Laboratorio di Calcolo Numerico

Federica Pes

Università degli Studi di Cagliari

Laurea Magistrale in Ingegneria Ambientale per lo Sviluppo Sostenibile

A.A. 2024-2025

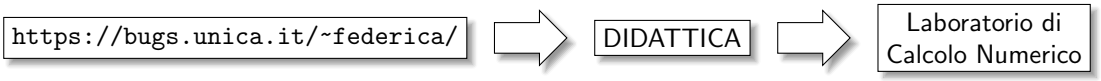
Informazioni

- 30 ore, 3 CFU
- Lezioni: martedì 08:00 - 11:00
- Ricevimento su appuntamento `federica.pes@unica.it`

Materiale didattico e informazioni disponibili nel sito

https://bugs.unica.it/~federica/LabCalcNum2425.html

oppure



- Modalità d'esame: tesina + discussione orale (singolarmente o in gruppo di max 3 persone)

Installare Matlab

Licenza Campus (durata annuale, rinnovabile) ottenibile usando le credenziali Esse3

Istruzioni per scaricare Matlab:
Cercare su Google *matlab unica* $\longrightarrow$ `https://web.unica.it/unica/it/studenti_s08_ss09.page`
Guida più completa $\longrightarrow$ `https://people.unica.it/idem/matlab-via-fedauthn/`

Materiale:

- Il programma contiene un manuale in linea molto completo (`help`)
- Su Internet si trovano manuali e dispense

Esistono cloni Open Source: Octave, Scilab

Introduzione

Il nome MATLAB è acronimo di MATRIX LABORATORY.

La principale caratteristica è che non opera con numeri, ma con matrici: i vettori e i numeri sono considerati casi particolari di matrici (numeri = matrici 1×1; vettori = matrici $n \times 1$ o $1 \times n$)

- Ambiente integrato per il calcolo scientifico[1] e la visualizzazione grafica
- Interfaccia verso librerie scientifiche di pubblico dominio (BLAS, Lapack, FFTW, etc.)
- Tutte le subroutines di calcolo sono documentate
- Implementazione e debugging veloci
- Utilizzato per didattica, ricerca e sviluppo

[1] Calcolo scientifico: si occupa di sviluppare, implementare e analizzare algoritmi numerici da impegare per risolvere modelli matematici.

Analisi Numerica

L'Analisi Numerica è lo studio degli algoritmi
per i problemi della matematica del continuo.

- Caratteristiche di un algoritmo:
  - stabilità (propagazione degli errori)
  - complessità computazionale (numero di operazioni richieste, è proporzionale al tempo di calcolo)
  - occupazione di memoria
- Ulteriori insidie possono essere nascoste nel problema da risolvere:
  - buona o cattiva posizione (esiste una e una sola soluzione?)
  - corrispondenza col problema fisico reale (i risultati ottenuti sono significativi?)
  - condizionamento (amplificazione degli errori sui dati, a prescidere dall'algoritmo utilizzato)

Storia

MATLAB fu creato negli anni '70 da Cleve Moler per permettere ai suoi studenti di fare calcolo matriciale senza dover conoscere il Fortran.

La prima versione commerciale risale al 1984.

Nuovi aggiornamenti vengono rilasciati ogni sei mesi circa.
La versione più recente è MATLAB R2024b

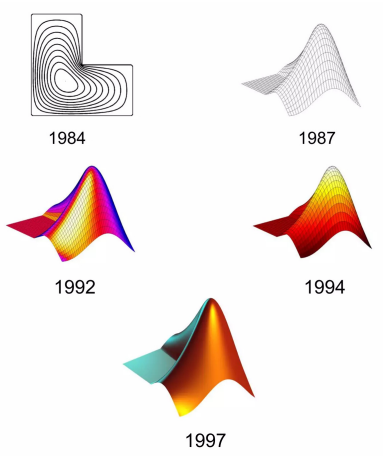
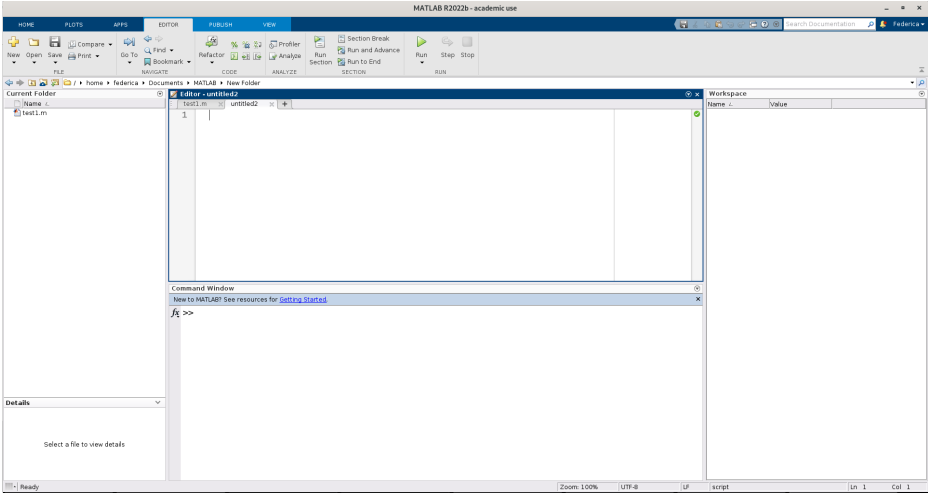


