2];
>> y = exp(-x.^2).*cos(pi*x);
>> plot(x,y)
>> [maximo p_max]= max(y);
```



Esercizi

- Disegnare la parabola $y = x^2+2x$ nell'intervallo [-10,10] e calcolarne le coordinate del vertice

- Disegnare tra 0 e 2π la funzione sin(x) e le sue approssimazioni di Maclaurin del 1° e 5° grado
  - le 3 curve devono essere rispettivamente a tratto continuo, tratteggiata e marcata con cerchietti

- Disegnare un cerchio

- Sia **`x=1:10`**, **`y1=atan(x)`** **`e`** **`y2=log(x)`**. Disegnare y1 e y2 marcandoli opportunamente e abbellire con varie etichette i grafici

Script, elementi di programmazione

Files .m

- Al posto di eseguire i comandi direttamente da linea di comando, possiamo memorizzare la successione dei comandi in un file di testo, salvarli e successivamente eseguirli; file di questo tipo sono detti **m-file** (files `.m`)
- Un file `.m` può essere generato con qualsiasi editor di testo ASCII (ad es. notepad di windows o l'editor di Matlab)

- Il **path** di Matlab è un insieme di directory sul computer locale in cui vengono cercate le funzioni o gli script che vengono chiamati dalla linea di comando
- Per visualizzare l'attuale path
  - dalla GUI di Matlab
  - dal prompt digitare il comando **`path`**

Path

- Se creiamo un nuovo file `.m` possiamo
  - salvare il file in una directory contenuta nel path
  - aggiungere la directory, in cui è salvato il file, al path
  - cambiare la directory di lavoro (Current Directory)

- **Per aggiungere una directory al path**
  - Menù **File** `->` **Set Path** apre il **Path Browser**
    - da Path Browser: selezionare la directory con il pulsante **Add Folder** (o **Add with subfolders**)
    - questa operazione è consigliata solo per inserire nuovi pacchetti o quando si vuole aggiungere una nuova funzione ai toolbox esistenti. Negli altri casi è consigliato lavorare nella directory di lavoro o sotto-directory
  - con il comando **`addpath('nome_dir')`**

Script e funzioni

- **Script files**: **`nome_file.m`**
  - eseguono una lista di istruzioni
  - non prevedono parametri di ingresso
  - utilizzano il workspace di MATLAB, **le variabili usate sono messe nella memoria di lavoro di MATLAB**
- **Funzioni: `nome_funzione.m`**
  - si possono passare parametri in ingresso ed ottenerne in uscita
  - sintassi **`function [y1,…,yn] = nome_funzione(x1,…,xn)`**
    - **`y1,…,yn`** -> parametri in uscita
    - **`x1,…,xn`** –> parametri in entrata
  - **le variabili usate all'interno sono locali**

Creazione di m-files

- **Per creare un nuovo m-file**
  - Menù **File** -> **New** ->**M-File**
- Per lanciare uno **script** (e quindi eseguire i comandi in esso contenuti)
  - **se siamo nella stessa directory** dove è salvato il file digitare il nome dello script nella linea di comando
  - **se siamo in una directory** diversa rsipetto a quella in cui è salvato lo script
    - digitare dal prompt **`run .../file.m`** **`...`** indica il percorso dalla cartella dove stiamo lavorando alla cartella in cui il file è salvato
    - è possibile cambiare la directory di lavoro utilizzando il comando **`cd`** (si digiti **`help cd`** per maggiori chiarimenti) oppure utilizzando le apposite icone nella barra dei comandi.

Script

- Tutte le variabili utilizzate nello script durante l'esecuzione dell' M-file vengono automaticamente messe nella memoria di lavoro di MATLAB
  - vedremo come questo non valga nel caso in cui si crei una funzione

- Per una minima manipolazione dei file su disco, MATLAB mette a disposizione alcuni comandi

```
dir, delete, cd, pwd, mkdir, copyfile
```

- **Esempio:** disegniamo una retta e una parabola

Esempio

- Creiamo il file disegna.m e scriviamo la lista di comandi

```
% Disegna una retta e una parabola
%
n = 5;
x = linspace(-n,n);
y1 = x;
y2 = x.^2+5*x+1;

figure, hold on
plot(x,y1)
plot(x,y2,'r')
```

- Per eseguire il file, dal prompt

