

# Calcolo Numerico - Appunti

260236

luglio 2024

## Prefazione

Ogni sezione di teoria presente in questi appunti, è stata ricavata dalle seguenti risorse:

- Calcolo Scientifico: Esercizi e problemi risolti con MATLAB e Octave. [1]

Altro:

 [GitHub repository](#)

# Indice

<b>1</b>	<b>Equazioni non lineari</b>	<b>4</b>
1.1	Introduzione . . . . .	4
1.2	Il metodo di bisezione (o iterativo) . . . . .	4
1.3	Il metodo di Newton . . . . .	8
1.3.1	Come arrestare il metodo di Newton . . . . .	9
1.4	Il metodo delle secanti . . . . .	10
1.5	I sistemi di equazioni non lineari . . . . .	11
1.6	Iterazioni di punto fisso . . . . .	13
<b>2</b>	<b>Metodi risolutivi per sistemi lineari e non lineari</b>	<b>17</b>
2.1	Metodi diretti per sistemi lineari . . . . .	17
2.1.1	Metodo delle sostituzioni in avanti e all'indietro . . . . .	17
2.1.2	Fattorizzazione LRU: MEG e Cholesky . . . . .	20
2.1.3	La tecnica del pivoting . . . . .	21
2.2	Metodi iterativi per sistemi lineari . . . . .	22
2.3	Metodi numerici per sistemi non lineari . . . . .	23
<b>3</b>	<b>Laboratorio</b>	<b>24</b>
3.1	Introduzione al linguaggio MATLAB . . . . .	24
3.1.1	Esercizio . . . . .	41
3.2	Zeri di funzione . . . . .	42
3.2.1	Grafici di funzione . . . . .	42
3.3	Risoluzione di Sistemi di Equazioni Lineari . . . . .	46
3.3.1	Metodi diretti . . . . .	46
	<b>Index</b>	<b>52</b>

# 1 Equazioni non lineari

## 1.1 Introduzione

Il **calcolo degli zeri di una funzione**  $f$  reale di variabile reale o delle **radici dell'equazione**  $f(x) = 0$ , è un problema assai ricorrente nel Calcolo Scientifico.

In generale, *non è possibile* approntare metodi numerici che calcolino gli zeri di una generica funzione in un numero finito di passi. I metodi numerici per la risoluzione di questo problema sono pertanto necessariamente *iterativi*. A partire da uno o più dati iniziali, scelti convenientemente, essi generano una successione di valori  $x^{(k)}$  che, sotto opportune ipotesi, convergerà ad uno zero  $\alpha$  della funzione  $f$  studiata.

---

## 1.2 Il metodo di bisezione (o iterativo)

Sia  $f$  una funzione continua in  $[a, b]$  tale che  $f(a)f(b) < 0$ . Per cui, vale il **teorema degli zeri di una funzione continua**, ossia  $f$  ammette almeno uno zero in  $(a, b)$ .

Si supponga che ci sia un solo zero, indicato con  $\alpha$  e nel caso in cui ce ne sia più di uno, individuare un intervallo tale che ne contenga solo uno.

Il **metodo di bisezione** (o **iterativo**) è una strategia che si suddivide nei seguenti passaggi:

1. **Dimezzare l'intervallo di partenza;**
2. **Selezionare tra i due sotto-intervalli ottenuti quello nel quale  $f$  cambia di segno agli estremi;**
3. **Applicare ricorsivamente questa procedura all'ultimo intervallo selezionato.**

Matematicamente parlando, dato  $I^{(0)} = (a, b)$ , e più in generale,  $I^{(k)}$  il sotto-intervallo selezionato al passo  $k$ -esimo, si sceglie come  $I^{(k+1)}$  il semi-intervallo di  $I^{(k)}$  ai cui estremi  $f$  cambia di segno.

Questa procedura garantisce che ogni sotto-intervallo selezionato  $I^{(k)}$  conterrà  $\alpha$ . Questo poiché la successione  $\{x^{(k)}\}$  dei punti medi dei sotto-intervalli  $I^{(k)}$  dovrà ineluttabilmente convergere a  $\alpha$ , in quanto la **lunghezza dei sotto-intervalli tende a 0** per  $k$  che **tende all'infinito**.