Figura: Immagine tratta da *C. Moler and J. Little, A History of MATLAB, 2020*

Introduzione

L'interfaccia principale è composta da diverse finestre. Le principali sono:
Current folder Command window Workspace Editor



Introduzione. Hello World!

```c
#include <stdio.h>

int main() {
  printf("Hello World\n");
  return 0;
}
```

Figura: Tipicamente, il primo programma scritto quando si impara un linguaggio di programmazione

- In Matlab, semplicemente, `disp('Hello world!')`

Operazioni su scalari e variabili

- Usiamo Matlab come calcolatrice: scriviamo, ad esempio, `2+3` e premiamo enter/invio
  Il risultato viene memorizzato in una variabile `ans` (answer)
- Assegniamo il valore dell'espressione precedente ad una variabile `x`: scriviamo `x=2+3`
- Se scriviamo `x=2+3;` il "`;`" alla fine dell'espressione sopprime la visualizzazione del risultato
- Se vogliamo visualizzare il valore della variabile dopo averla definita, basta scrivere `x`
- Una variabile può essere sovrascritta `x=2*3;` perdendo il suo valore iniziale

I nomi delle variabili devono rispettare le seguenti regole:

- possono contenere solo lettere, numeri ed il segno _ (underscore)
- in prima posizione deve esserci sempre una lettera
- MATLAB è *case sensitive* (ossia distingue fra maiuscole e minuscole, per cui le variabili `X` e `x` risultano fra loro diverse)
- Alcune parole non possono essere usate come nome di variabile: for, end, if, elseif, else, while, function, return, switch, case, otherwise, continue, break, ...

Operazioni su scalari e variabili

- Editing della linea di comando: è possibile recuperare i comandi precedentemente digitati utilizzando i tasti $\uparrow$, $\downarrow$ e modificarli
- Prendere appunti con la funzione `diary`: permette di salvare una sessione di lavoro
  - `diary` *`nomefile`*, `diary off`
    `diary Laboratorio1.txt`, ..., `diary off`
    `type Laboratorio1.txt` permette di visualizzare tutti i comandi memorizzati nel file appena creato

- Operazioni aritmetiche tra scalari $+, -, *$ e $/$
- Potenze `^` , radici `sqrt()`, funzioni trascendenti `sin()`, `cos()`, `tan()`, `log()`, etc.
  - `y=sqrt(100)` , `z=7^3` , `cos(z), sin(pi), arcsin(1), tan(2*pi),`
- Controllo dell'output: `;`
- Variabili predichiarate: `pi` greco, `i` (unità immaginaria), `eps` (precisione macchina `2.2204e-16`)

Operazioni: priorità operatori aritmetici

Scala priorità:

^	potenza
$*$ /	prodotto e divisione
$+$ $-$	somma e differenza

- 2^3 $*$ 4 $= 8 * 4 = 32$ $\qquad 2 + 8/4 = 2 + 2 = 4$

Uso delle parentesi tonde per variare la priorità degli operatori aritmetici

- 2^(3 $*$ 4) $= 2^{12} = 4096$ $\qquad (2 + 8)/4 = 10/4 = 2.5$ $\qquad 8/(2 + 2) = 8/4 = 2$

Le uniche parentesi che possono essere utilizzate per variare la priorità degli operatori aritmetici sono le parentesi tonde. Come si scrive in Matlab la formula seguente?

$$x = -2\left\{\frac{3}{2} + 6\left[(2 + 3)(4 - 5) + 1\right]\right\}$$

```
x=-2*(3/2+6*((2+3)*(4-5)+1))
```

Workspace

La finestra Workspace contiene la lista della variabili e le seguenti informazioni:

- Name: nome della variabile.
- Value: valore assegnato alla variabile.
- Size: dimensione (righe per colonne).
- Bytes: occupazione di memoria in termini di bytes.
- Class: il tipo di variabile char, double, sparse, struct, ...

Di default, Matlab lavora con variabili in doppia precisione'. Ogni numero memorizzato in doppia precisione occupa 8 Bytes.

Una volta creata una variabile rimane nell'ambiente di Matlab e può essere utilizzata in qualsiasi momento, fino a che non viene espressamente cancellata con il comando `clear`.

Variabili e controllo del workspace

Comandi utili per gestire le variabili:

- `who` permette di sapere quali sono le variabili definite dell'ambiente
- `whos` mostra le variabili, il tipo, la dimensione e l'occupazione di memoria
- `clear` *`variabile`* elimina la variabile menzionata
- `clear` elimina tutte le variabili
- `clc` pulisce lo schermo e non elimina le variabili

Memorizzazione e caricamento dati:

- `save` *`nomefile variabili`* salva le variabili menzionate nel file nomefile.mat
- `save` *`nomefile`* salva tutte le variabili nel file nomefile.mat
- `load` *`nomefile`* carica nell'ambiente le variabili salvate precedentemente nel file nomefile.mat

Formato di visualizzazione dei numeri

Tutti i calcoli vengono effettuati in doppia precisione, mentre diversa è la visualizzazione delle variabili che viene determinata con il comando `format`

Vediamo come viene visualizzato il numero `a = 3.123456789`

- `format short`: 4 cifre decimali (con arrotondamento) `3.1235`
- `format long`: 15 cifre decimali `3.123456789000000`
- `format shortE`: notazione esponenziale `3.1235e+00`
- `format longE`: notazione esponenziale `3.123456789000000e+00`
- `format rational`: `253/81`
- ⋮

Notazione esponenziale

$2.7 \times 10^4 \rightarrow$ scriviamo `2.7e4` $\qquad 1.5 \times 10^{-3} \rightarrow$ scriviamo `1.5e-3`

Numeri complessi

Le variabili `i` e `j` sono predefinite e rappresentano entrambe l'unità immaginaria $\sqrt{-1}$, `0+1.000i` (se vengono riassegnate perdono il loro valore di default!)