```
>> disegna
```

N.B. Il file deve essere salvato nella directory di lavoro che è quella in cui ci troviamo!!!

Commenti

- Il carattere **%** serve per introdurre un commento all'interno dello script, MATLAB ignora il contenuto alla destra del carattere **%** fino alla linea successiva
  - **CTRL R**(**CTRL T**) per commentare (eliminare il commento da) una riga
- Il commento all'inizio dello script file è particolarmente importante in MATLAB, infatti richiamando il comando help seguito dal nome dello script otteniamo come risposta il commento inserito all'inizio dello script stesso

```
>> help disegna
disegna.m
Disegna una retta e una parabola
```

- Una caratteristica degli script è quella di non avere parametri in ingresso modificabili. Ad esempio se vogliamo modificare i valori di n dobbiamo modificare ogni volta lo script.

Programmare in MATLAB

- Anche gli algoritmi più semplici richiedono l'esecuzione ripetuta di istruzioni e l'esecuzione condizionata di alcune parti
- Esistono molti comandi per creare m-file complessi e versatili e strutture logiche simili a quelle usate nei linguaggi di programmazione
- La **costruzione ripetuta di blocchi di codice** in MATLAB viene eseguita tramite cicli. Esistono due diversi modi per realizzare cicli
  - il **ciclo incondizionato `for...end`**
  - il **ciclo condizionato `while...end`**
- Importante è l'uso degli **operatori relazionali**

Programmare in MATLAB

Alcune strutture di programmazione elementari

- **Operatori relazionali:** <, <=, >, >=, ==, =
- **Operatori logici:** & (and), / (or), ~ (not)
- **Cicli controllati da un contatore:** `for...end`
- **Cicli condizionati:** `while...end`
- **Strutture condizionali:** `if - elseif - else – end`
- **Uscita incondizionata:** `break`

Nota: tutte le strutture possono essere scritte su più righe oppure su un'unica riga separate da virgole

Operatori relazionali

Operatori relazionali

`< , <= , > , >= , == , ~=`

si usano per **confrontare** tra di loro gli elementi di 2 matrici; il risultato dell'operazione sarà

- 0 se la relazione è falsa
- 1 se la relazione è vera

Esempio:

```
>> 2==3
ans =
0

>> 2 ~ =3
ans =
1
```

Operatori logici

Operatori logici

& , | ,~

si usano per **combinare** tra loro gli operatori relazionali

Operatore	Significato
&	and
|	or
~	not
xor	or esclusivo

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

Esempio:

```
>> a = [0 0 1 1]; b = [0 1 0 1]
>> c = xor(a,b)
c =
    0 1 1 0
```

Osservazioni

- Sono **operazioni binarie** come la somma e il prodotto e danno luogo ad un risultato
  - la differenza è che il risultato può assumere solo 2 valori: 0 (se la relazione è falsa) e 1 (se la relazione è vera)
  - il risultato è una matrix di tipo **logical**
- Come visto dall'ultimo esempio, in MATLAB anche gli operatori logici e relazionali possono esssere applicati a vettori (matrici)
  - consentono di confrontare tra loro gli elementi di due vettori(matrici)
  - l'operatore relazionale è applicato ad ogni elemento delle matrici che si confrontano per cui le matrici devono avere le stesse dimensioni, oppure una delle due deve essere uno scalare
- Gli operatori logici e relazionali sono usati nei **test condizionali** (**`if…end`**, **`while…end`**), ma le matrici di tipo logical possono essere usate per selezionare elementi di una matrice

Osservazioni

Esempio:

```
>> x = [30 12 19 7 5]
>> x > 15
ans =
   1  0  1  0  0
>> x(x>15)
ans =
   30 19
```

- In alcune circostanze questa notazione compatta risulta particolarmente utile, ad esempio per individuare gli elementi non nulli di una matrice o di un vettore

Ciclo for…end

Ciclo incondizionato

```
for indice = m:p:n
    blocco di istruzioni
end
```

Ripete il blocco di istruzioni un numero fissato di volte

- `indice` contatore
- `m` valore iniziale del contatore
- `p` incremento del contatore (positivo o negativo)
- `n` valore finale del contatore

`m, p, n` possono essere variabili intere o reali

Nota: `indice = m:n` equivale a `indice = m:1:n`

Ciclo for…end - esempio

Consideriamo il semplice problema del calcolo del **valore medio** di un vettore o di una matrice. Dato un vettore x di n componenti il valore medio è definito come

$$m = \frac{1}{n} \sum_{i=1}^{n} x_i$$

In uno script