Formalizziamo questa idea con un piccolo algoritmo. Ponendo:

$$a^{(0)} = a, \quad b^{(0)} = b, \quad I^{(0)} = (a^{(0)}, b^{(0)}), \quad x^{(0)} = \frac{a^{(0)} + b^{(0)}}{2}$$

Al passo  $k \geq 1$  il metodo di bisezione calcolerà il semi-intervallo  $I^{(k)} = (a^{(k)}, b^{(k)})$  dell'intervallo  $I^{(k-1)} = (a^{(k-1)}, b^{(k-1)})$ , nel seguente modo (si ricorda che  $\alpha$  è lo zero che si sta cercando):

1. Calcolo  $x^{(k-1)} = \frac{a^{(k-1)} + b^{(k-1)}}{2}$
2. Se  $f(x^{(k-1)}) = 0$ :
  - (a) Allora  $\alpha = x^{(k-1)}$  e l'algoritmo termina.
3. Altrimenti, se  $f(a^{(k-1)}) \cdot f(x^{(k-1)}) < 0$ :
  - (a) Si pone  $a^{(k)} = a^{(k-1)}$
  - (b) Si pone  $b^{(k)} = x^{(k-1)}$
  - (c) Si incrementa  $k + 1$  e si ripete ricorsivamente.
4. Altrimenti, se  $f(x^{(k-1)}) \cdot f(b^{(k-1)}) < 0$ :
  - (a) Si pone  $a^{(k)} = x^{(k-1)}$
  - (b) Si pone  $b^{(k)} = b^{(k-1)}$
  - (c) Si incrementa  $k + 1$  e si ripete ricorsivamente.

### Esempio 1

Data la funzione  $f(x) = x^2 - 1$ , si parta da  $a^{(0)} = -0.25$  e  $b^{(0)} = 1.25$ , e si applichi il metodo di bisezione:

1. Con  $a^{(0)} = -0.25$  e  $b^{(0)} = 1.25$ :

- (a) Si calcola il punto medio:

$$x^{(0)} = \frac{a^{(0)} + b^{(0)}}{2} = \frac{-0.25 + 1.25}{2} = 0.5$$

- (b) Si calcola la funzione con il punto medio come parametro:

$$f(0.5) = 0.5^2 - 1 = -0.75$$

- (c) Dato che la funzione nel punto medio non è uguale a zero, l'algoritmo deve continuare. Per farlo, bisogna sostituire il punto medio con uno dei due estremi. Per decidere quale dei due sostituire, è necessario capire in quale cambia valore la funzione. Si verifica inizialmente con  $a^{(0)}$ :

$$\begin{aligned} f(a^{(0)}) f(x^{(0)}) < 0 &= f(-0.25) f(0.5) < 0 \\ &= (-0.9375) \cdot (-0.75) < 0 \\ &= 0.703125 \quad \times \end{aligned}$$

(d) Si procede con l'algoritmo, provando adesso la  $b^{(0)}$ :

$$\begin{aligned} f(x^{(0)}) f(b^{(0)}) < 0 &= f(0.5) f(1.25) < 0 \\ &= (-0.75 \cdot 0.5625) < 0 \\ &= -0.421875 \checkmark \end{aligned}$$

(e) Si pone  $a^{(1)} = x^{(0)} = 0.5$

(f) Si pone  $b^{(1)} = b^{(0)} = 1.25$

(g) Si incrementa  $k$ ,  $k = k + 1 = 0 + 1 = 1$

2. Con  $a^{(1)} = 0.5$  e  $b^{(1)} = 1.25$ :

(a) Si calcola il punto medio:

$$x^{(1)} = \frac{a^{(1)} + b^{(1)}}{2} = \frac{0.5 + 1.25}{2} = 0.875$$

(b) Si calcola la funzione con il punto medio come parametro:

$$f(0.875) = 0.875^2 - 1 = -0.234375$$

(c) Dato che la funzione nel punto medio non è uguale a zero, l'algoritmo deve continuare:

$$\begin{aligned} f(a^{(1)}) f(x^{(1)}) < 0 &= f(0.5) f(-0.234375) < 0 \\ &= (-0.75 \cdot -0.945068359375) < 0 \\ &= 0.70880126953125 \times \end{aligned}$$

(d) Si procede con l'algoritmo:

$$\begin{aligned} f(x^{(1)}) f(b^{(1)}) < 0 &= f(-0.234375) f(1.25) < 0 \\ &= (-0.945068359375 \cdot 0.5625) < 0 \\ &= -0.5316009521484375 \checkmark \end{aligned}$$