`z = 5 + 2i`
`real(z)` mostra la parte reale del numero complesso z
`imag(z)` mostra la parte immaginaria del numero complesso z
`abs(z)` calcola la norma di z
`angle(z)` calcola la fase di z

Vettori e matrici

- Definizione estensiva di vettori e matrici (parentesi quadre):
  - vettore riga `x = [ 6 3 -6 21 0 -10 ]` , `x = [ 6, 3, -6, 21, 0, -10 ]`
  - vettore colonna `x = [ 6; 3; -6; 21; 0; -10 ]` , `x = [ 6 3 -6 21 0 -10 ]'`
    Il simbolo `'` indica la trasposizione.
  - matrice 2×3 `A = [ 2 -3 12 ; 21 -43 0 ]`

- Definizione intensiva di vettori: l'operatore *colon* (`:`) (parentesi quadre)
  - `y = [1:5]` crea il vettore riga $y = [1\ 2\ 3\ 4\ 5]$ `y = [1:4]'` crea il vettore colonna $y = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$
  - Il comando `v=[c:h:d]` genera un vettore riga in cui la prima componente ha valore c e le successive si incrementano di h fino ad arrivare a d
    `y = [1:2:9]` crea il vettore riga $y = [1\ 3\ 5\ 7\ 9]$ `y = [2:2:20]'` crea il vettore colonna ...
    `w = [1:.5:3]` crea il vettore riga $w = [1\ 1.5\ 2\ 2.5\ 3]$

- Lo spazio o la virgola separano elementi sulla stessa riga. Il punto e virgola separa le righe.

Vettori e matrici

- Funzioni che generano arrays: `ones`, `zeros`, `linspace`, `magic`, `rand`, `randn`, `eye`, `diag`
  - `linspace(a,b,n)` genera un vettore riga di lunghezza n le cui componenti sono n punti equispaziati tra a e b (estremi compresi). `v = linspace(0,5,4)`
  - Verificare che `magic(7)` sia un quadrato magico
- *Patchwork* (concatenazione) di arrays. Il vettore nullo (`[]`)
  - `z = [ x; y ]` `z = [ y x ]`
    i vettori devono avere dimensioni concordi per cui sia possibile concatenarli.
- Accesso a singoli elementi e a sottoarrays (e loro modifica) (parentesi tonde)
  - L'indicizzazione inizia da 1. La componente `y(0)` non esiste!!!
  - `y(3)` per accedere ad un elemento di un vettore `ans = 6`
  - `y(end)` per accedere all'ultimo elemento di un vettore
  - `y(2) = 1` per modificare un elemento di un vettore
  - `A(1, 3)` per accedere all'elemento $A_{1,3}$ della matrice `ans = 12`
  - `A(2, 1) = 10` per modificare l'elemento $A_{2,1}$ della matrice
  - `A(1, :)` per accedere alla prima riga della matrice A
  - `A(:, 2)` per accedere alla seconda colonna della matrice A
  - `A(1:2, 1:2)` per accedere ad un blocco della matrice A

Vettori e matrici: operazioni

`+` somma di vettori o matrici (elemento per elemento)
`-` differenza di vettori o matrici (elemento per elemento)
`*` prodotto tra vettori e/o matrici (righe per colonne)
`.*` prodotto tra vettori e/o matrici (elemento per elemento)
`.^` elevamento a potenza (elemento per elemento)

- Le dimensioni dei vettori/matrici devono essere concordi per cui le operazioni possano essere eseguite.

- Definiamo `x = [1 2 3 4]'` `y = [5 6 7 8]'`
  `A = [1 2 3 4; 5 6 7 8; 9 10 11 12]` `B = randn(3,4)`

  e calcoliamo `x+y`, `5*x`, `5*A`, `x'*y`, `x*y'`, `B*x`, `x'*A`, `A+B`, `A'*B`
  `x.*y`, `x.^2`, `A.*B`, `A.^3`

Documentazione

`help` *`comando`* visualizza una descrizione breve e concisa sul comando. Per esempio:
`help zeros`, `help eye`, `help norm`, `help eig`, `help plot`, `help min`, `help max`, `help lu`, `help rand`, ...

`doc` apre una finestra con il manuale

Vettori e matrici: operazioni

- Per conoscere lunghezza di un vettore o dimensione di un array `length`, `size`
- Determinante di una matrice quadrata `det`
- Inversa di una matrice `inv`
- Norme di vettori e matrici `norm`
- Funzioni statistiche: `sum`, `mean`, `std`, `min`, `max`
- Ottimizzazione del codice: preallocazione di un array
  - Valutazione del tempo di calcolo
- Altri tipi di variabili: `struct` e `cell`

Sorgenti di errore

I numeri su cui si opera sono tipicamente affetti da errori

- Dati sperimentali
- Semplificazioni introdotte nel modello matematico
- Aritmetica di macchina (errori di arrotondamento e/o loro propagazione nei calcoli), *smearing*, *overflow* e *underflow*.