```
n=length(x);
somma=0;
for i=1:n
   somma=somma+x(i);
end
m=somma/n;
```

```
for i = n1:passo:n2

    blocco di istruzioni

end
```

Ad esempio per **x=[1 2 3 4 5 6]**

Ciclo for…end

- Nell'esempio di prima l'uso del ciclo for può essere evitato utilizzando la funzione MATLAB **sum** nella forma

```
somma=sum(x)
```

che assegna alla variabile somma la somma degli elementi del vettore x

- In MATLAB esiste la funzione predefinita per il calcolo della media di un vettore

```
help mean
```

Osservazioni

- Non è obbligatorio, ma è fortemente consigliato, scrivere il blocco di istruzioni all'interno del ciclo allineandolo con l'espressione che governa il ciclo. Questo modo di procedere consente di evidenziare bene le parti di codice che vengono eseguite nei cicli o sotto opportune condizioni

- È possibile **annidare** più cicli for

```
for i = n1:passo1:n2
    for j = m1:passo2:m2
        blocco di istruzioni
    end
end
```

Ciclo while…end

Ciclo condizionato

```
while condizione
    blocco di istruzioni
end
```

Esegue un blocco di istruzioni un numero indenito di volte fino al persistere di una certa condizione

condizione è un'espressione che MATLAB valuta numericamente e che viene interpretata come vera se diversa da zero

- in un contesto logico una variabile numerica restituisce 1 (vero) se la variabile è diversa da zero, altrimenti 0 (falso). Non è quindi noto a priori il numero di ripetizioni del blocco di istruzioni
- nel ciclo for si ripete il blocco d'istruzioni tot volte. In molte circostanze si ha la necessità di ripetere un certo numero di operazioni diverse volte a seconda che una certa condizione sia verificata oppure no. In questo caso si utilizza il costrutto while

Ciclo while…end - esempio

In uno script

```
n=length(x);
somma=0;
i=1;
while i<=n
   somma=somma+x(i);
end
m=somma/n;
```

```
while condizione
    blocco di istruzioni
end
```

Esegue il ciclo fino a quando i è minore uguale a n

Esempio: visualizzare i primi 3 numeri naturali