(e) Si pone  $a^{(2)} = x^{(1)} = 0.875$

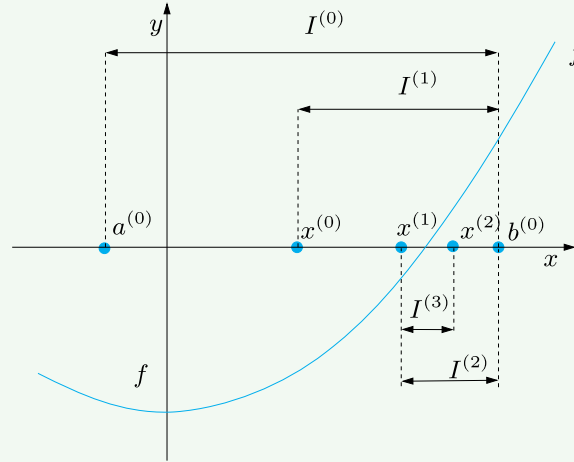
(f) Si pone  $b^{(2)} = b^{(1)} = 1.25$

(g) Si incrementa  $k$ ,  $k = k + 1 = 1 + 1 = 2$

Si omettono i restanti calcoli per  $k = 2, k = 3$ , ma si lasciano qua di seguito i risultati:

- $I^{(2)} = (0.875, 1.25)$  e  $x^{(2)} = 1.0625$
- $I^{(3)} = (0.875, 1.0625)$  e  $x^{(2)} = 0.96875$

Nella seguente figura si possono vedere le iterazioni effettuate:



Iterazioni effettuate. [1]

Si noti che ogni intervallo  $I^{(k)}$  contiene lo zero  $\alpha$ . Inoltre, la successione  $\{x^{(k)}\}$  converge necessariamente allo zero  $\alpha$  in quanto ad ogni passo l'ampiezza  $|I^{(k)}| = b^{(k)} - a^{(k)}$  dell'intervallo  $I^{(k)}$  si dimezza.

Il valore  $I^{(k)}$  può essere riassunto come:

$$|I^{(k)}| = \left(\frac{1}{2}\right)^k \cdot |I^{(0)}|$$

E di conseguenza l'errore al passo  $k$  può essere calcolato come:

$$|e^{(k)}| = |x^{(k)} - \alpha| < \frac{1}{2} \cdot |I^{(k)}| = \left(\frac{1}{2}\right)^{k+1} \cdot (b - a)$$

Inoltre, data una certa **tolleranza**  $\varepsilon$ , per **garantire che l'errore al passo  $k$  sia minore della tolleranza data** (ovvero,  $|e^{(k)}| < \varepsilon$ ), basta applicare la seguente formula:

$$k_{\min} > \log_2 \left( \frac{b - a}{\varepsilon} \right) - 1 \quad (1)$$

Dove  $k_{\min}$  rappresenta il **numero minimo** di iterazioni prima di trovare un intero che soddisfi la disuguaglianza.

### ⚠ Possibile svantaggio

Il metodo di bisezione **non garantisce una riduzione monotona dell'errore**, ma solo il dimezzamento dell'ampiezza dell'intervallo all'interno del quale si cerca lo zero. Infatti, **non viene tenuto conto del reale andamento di  $f$**  e questo può provocare il mancato coinvolgimento di approssimazioni di  $\alpha$  accurate.

### 1.3 Il metodo di Newton

Il **metodo di Newton** sfrutta la funzione  $f$  maggiormente rispetto al metodo di bisezione, usando i suoi valori e la sua derivata.

Si ricorda che la retta tangente alla curva  $(x, f(x))$  nel punto  $x^{(k)}$  è:

$$y(x) = f(x^{(k)}) + f'(x^{(k)})(x - x^{(k)})$$

Cercando un  $x^{(k+1)}$  tale che la **retta tangente in quel punto sia uguale a zero**  $y(x^{(k+1)}) = 0$ , allora si trova:

$$x^{(k+1)} = x^{(k)} - \frac{f(x^{(k)})}{f'(x^{(k)})} \quad k \geq 0 \quad (2)$$

Purché la derivata prima nel punto  $x^{(k)}$  sia diversa da zero, cioè  $f'(x^{(k)}) \neq 0$ .

Questa equazione consente di calcolare una successione di valori  $x^{(k)}$  a partire da un dato iniziale  $x^{(0)}$ . In altre parole, il **metodo di Newton calcola lo zero di  $f$  sostituendo localmente a  $f$  la sua retta tangente**.