Conseguenze

1. Tutti i numeri sono approssimati
2. I computer "sbagliano" (commettono errori nei calcoli) sistematicamente

Esempi sui numeri di macchina

- $\sqrt{2^2} - 2$
- $(\sqrt{2})^2 - 2$
- $3\left(\frac{4}{3} - 1\right) - 1$
  - il risultato è 0?

- $1 + \frac{\texttt{eps}}{2}$
  - `eps/2` è troppo piccolo: *smearing*

- 10^{400}
  - `Inf`: il numero è troppo grande: *overflow*

- $v = \begin{bmatrix} 11^{\pm 200} \\ -11^{\pm 200} \end{bmatrix}$
  - $\sqrt{v_1^2 + v_2^2} =$ `Inf` *overflow*
  - `norm(v)` restituisce il risultato

- Operazioni "proibite": `Inf` e `NaN`, infinito e risultato numerico non definito (not-a-number)

- Propagazione degli errori

  $\frac{1-\cos x}{x^2}$ vs. $\frac{1}{2}\left(\frac{\sin\frac{x}{2}}{\frac{x}{2}}\right)^2$ per $x \to 0$
  - Il risultato è $1/2$. Cosa si ottiene in matlab?

Programmazione mediante script

- Uso dell'editor e creazione di uno *script* (è un file .m)
- Le righe di commento iniziano con `%`, help in linea
  - Ogni script ben scritto generalmente inizia con dei commenti che ne presentano il contenuto. Prima dell'inizio dei comandi, si lascia una riga vuota.
    Dalla finestra si può vedere il commento digitando `help` *`nomefile`*.
- Controllo dell'iterazione: `while` e `for`
- Istruzioni condizionali: `if` e `switch`
- Operatori e variabili logiche
- Le istruzioni `input`, `disp`, `error` e `fprintf`
- Esempi

Programmazione in Matlab: cicli for e while

Un ciclo o loop è una struttura che consente di ripetere una o più istruzioni più volte.

- Ciclo `for` esegue delle operazioni un numero fissato di volte.

```
for k = 1:n
  comandi
end
```

- Ciclo `while` esegue delle operazioni fino a che una certa condizione di controllo rimane vera. La condizione di controllo è un'espressione logica.

```
while espressione logica
  comandi
end
```

Programmazione in Matlab: if, elseif, else

Molto spesso capita che determinate operazioni si devono fare solo se una determinata condizione è verificata. Per fare ciò si usano delle espressioni logiche di controllo:

```
if espressione logica
  comandi
end
```

```
if espressione logica
  comandi
else
  comandi
end
```

```
if espressione logica
  comandi
elseif espressione logica
  comandi
else
  comandi
end
```

Operatori relazionali

$<$	minore di
$>$	maggiore di
$<=$	minore di o uguale a
$>=$	maggiore di o ugualea
$==$	uguale a
$\sim=$	diverso da

Esempio:

```
x=3; y=6;
x<y     x==y     x~=y
```

Matlab restituirà 0 se il confronto è falso oppure 1 se il confronto è vero.

Operatori logici

Un'espressione logica può essere ottenuta come unione di più condizioni.

&&	AND	L'espressione è vera se sono vere entrambe le condizioni, mentre è falsa se almeno una condizione è falsa.
||	OR	L'espressione è vera se almeno una delle condizioni è vera, mentre è falsa se entrambe le condizioni sono false
$\sim$	NOT	Nega la condizione. Se la condizione è vera, la rende falsa e viceversa.

Esempio:

```
if condizione1 && condizione2
  comandi
end
```

Programmazione in Matlab: switch case

```
switch espressione
  case 1
    comandi
  case 2
    comandi
  case 3
    comandi
  otherwise
    comandi
end
```

Valuta un'espressione e decide quali operazioni eseguire in base al *case*.

Ottimizzazione del codice: preallocazione di un array

```
x = zeros(100,1)
A = zeros(200,100)
```

Valutiamo il tempo di un calcolo senza preallocare e preallocando un array.

Valutazione del tempo di calcolo

Per valutare l'efficienza di un programma in termini di tempo d'esecuzione espresso in secondi, si possono utilizzare i comandi `tic` e `toc`.

`tic` start orologio
  calcoli
`toc` tempo trascorso a partire dal tic (in secondi)

Gestione input

Il comando *variabile* = input('*stringa*') serve per assegnare interattivamente un dato da tastiera, quindi attende un dato in ingresso, dopo aver visualizzato la stringa di caratteri *stringa*, e lo assegna a *variabile*.

- Utilizzabile per assegnare una sola variabile.
- Durante l'esecuzione di un programma, questa istruzione interrompe l'esecuzione fino a che non viene introdotto da tastiera il valore che si vuole assegnare alla variabile.
- Nell'angolo in basso a sinistra della finestra di Matlab appare la scritta `Waiting for input`.

Esempio: `a = input('Inserisci un numero tra 0 e 100 ')`

Gestione output

Il comando `disp` (display) può essere utile per visualizzare a schermo un messaggio.

Esempio: `disp('Script che risolve equazioni di secondo grado.')`

Il comando `error` segnala la presenza di un errore e interrompe l'esecuzione.

Esempio: `error('Non puoi dividere per zero!')`

Il comando `warning` visualizza un messaggio di warning, senza interrompere l'esecuzione.