```
>>x=1;
>>while x~=4, disp(x), x=x+1; end
     1
     2
     3
```

Osservazioni

- Nel ciclo while si fa uso degli operatori relazionali
- L'operatore relazionale **==** che confronta il valore della variabile a sinistra dell'operatore con quello della variabile a destra, non va confuso con l'operatore di assegnazione **=** che invece assegna il valore della quantità alla destra dell'operatore alla variabile a sinistra dello stesso
- Qualora, a causa di un errore di programmazione, il programma dovesse ripetere un numero indefinito di volte il blocco di istruzioni perché la condizione di persistenza è sempre vericata, è possibile interrompere l'esecuzione del programma premendo contemporaneamente i tasti **Ctrl+C**

Osservazioni

Privilegiare operazioni vettoriali ai cicli `for…end` e `while…end`

- la caratteristica principale di MATLAB consiste nella possibilità di eseguire operazioni vettoriali. L'uso efficiente di MATLAB è strettamente legato alla capacità dell'utente di sfruttare tale caratteristica
- un ciclo di istruzioni di tipo **`for ...end o while ...end`** comporta l'esecuzione ripetuta in forma sequenziale di un blocco di istruzioni
- è buona regola prima di scrivere un ciclo cercare di vedere se è possibile evitarlo tramite un uso opportuno di istruzioni vettoriali
- MATLAB è un linguaggio interpretato e solo recentemente sono stati introdotti opportuni compilatori. L'efficienza e la velocità saranno quindi ridotte rispetto agli usuali programmi scritti in linguaggi ad alto livello, come il C o il Fortran, e compilati.

Comandi “utili”

- **`break`**
  - per uscire in maniera forzata da un ciclo
  - MATLAB salta automaticamente all’istruzione **end** che termina il ciclo
- **`return`**
  - interrompe l’esecuzione della funzione
  - svolge una funzione analoga a break, la differenza è che return interrompe l’esecuzione della funzione e ritorna al programma da cui tale funzione era stata chiamata

Esercizio

- Scrivere uno script **`fattoriale.m`** che calcola il fattoriale di un numero intero n
  - usare un ciclo **`for…end`**
  - eseguire lo script per diversi valori di n

Esercizio

- Generare un vettore **v** di lunghezza 20 con numeri compresi da 1 a 10
  - usare la funzione **`rand`**

- A partire da **v** costruire due nuovi vettori **a** e **b** contenenti rispettivamente gli elementi di **v** minori e maggiori di 5

Errori

Le sorgenti di errore possono essere, in generale, suddivise in cinque gruppi

- **Errori di impostazione del problema**
  - non sempre le formule matematiche rappresentano modelli esatti dei fenomeni reali; generalmente questi modelli sono più o meno idealizzati. Infatti, nello studio dei vari fenomeni della natura, siamo costretti, al fine di rendere più semplice il problema, ad ammettere certe condizioni e ciò implica una serie di errori.

- **Errori di metodo**
  - talvolta succede che è difficile, se non impossibile, risolvere un problema impostato in termini rigorosi. In questo caso il problema in esame viene sostituito da un altro problema approssimato i cui risultati si distinguono di poco dal problema dato

Errori

- **Errori di troncamento**
  - le funzioni che appaiono in formule matematiche sono spesso date in forma di successioni infinite o di serie e numerose equazioni matematiche non possono essere risolte che attraverso una descrizione di processi infiniti, i cui limiti sono appunto le soluzioni cercate. Siccome un processo infinito non può, in generale, terminare in un numero finito di passi, siamo costretti a interrompere la successione infinita di passi computazionali, necessaria per ottenere un risultato esatto.
- **Errori di rappresentazione dei dati o di Arrotondamento**
  - non tutti i numeri reali possono essere rappresentati in un calcolatore. Se il calcolatore utilizza una rappresentazione con t cifre per la mantissa, tutti i numeri reali compresi tra l'estremo inferiore e l'estremo superiore di F la cui mantissa ha un numero di cifre superiore a t dovranno in qualche modo essere arrotondati a t cifre.

Errori

- **Errori nei calcoli**
  - effettuando calcoli tra numeri macchina approssimati, gli errori di rappresentazione dei dati iniziali si propagano in qualche misura sui risultati di tali calcoli. L'entità della propagazione dipende anche dall'algoritmo utilizzato.

- L'analisi degli errori è fondamentale per comprendere come evitarli, o (qualora non fosse possibile evitarli) ridurli al minimo

Cancellazione numerica

- La cancellazione numerica è la perdita di cifre significative
- E' un fenomeno che si verifica durante l'operazione di **sottrazione** tra due numeri "quasi uguali"
  - se due numeri sono quasi uguali, dove uguali s'intende a meno della precisione macchina, allora è possibile il verificarsi della cancellazione numerica.
- Siano *x1 e x2* due numeri reali. Se *x=x1-x2* è "molto piccolo", l'errore relativo

$$\delta_x = \left| \frac{fl(x1 - x2) - x}{x} \right|$$

può essere molto grande e ciò produce una perdita di cifre significative nel calcolo di *fl(x1-x2)*

- E' sempre preferibile evitare la sottrazione tra numeri macchina "quasi uguali"

Esempio

- Consideriamo le due forme equivalenti di uno stesso polinomio:

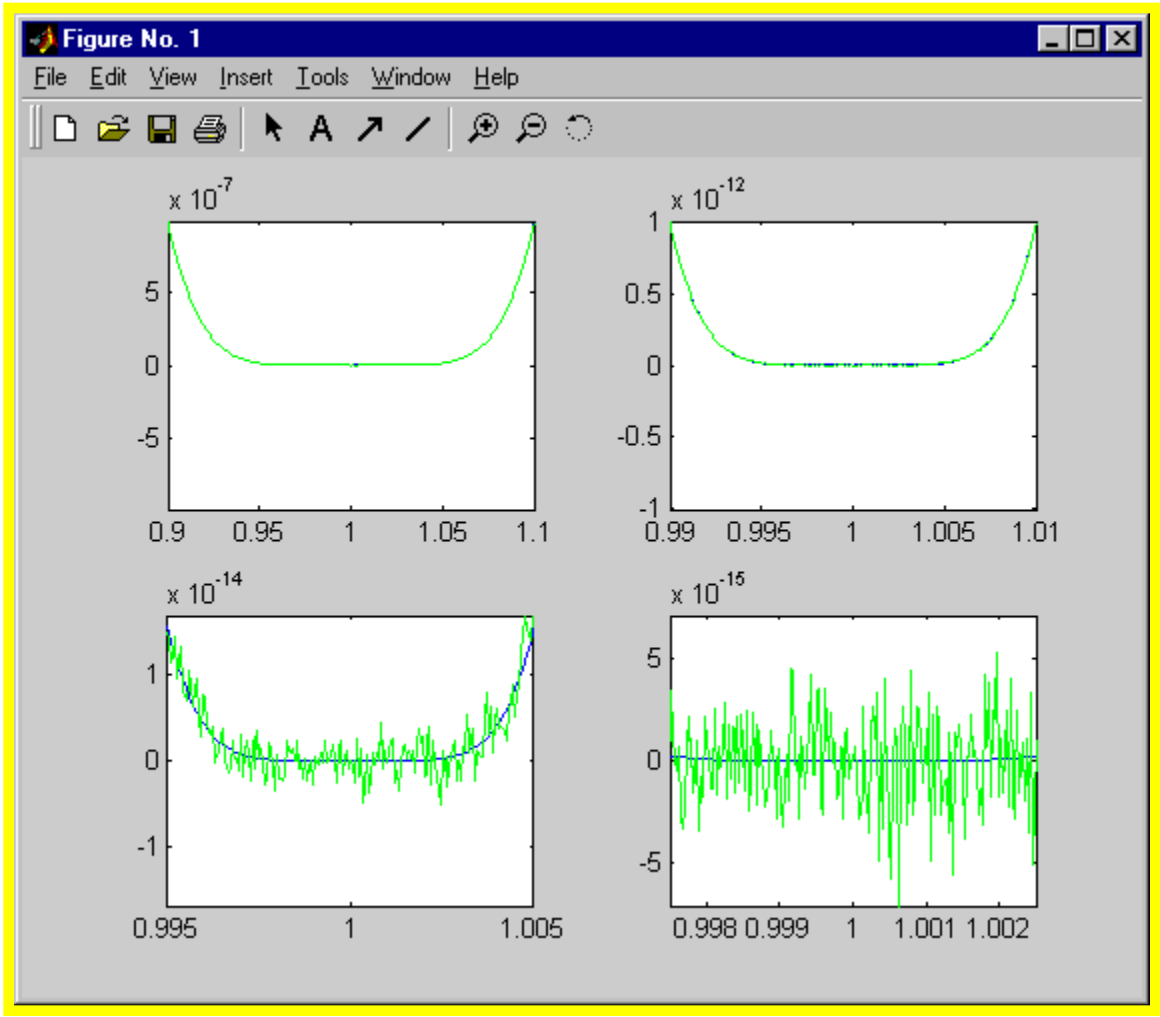
- $p(x) = (x-1)^6 =$
  $= x^6 - 6x^5 + 15x^4 - 20x^3 + 15x^2 - 6x + 1$

- Stabiliamo se, valutate al calcolatore in uno stesso punto, forniscono lo stesso risultato

Esempio di cancellazione numerica