A differenza del metodo di bisezione, tale **metodo converge allo zero in un solo passo quando la funzione  $f$  è lineare**, ovvero nella forma  $f(x) = a_1x + a_0$ .

#### Limitazione

La **convergenza** del metodo di Newton non è garantita **per ogni scelta** di  $x^{(0)}$ , ma **soltanto** per valori di  $x^{(0)}$  **sufficientemente vicini** ad  $\alpha$ , ovvero **appartenenti ad un intorno  $I(\alpha)$  sufficientemente piccolo di  $\alpha$** .

Alcune osservazioni a seguito anche di questa limitazione:

- A seguito di questa limitazione, risulta evidente che se  $x^{(0)}$  è stato scelto opportunamente e se lo zero  $\alpha$  è semplice ( $f'(\alpha) \neq 0$ ), allora il metodo converge.
- Nel caso in cui  $f$  è derivabile con continuità pari a due, allora si ottiene la seguente convergenza:

$$\lim_{k \rightarrow \infty} \frac{x^{(k+1)} - \alpha}{(x^{(k)} - \alpha)^2} = \frac{f''(\alpha)}{2f'(\alpha)} \quad (3)$$

Il significato è: se  $f'(\alpha) \neq 0$  il metodo di Newton converge almeno quadraticamente o con **ordine 2**.

In parole povere, **per  $k$  sufficientemente grande, l'errore al passo  $(k+1)$ -esimo si comporta come il quadrato dell'errore al passo  $k$ -esimo, moltiplicato per una costante indipendente da  $k$** .

- Se lo zero  $\alpha$  ha molteplicità  $m$  maggiore di 1, ovverosia:

$$f'(\alpha) = 0, \dots, f^{(m-1)}(\alpha) = 0$$



Allora il metodo di Newton è ancora convergente, purché  $x^{(0)}$  sia scelto opportunamente e  $f'(x) \neq 0 \forall x \in I(\alpha) \setminus \{\alpha\}$ . Tuttavia in questo caso l'ordine di convergenza è pari a 1. In tal caso, l'ordine 2 può essere ancora recuperato usando la seguente relazione al posto dell'equazione 2 ufficiale:

$$x^{(k+1)} = x^{(k)} - m \cdot \frac{f(x^{(k)})}{f'(x^{(k)})} \quad k \geq 0 \quad (4)$$

Purché  $f'(x^{(k)}) \neq 0$ . Naturalmente, questo **metodo di Newton modificato** richiede una conoscenza a priori di  $m$ .

### 1.3.1 Come arrestare il metodo di Newton

Data una tolleranza fissa  $\varepsilon$ , esistono due tecniche applicabili per capire quando è necessario fermarsi ed evitare di continuare ad iterare:

- La **differenza fra due iterate consecutive**, il quale si arresta in corrispondenza del più piccolo intero  $k_{\min}$  per il quale:

$$\left| x^{(k_{\min})} - x^{(k_{\min}-1)} \right| < \varepsilon \quad (5)$$

(test sull'incremento).

- Un'altra tecnica applicata anche per altri metodi iterativi è il **residuo** al passo  $k$ , il quale è definito come:

$$r^{(k)} = f(x^{(k)})$$

Che è nullo quando  $x^{(k)}$  è uno zero di  $f$ . In questo modo, il metodo viene arrestato alla prima iterata  $k_{\min}$ :

$$\left| r^{(k_{\min})} \right| = \left| f(x^{(k_{\min})}) \right| < \varepsilon \quad (6)$$

Da notare che tale tecnica fornisce una **stima accurata dell'errore** solo quando  $|f'(x)|$  è circa pari a 1 in un intorno di  $I_\alpha$  dello zero  $\alpha$  cercato.

**Attenzione!** Se la derivata non è circa pari a 1 in un intorno dello zero cercato, la tecnica porterà:

- Ad una **sovrastima** dell'errore se  $|f'(x)| \gg 1$  per  $x \in I_\alpha$
- Ad una **sottostima** dell'errore se  $|f'(x)| \ll 1$  per  $x \in I_\alpha$

### 1.4 Il metodo delle secanti

Nel caso in cui la funzione  $f$  non sia nota, il metodo di Newton non può essere applicato. Per fortuna, arriva in soccorso il **metodo delle secanti**, il quale esegue una valutazione di  $f'(x^{(k)})$  andando a sostituire quest'ultima con un **rapporto incrementale calcolato su valori di  $f$  già noti**.