Esempio: `warning('Il metodo iterativo ha superato il numero massimo di iterazioni')`

Gestione output

Per visualizzare dati di output con un certo formato si utilizza il comando `fprintf`

```
fprintf('testo e formato variabili', variabili)
```

- `%d` formato per numeri interi
- `%f` formato fixed point per numeri reali
- `%e` formato esponenziale per numeri reali
- `%g` seleziona il formato per numeri interi, fixed point o esponenziali
- `\n` inserisce un ritorno a capo, `\t` inserisce uno spazio di tabulazione

Esempio:

```
n=4; x=12.34567; err=1.3456e-4;
fprintf('Iter = %d \t Valore x = %f \t Errore = %e \n', n, x, err)
Iter = 4    Valore x = 12.345670    Errore = 1.345600e-04
```

Per decidere quante cifre decimali visualizzare, bisogna aggiungere un punto e il numero di cifre desiderate

```
fprintf('Iter = %d \t Valore x = %.4f \t Errore = %.2e \n', n, x, err)
Iter = 4    Valore x = 12.3457    Errore = 1.35e-04
```

Game of Life[2]

Il Gioco della vita è un automa cellulare sviluppato dal matematico John Conway alla fine degli anni '60. Il suo scopo è quello di mostrare come comportamenti simili alla vita possano emergere da regole semplici e interazioni a molti corpi, principio che è alla base dell'ecobiologia.

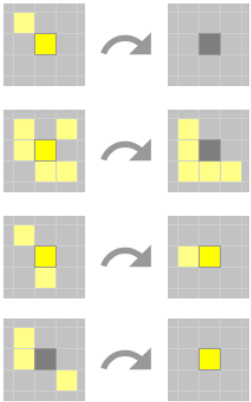
- Si tratta di un gioco senza giocatori: la sua evoluzione è determinata dal suo stato iniziale.
- Si svolge su una griglia di caselle quadrate (celle) che si estende all'infinito in tutte le direzioni. Ogni cella ha 8 vicini (le celle ad essa adiacenti).
- Ogni cella può trovarsi in due stati: viva o morta (o accesa e spenta). Lo stato della griglia evolve in intervalli di tempo discreti. Gli stati di tutte le celle in un dato istante sono usati per calcolare lo stato delle celle all'istante successivo. Tutte le celle vengono quindi aggiornate simultaneamente nel passaggio da un istante a quello successivo: passa così una generazione.

[2] https://it.wikipedia.org/wiki/Gioco_della_vita

Game of Life[3]

Le transizioni dipendono unicamente dallo stato delle celle vicine:

- Qualsiasi cella viva con meno di due celle vive adiacenti muore, come per effetto d'isolamento;
- Qualsiasi cella viva con più di tre celle vive adiacenti muore, come per effetto di sovrappopolazione;
- Qualsiasi cella viva con due o tre celle vive adiacenti sopravvive alla generazione successiva;
- Qualsiasi cella morta con esattamente tre celle vive adiacenti diventa una cella viva, come per effetto di riproduzione.



[3] https://playgameoflife.com/

Game of Life. Implementazione

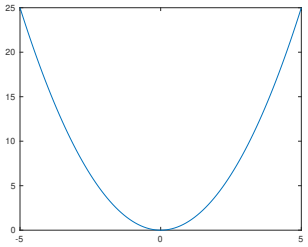
1. Creazione dello stato iniziale: creiamo la griglia come matrice quadrata e ogni elemento della matrice sarà uguale a 0 o a 1 se la cella è morta o viva, rispettivamente
2. Transizione: si parte dalla matrice dello stato precedente e se ne crea una nuova seguendo le regole (vedi slide precedente)
3. Visualizzazione dello stato: ad ogni transizione visualizziamo la matrice con il comando `imagesc(A)`

Grafico di una funzione

- Operatori che agiscono componente per componente: `+`, `-`, `.*`, `./` e `.^`

- Applicazione: grafico di una funzione
  1. campionamento della variabile indipendente
  2. calcolo della funzione sui punti di campionamento
  3. tracciamento del grafico della funzione x^2 nell'intervallo $x \in [-5, 5]$

```
x = [-5:.1:5]';
y = x.^2;
plot(x,y)
```



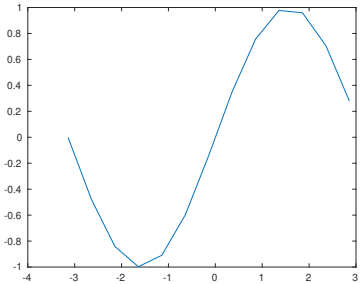
Si possono aggiungere titolo e label negli assi e si può modificare stile e colore della linea.

`title('grafico')`
`xlabel('x')`
`ylabel('y')`
`plot(x,y,'r')` linea rossa
`plot(x,y,'--r')` linea rossa tratteggiata

Grafico di una funzione

Il plot crea una spezzata che congiunge i punti (x_i, y_i). Diminuendo il passo, ovvero aumentando il numero dei punti, i segmenti di raccordo sono così piccoli da avere l'impressione di una linea continua.

```
x = [-pi:0.5:pi]';
y = sin(x);
plot(x,y)
```




```
x = [-pi:0.1:pi]';
y = sin(x);
plot(x,y)
```