```
%Calcola (1-x)^6 con le due formule:
%y1 = (1-x)^6
%y2 = 1-6x +15x^2 -20x^3 +15x^4 -6x^5 +x^6
%e confronta y1 e y2 in un intorno di uno
%
k = 0
for delta = [0.1, 0.01, 0.005, 0.0025]
  h = delta/100;
  x = 1-delta:h:1+delta;
  y1 = (1-x).^6;
  y2 = 1 -6*x +15*x.^2 -20*x.^3 +15*x.^4 -6*x.^5 + x.^6;
  k = k+1;
  subplot(2,2,k)
     plot(x,y1)
     hold
     plot(x,y2,'g')
     axis([1-delta 1+delta -max(abs(y2)) max(abs(y2)) ])
end
```

L'output dello script precedente è:



Esercizi

- Consideriamo per ogni $k \in \mathrm{R}$ la funzione seguente

$$f(x) = kx^3 + \sqrt{k}\,x$$

Disegnare il grafico di $f(x)$ nell'intervallo [0,2] e trovare il punto di minimo (nell'intervallo [0,2]) con un'approssimazione di 0.02

- Come evitare il problema della cancellazione numerica nel calcolo delle radice

$$\sqrt{x+\delta} - \sqrt{x}, \quad \delta \to 0$$