Più formalmente, assegnati due punti  $x^{(0)}$  e  $x^{(1)}$ , per  $k \geq 1$  si calcola:

$$x^{(k+1)} = x^{(k)} - \left( \frac{f(x^{(k)}) - f(x^{(k-1)})}{x^{(k)} - x^{(k-1)}} \right)^{-1} \cdot f(x^{(k)}) \quad (7)$$

#### ❓ Quando converge?

Il metodo delle secanti converge a seguito di certe condizioni:

- **Converge ad  $\alpha$** , se:
  - $\alpha$  radice semplice<sup>1</sup>;
  - $I(\alpha)$  è un opportuno intorno di  $\alpha$ ;
  - $x^{(0)}$  e  $x^{(1)}$  sono sufficientemente vicini ad  $\alpha$
  - $f'(x) \neq 0 \quad \forall x \in I(\alpha) \setminus \{\alpha\}$
- **Converge con ordine  $p$  super-lineare**, se:
  - $f \in \mathcal{C}^2(I(\alpha))$
  - $f'(\alpha) \neq 0$

Ovvero, esiste una costante  $c > 0$  tale che:

$$\left| x^{(k+1)} - \alpha \right| \leq c \left| x^{(k)} - \alpha \right|^p \quad p = \frac{1 + \sqrt{5}}{2} \approx 1.618 \dots \quad (8)$$

- **Convergenza lineare**, se:
  - Radice  $\alpha$  è multipla.

Come succederebbe usando il metodo di Newton.

---

<sup>1</sup> $f'(\alpha) \neq 0$

## 1.5 I sistemi di equazioni non lineari

Di solito i metodi presentati nelle pagine precedenti vengono inseriti in dei sistemi. Nella realtà ci sono varie condizioni che influiscono sul sistema in analisi. È per questo motivo che si introducono i sistemi.

Si consideri un generale sistema di equazioni non lineari:

$$\begin{cases} f_1(x_1, x_2, \dots, x_n) = 0 \\ f_2(x_1, x_2, \dots, x_n) = 0 \\ \vdots \\ f_n(x_1, x_2, \dots, x_n) = 0 \end{cases}$$

Dove  $f_1, \dots, f_n$  sono funzioni non lineari. Si pongono i seguenti vettori:

- $\mathbf{f} \equiv (f_1, \dots, f_n)^T$
- $\mathbf{x} \equiv (x_1, \dots, x_n)^T$

Con l'obiettivo di riscrivere il sistema in maniera più agevole:

$$\mathbf{f}(\mathbf{x}) = \mathbf{0}$$

### Esempio 2: esempio di sistema non lineare

Un esempio banale di sistema non lineare:

$$\begin{cases} f_1(x_1, x_2) = x_1^2 + x_2^2 - 1 = 0 \\ f_2(x_1, x_2) = \sin\left(\pi \frac{x_1}{2}\right) + x_2^3 = 0 \end{cases}$$

Prima di estendere i metodi di Newton e delle secanti si introduce la matrice Jacobiana.

### Definizione 1: matrice Jacobiana

Senza entrare troppo nel gergo matematico (non è l'obiettivo del corso), la **matrice Jacobiana** di una funzione è quella **matrice i cui elementi sono le derivate parziali prime della funzione**.

$$\mathbf{J}_{\mathbf{f}} = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \dots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \dots & \frac{\partial f_m}{\partial x_n} \end{pmatrix} \quad (9)$$

Che può essere riscritto in modo più leggibile come:

$$(\mathbf{J}_{\mathbf{f}})_{ij} \equiv \frac{\partial f_i}{\partial x_j} \quad i, j = 1, \dots, n \quad (10)$$

Dove rappresenta la derivata parziale della funzione  $f_i$  rispetto a  $x_j$ .

Il metodo di Newton e delle secanti può essere esteso sfruttando la matrice Jacobiana:

- Il **metodo di Newton** usando un sistema di equazioni non lineari diventa: dato  $\mathbf{x}^{(0)} \in \mathbb{R}^n$ , per  $k = 0, 1, \dots$ , fino a convergenza

$$\begin{aligned} &\text{risolvere} && \mathbf{J}_f(\mathbf{x}^{(k)}) \delta \mathbf{x}^{(k)} = -\mathbf{f}(\mathbf{x}^{(k)}) \\ &\text{porre} && \mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \delta \mathbf{x}^{(k)} \end{aligned} \quad (11)$$

Se ne deduce che venga richiesto ad ogni passo la soluzione di un sistema lineare di matrice  $\mathbf{J}_f(\mathbf{x}^{(k)})$ .