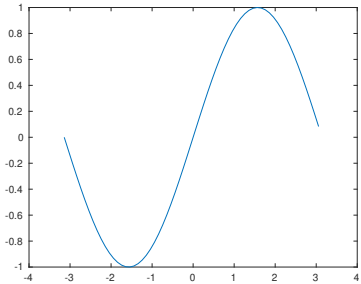


Grafico di una funzione

`plot(vettorex, vettorey, 'opzioni')`

Line Style Specifiers

Specifier	Line Style
-	solid line (default)
--	dashed line
:	dotted line
-.	dash-dot line

Color Specifiers

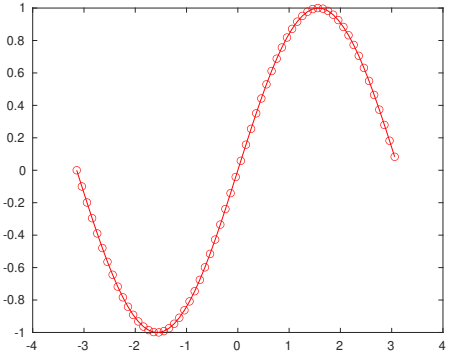
Specifier	Color
r	red
g	green
b	blue
c	cyan
m	magenta
y	yellow
k	black
w	white

Marker Specifiers

Specifier	Marker Type
+	plus sign
o	circle
*	asterisk
.	point
x	cross
s	square
d	diamond
^	upward pointing triangle
v	downward pointing triangle
>	right pointing triangle
<	left pointing triangle
p	five-pointed star (pentagram)
h	six-pointed star (hexagram)

Grafico di una funzione

```
x = [-pi:.1:pi]'
y = sin(x);
plot(x,y,'-or')
```



linea '-' continua
marcatore 'o' cerchi
colore 'r' rosso

L'opzione 'linewidth' seguita da un numero modifica lo spessore della linea

```
plot(x,y,'--r','linewidth',2)
```

Grafico di una funzione

Per costruire un vettore di punti x_i si può usare anche il comando `linspace`, che crea un vettore di n punti equispaziati in un intervallo $[a, b]$

```
x = linspace(a,b,n)'
```

Il vettore ha componenti

$$x_i = a + (i-1)\frac{b-a}{n-1}, \qquad i = 1, \ldots, n$$

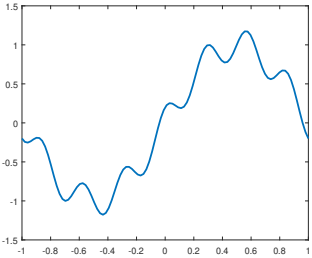
Esempio: `x = linspace(-1,1,101)'`

Esercizio: fare il grafico delle seguenti funzioni definite nell'intervallo $[-1, 1]$

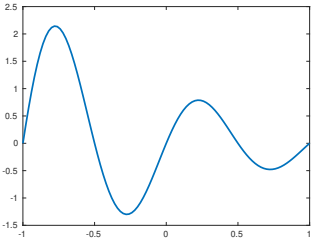
$$\sin(\pi x) + \frac{1}{5}\cos(7\pi x), \qquad e^{-x}\sin(2\pi x), \qquad \frac{\sin(10x)}{10x}$$

Grafico di una funzione

$$\sin(\pi x) + \frac{1}{5}\cos(7\pi x),$$



$$e^{-x}\sin(2\pi x),$$



$$\frac{\sin(10x)}{10x}$$

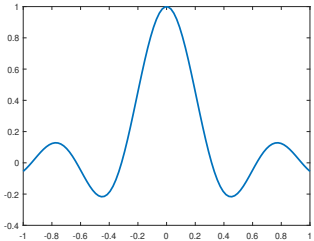


Grafico di più funzioni

```
x = linspace(-pi,pi,101)';
y1 = sin(x);
y2 = cos(x);
plot(x,y1,'b',x,y2,'--r')
legend('sin(x)', 'cos(x)')
```

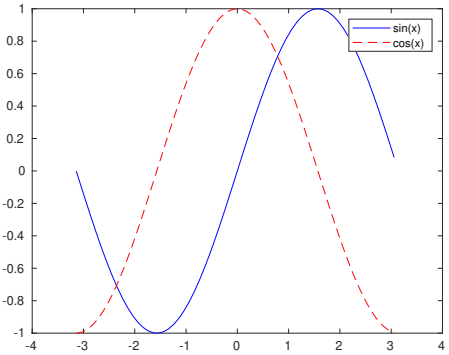
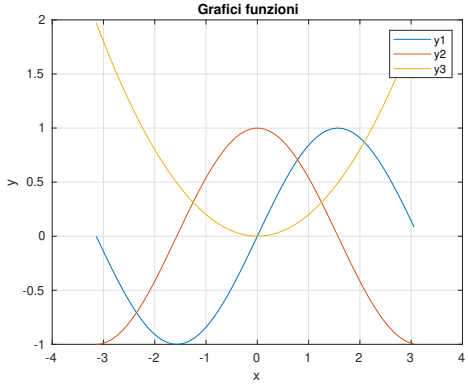


Grafico di più funzioni, hold on, hold off

```
x = linspace(-pi,pi,101)';
y1 = sin(x);
y2 = cos(x);
y3 = (1/5)*x.^2;

plot(x,y1)
hold on
plot(x,y2)
plot(x,y3)
hold off
legend('y1', 'y2', 'y3')
xlabel('x')
ylabel('y')
title('Grafici funzioni')
grid on
```