- Il **metodo delle secanti** usando un sistema di equazioni non lineari si basa sulla matrice Jacobiana e sul metodo di Broyden.

L'**idea di base** è sostituire le matrici Jacobiane  $\mathbf{J}_f(\mathbf{x}^{(k)})$  (per  $k \geq 0$ ) con delle matrici chiamate  $B_k$ , definite ricorsivamente a partire da una matrice  $B_0$  che sia una approssimazione di  $\mathbf{J}_f(\mathbf{x}^{(0)})$ .

Dato  $\mathbf{x}^{(0)} \in \mathbb{R}^n$ , data  $B_0 \in \mathbb{R}^{n \times n}$  per  $k = 0, 1, \dots$ , fino a convergenza

$$\begin{aligned} &\text{risolvere} && B_k \delta \mathbf{x}^{(k)} = -\mathbf{f}(\mathbf{x}^{(k)}) \\ &\text{porre} && \mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \delta \mathbf{x}^{(k)} \\ &\text{porre} && \delta \mathbf{f}^{(k)} = \mathbf{f}(\mathbf{x}^{(k+1)}) - \mathbf{f}(\mathbf{x}^{(k)}) \\ &\text{calcolare} && B_{k+1} = B_k + \frac{(\delta \mathbf{f}^{(k)} - B_k \delta \mathbf{x}^{(k)}) \delta \mathbf{x}^{(k)T}}{\delta \mathbf{x}^{(k)T} \delta \mathbf{x}^{(k)}} \end{aligned} \quad (12)$$

Da notare che non si chiede alla successione  $\{B_k\}$  così costruita di convergere alla vera matrice Jacobiana  $\mathbf{J}_f(\boldsymbol{\alpha})$  ( $\boldsymbol{\alpha}$  è la radice del sistema); questo risultato non è garantito tuttavia.

## 1.6 Iterazioni di punto fisso

### Esempio 3: esempio di introduzione

Con una calcolatrice si può facilmente verificare che applicando ripetutamente la funzione  $\cos$  partendo dal numero 1 si genera la seguente successione di numeri reali:

$$\begin{aligned} x^{(1)} &= \cos(1) = 0.54030230586814 \\ x^{(2)} &= \cos(x^{(1)}) = 0.85755321584639 \\ &\vdots \\ x^{(10)} &= \cos(x^{(9)}) = 0.74423735490056 \\ &\vdots \\ x^{(20)} &= \cos(x^{(19)}) = 0.73918439977149 \end{aligned}$$

Che tende al valore  $\alpha = 0.73908513$ .

Con l'esempio di introduzione è possibile capire il punto fisso. Essendo per costruzione  $x^{(k+1)} = \cos(x^{(k)})$  per  $k = 0, 1, \dots$  (con  $x^{(0)} = 1$ ),  $\alpha$  è tale che  $\cos(\alpha) = \alpha$ . Quindi,  $\alpha$  viene detto punto fisso della funzione coseno.

### ? Perché è interessante?

Se  $\alpha$  è un punto fisso per il coseno, allora esso è uno zero della funzione  $f(x) = x - \cos(x)$  ed il metodo appena proposto potrebbe essere usato per il calcolo degli zeri di  $f$ .

### ⚠ Non tutte le funzioni hanno un punto fisso

Non tutte le funzioni ammettono punti fissi. Ad esempio, ripetendo l'esperimento dell'esempio con una funzione esponenziale a partire da  $x^{(0)} = 1$ , dopo soli 4 passi si giunge ad una situazione di *overflow* (figura 1, pagina 14).

### Definizione 2

Data una funzione  $\phi : [a, b] \rightarrow \mathbb{R}$ , trovare  $\alpha \in [a, b]$  tale che:

$$\alpha = \phi(\alpha)$$

Se tale  $\alpha$  esiste, viene detto un **punto fisso** di  $\phi$  e lo si può determinare come limite della seguente successione:

$$x^{(k+1)} = \phi(x^{(k)}) \quad k \geq 0 \quad (13)$$

Dove  $x^{(0)}$  è un dato iniziale. L'algoritmo viene chiamato **iterazioni di punto fisso** e la funzione  $\phi$  è detta **funzione di iterazione**.

Dalla definizione, si deduce che l'esempio introduttivo è un algoritmo di iterazioni di punto fisso per la funzione  $\phi(x) = \cos(x)$ .