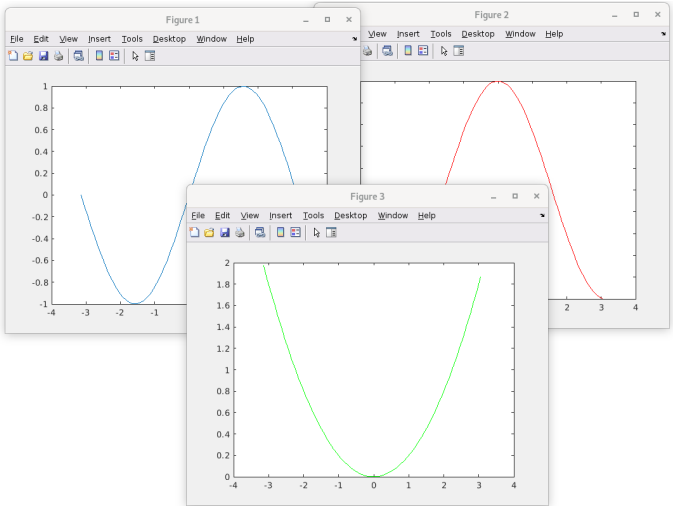
Grafici in più finestre, figure(...)

```
x = [-pi:.1:pi]';
y1 = sin(x);
y2 = cos(x);
y3 = (1/5)*x.^2;

figure(1)
plot(x,y1)

figure(2)
plot(x,y2,'r')

figure(3)
plot(x,y3,'g')
```



Istruzioni grafiche

- Visualizzazione contemporanea di più serie di dati
  - `hold on`, `hold off`
- Apertura di più finestre grafiche
  - `figure(1)`, `figure(2)`, etc
- Modifica dello stile delle linee
- Annotazioni: `title`, `legend`, `xlabel` e `ylabel`
- Altri tipi di grafici: `bar`, `stairs`, `stem`, `piechart`, `histogram`

Funzioni anonime

Assegnata una funzione del tipo $f(x)$, vogliamo valutare i valori assunti da f per diversi valori di x. Per fare ciò, usiamo le *funzioni anonime*: definiamo la funzione una volta per tutte senza dover riscrivere la sua espressione ogni volta che la si vuole valutare in punti differenti.

```
f = @(argomenti) espressione
```

Gli "argomenti" sono le variabili da cui dipende f.

```
f = @(x) 3*x.^2.*atan(x)
g = @(x,y) sqrt(x.^2+y.^2)
```

Usiamo le operazioni puntuali `.*` e `.^` perché vogliamo valutare le funzioni anche in vettori.

Programmazione in Matlab: script e function

I files .m possono essere di due tipi:

Script: Utile per automatizzare una serie di istruzioni che si devono eseguire più volte.

- Sono definiti semplicemente da una serie di comandi;
- Opera sui dati presenti nel Workspace;
- Non accetta variabili in input;
- Non ha variabili di output;
- Per eseguirlo è necessario digitare il nome del file nel prompt del Matlab.

Function: Sono degli m-file la cui prima riga deve essere della forma

```
function [x,y] = nomefunction(a,b)
```

- Le variabili interne sono locali;
- Può accettare variabili in input `(a,b)`;
- Può avere variabili in output `[x,y]`;
- Per eseguirlo è necessario copiare questa prima riga all'interno del prompt, ad eccezione della parola function.

Osservazione: una function può essere chiamata all'interno di uno script.

Programmazione in Matlab: function

Perché sono utili le **function**?

- *Riusabilità*: scriviamo una sola volta codice che può essere utilizzato spesso e/o in futuro.
- *Leggibilità*: incapsulo porzioni di codice in un file separato.

Importante! Il nome del file .m in cui viene salvata una function deve avere lo stesso nome della function. Deve iniziare con una lettera dell'alfabeto, può contenere numeri e underscore, MA non può contenere spazi.

Esercizio. Creare una function per il metodo di Newton. Creare uno script che contiene gli ingredienti del problema (equazione non lineare, derivata, punto iniziale, ecc), chiama la function newton, visualizza il risultato.

script `eqnonlin.m`

```
fun = @(x) ...
der = @(x) ...
x0 = ...
tol = 1e-6;
kMax = 100;
[x,k] = newton(fun,der,x0,tol,kMax);
```

function `newton.m`

```
function [x,k] = newton(f,df,x0,tol,kMax)
% codice del metodo di Newton
.
.
.
```

Programmazione in Matlab: function

Osservazione: per usare una function all'interno di uno script, il file della function deve essere nella stessa directory dello script.

Consiglio: la prima riga `function [x,y,...] = nomefunction(a,b,...)` deve essere seguita da dei commenti `%` che spiegano "cosa fa" la function e da una breve descrizione delle variabili in input ed output. Queste righe di commenti costituiscono l'`help` della function.

Comandi `nargin`, `nargout`

- `nargin` numero di argomenti in input di una function
- `nargout` numero di argomenti in output di una function

sono comandi usati solo all'interno delle function.