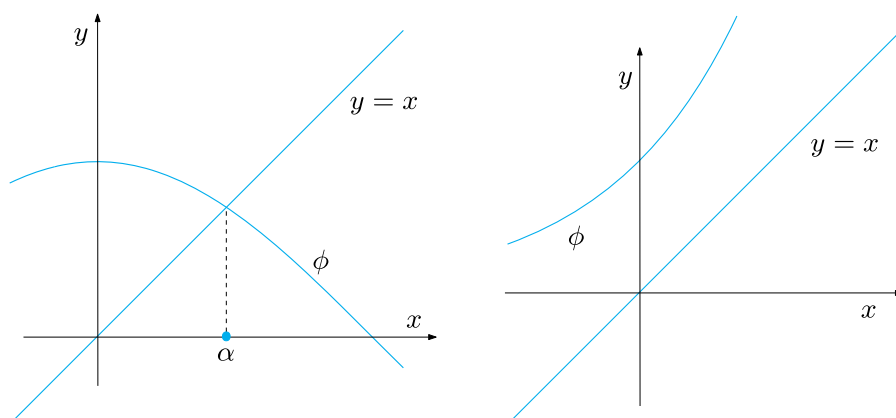


Figura 1: La funzione  $\phi(x) = \cos(x)$  (sx) ammette un solo punto fisso, mentre la funzione  $\phi(x) = e^x$  (dx) non ne ammette alcuno.

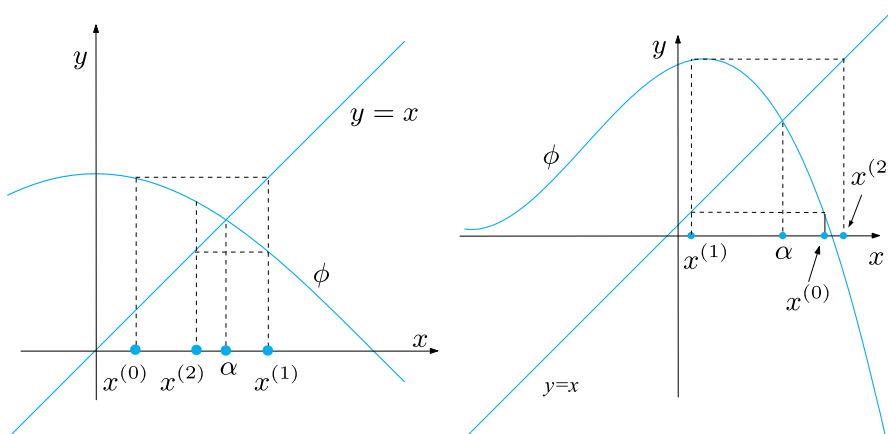


Figura 2: Rappresentazione delle prime iterazioni di punto fisso per due funzioni di iterazione. Le iterazioni convergono verso il punto fisso  $\alpha$  (sx), mentre si allontanano da  $\alpha$  (dx).

**Definizione 3: quando una funzione ha un punto fisso?**

Si consideri la successione (formula) 13 a pagina 13.

1. Si supponga che  $\phi(x)$  sia continua nell'intervallo  $[a, b]$  e che  $\phi(x) \in [a, b]$  per ogni  $x \in [a, b]$ ; allora **esiste almeno un punto fisso**  $\alpha \in [a, b]$ .
2. Si supponga inoltre che esista un valore  $L$  minore di 1 tale per cui:

$$|\phi(x_1) - \phi(x_2)| \leq L |x_1 - x_2|$$

Per ogni  $x_1, x_2$  appartenente all'insieme  $[a, b]$ . Con tale supposizione, allora  $\phi$  ha un **unico punto fisso**  $\phi \in [a, b]$  e la successione definita nell'equazione 13 a pagina 13 converge a  $\alpha$ , qualunque sia il dato iniziale  $x^{(0)}$  in  $[a, b]$ .