Algebra lineare

- Norme vettoriali `norm(x)`, `norm(x,2)`, `norm(x,1)`, `norm(x,inf)`
- Matrici
  - inversa `inv(A)`, determinante `det(A)`
  - spettro `lambda=eig(A)`, raggio spettrale `rho=max(abs(lambda))`
  - dopo aver costruito `A=rand(5)`, si può costruire una matrice simmetrica `H=A'*A`, ortogonale `Q=orth(A)`, triangolare superiore `U=triu(A)`, triangolare inferiore `L=tril(A)`
  - dopo aver costruito un vettore `d=rand(5,1)`, si può costruire una matrice diagonale `D=diag(d)`
  - `n=100`, `dens=0.1`, matrice sparsa `A=sprand(n,n,dens)`, `spy(A)` per visualizzarla, matrice sparsa simmetrica `A=sprandsym(n,dens)`
  - per costruire una matrice strettamente diagonalmente dominante

```
n = 10; A = rand(n);
A = A - diag(diag(A)); %senza diagonale
s = sum(abs(A), 2); %somme riga
A = A + 2*diag(s);
```

`help matfun` : lista completa di comandi relativi al calcolo matriciale

Algebra lineare

- Matrici a banda
  Esempio: matrice tridiagonale

$$\begin{bmatrix} -1 & 4 & & & & \\ 2 & -1 & 4 & & & \\ & 2 & -1 & 4 & & \\ & & \ddots & \ddots & \ddots & \\ & & & 2 & -1 & 4 \\ & & & & 2 & -1 \end{bmatrix}$$

```
A = diag(-1*ones(n,1)) + ...
    diag(4*ones(n-1,1),1) + ...
    diag(2*ones(n-1,1),-1)
```

- Norme matriciali `norm(A)`, `norm(A,2)`, `norm(A,1)`, `norm(A,inf)`, `norm(A,'fro')`

Sistemi di equazioni lineari

$$Ax = b$$

$A \in \mathbb{R}^{n \times n}$ matrice quadrata, $b \in \mathbb{R}^n$ vettore colonna

- Risoluzione di sistemi lineari: `x=A\b;`
- Confronto dei tempi di calcolo usando `inv` e `\`
- Confronto dei tempi di calcolo usando matrice tridiagonale "piena" e sparsa (`sparse`)

Sistemi di equazioni lineari. Metodi di Jacobi e Gauss-Seidel

$$Ax = b, \qquad A \in \mathbb{R}^{n \times n}, \qquad b \in \mathbb{R}^n, \qquad x \in \mathbb{R}^n$$

Esercizio: Implementare il metodo iterativo di **Jacobi** nel file di tipo *function* `jacobi.m`.

- Riceve in **input**: la matrice dei coefficienti A, il termine noto b, un numero massimo di iterazioni $N_{\max}$, una tolleranza τ ed un vettore colonna iniziale x_0.
- Restituisce in **output**: il vettore soluzione x e il numero di iterazioni compiute k.
- Regole di stop:
  - $\|x_k - x_{k-1}\| \leq \tau \|x_k\|$
  - $k > N_{\max}$
  - $\|b - Ax_k\| \leq \tau \|b\|$
- Prima di iniziare il ciclo, calcolare il raggio spettrale della matrice di iterazione B.

Esercizio: Implementare il metodo iterativo di **Gauss-Seidel** nel file di tipo *function* `gs.m` seguendo le stesse istruzioni della function `jacobi`.

Sistemi di equazioni lineari. Metodi di Jacobi e Gauss-Seidel

Esercizio: Preparare uno *script* che costruisce un sistema lineare test la cui matrice dei coefficienti A sia strettamente diagonalmente dominante, il termine noto b sia definito in modo tale che la corrispondente soluzione x coincida con il vettore unitario.

```
n = 100;
alfa = 2; % regola la dominanza diagonale
A = rand(n);
A = A - diag(diag(A));
s = sum(abs(A), 2);
A = A + alfa*diag(s); % matrice strettam. diagonalm. dominante

sol = ones(n,1); % soluzione esatta
b = A*sol; % termine noto
```

Calcolare la soluzione tramite i metodi di Jacobi e Gauss-Seidel.
Calcolare l'errore tra le soluzioni calcolate e quella esatta.
Osservare cosa succede al variare di *alfa*.

Sistemi di equazioni lineari. Metodi di Jacobi e Gauss-Seidel

Se A è una matrice tridiagonale con elementi diagonali non nulli, vale la seguente relazione sul raggio spettrale della matrice di iterazione di Jacobi e quello di Gauss-Seidel

$$\rho(B_{GS}) = (\rho(B_J))^2$$

Esercizio: Allo script creato nell'esercizio precedente aggiungere con `switch case` un'altra matrice dei coefficienti: la matrice A tridiagonale

$$A = \begin{bmatrix} 7 & 3 & & & \\ 2 & 7 & 3 & & \\ & \ddots & \ddots & \ddots & \\ & & 2 & 7 & 3 \\ & & & 2 & 7 \end{bmatrix}$$

Osservare se esiste qualche relazione nella velocità di convergenza tra i due metodi.

Esercizio: All'esercizio precedente, aggiungere nel `switch case` le seguenti matrici dei coefficienti

$$\begin{bmatrix} 3 & 0 & 4 \\ 7 & 4 & 2 \\ -1 & -1 & -2 \end{bmatrix} \quad \begin{bmatrix} -3 & 3 & -6 \\ -4 & 7 & -8 \\ 5 & 7 & -9 \end{bmatrix} \quad \begin{bmatrix} 4 & 1 & 1 \\ 2 & -9 & 0 \\ 0 & -8 & -6 \end{bmatrix} \quad \begin{bmatrix} 7 & 6 & 9 \\ 4 & 5 & -4 \\ -7 & -3 & 8 \end{bmatrix}$$

Commentare i risultati su convergenza o non convergenza dei metodi di Jacobi e Gauss-Seidel e su velocità di convergenza.