La supposizione scritta in precedenza può essere riassunta in un'equazione:

$$\exists L < 1 \text{ t.c. } |\phi(x_1) - \phi(x_2)| \leq L |x_1 - x_2| \quad \forall x_1, x_2 \in [a, b] \quad (14)$$

Nella pratica è però spesso **difficile delimitare a priori l'ampiezza dell'intervallo**  $[a, b]$ ; in tal caso è utile il seguente risultato di convergenza locale:

**Theorem 1** (di Ostrowski). *Sia  $\alpha$  un punto fisso di una funzione  $\phi$  continua e derivabile con continuità in un opportuno intorno  $\mathcal{I}$  di  $\alpha$ . Se risulta  $|\phi'(\alpha)| < 1$ , allora esiste un  $\delta > 0$  in corrispondenza del quale la successione  $\{x^{(k)}\}$  converge ad  $\alpha$ , per ogni  $x^{(0)}$  tale che  $|x^{(0)} - \alpha| < \delta$ . Inoltre si ha:*

$$\lim_{k \rightarrow \infty} \frac{x^{(k+1)} - \alpha}{x^{(k)} - \alpha} = \phi'(\alpha) \quad (15)$$

Dal teorema si deduce che le iterazioni di punto fisso convergono almeno linearmente cioè che, per  $k$  sufficientemente grande, l'errore del passo  $k+1$  si comporta come l'errore al passo  $k$  moltiplicato per una costante,  $\phi'(\alpha)$  nel teorema, indipendente da  $k$  ed il cui valore assoluto è minore di 1. Per questo motivo la costante viene chiamata **fattore di convergenza** e la convergenza sarà tanto più rapida quanto più piccola è tale costante.

**Definizione 4: quando il metodo di punto fisso è convergente**

Si suppongano valide le ipotesi del teorema di Ostrowski 1. Se, inoltre,  $\phi$  è derivabile con continuità due volte e se:

$$\phi'(\alpha) = 0 \quad \phi''(\alpha) \neq 0$$

Allora il metodo di punto fisso (eq. 13) è convergente di ordine 2 e si ha:

$$\lim_{k \rightarrow \infty} \frac{x^{(k+1)} - \alpha}{(x^{(k)} - \alpha)^2} = \frac{1}{2} \phi''(\alpha) \quad (16)$$

Un'ultima osservazione interessante:

- Nel caso in cui  $|\phi'(\alpha)| > 1$ , se  $x^{(k)}$  è sufficientemente vicino ad  $\alpha$ , in modo tale che  $|\phi'(x^{(k)})| > 1$ , allora  $|\alpha - x^{(k+1)}| > |\alpha - x^{(k)}|$ , e **non è possibile che la successione converga al punto fisso**.
- Nel caso in cui  $|\phi'(\alpha)| = 1$  **non si può trarre alcuna conclusione** perché potrebbero verificarsi sia la convergenza sia la divergenza, a seconda delle caratteristiche della funzione di punto fisso.



## 2 Metodi risolutivi per sistemi lineari e non lineari

### 2.1 Metodi diretti per sistemi lineari

#### 2.1.1 Metodo delle sostituzioni in avanti e all'indietro

#### 🔍 Perché sono importanti i metodi numerici?

Si consideri il seguente **sistema lineare**:

$$Ax = b$$

Dove:

- $A \in \mathbb{R}^{n \times n}$  di componenti  $a_{ij}$  e  $b \in \mathbb{R}^n$  sono valori noti.
- $x \in \mathbb{R}^n$  è il vettore delle incognite.
- La costante  $n$  rappresenta il numero di equazioni lineari delle incognite  $x_j$ .

Con queste caratteristiche, è possibile rappresentare la  $i$ -esima equazione nel seguente modo:

$$\sum_{j=1}^n a_{ij}x_j = b_i \rightarrow a_{1i}x_1 + a_{2i}x_2 + \dots + a_{ni}x_n = b_i \quad \forall i = 1, \dots, n$$

La **soluzione esatta** del sistema, chiamata **formula di Cramer**, è:

$$x_j = \frac{\det(A_j)}{\det(A)} \quad (17)$$

Con  $A_j = |a_1 \dots a_{j-1} \ b \ a_{j+1} \dots a_n|$  e  $a_i$  le colonne di  $A$ . Ovviamente la soluzione **esiste ed è unica se il determinante** della matrice  $A$  è **diverso da zero**:

$$\det(A) \neq 0$$

Purtroppo questo metodo è **inutilizzabile** poiché il calcolo di un determinante richiede all'incirca  $n!$  (fattoriale di  $n$ ) operazioni.

Risulta evidente che sia necessario uno studio approfondito di **metodi numerici che si traducano in algoritmi efficienti** da farli eseguire su calcolatori. Nelle seguenti pagine si introducono i primi due algoritmi “efficienti”.

### Definizione 1

Il seguente algoritmo rappresenta il **metodo delle sostituzioni in avanti**. Dati:

- $L \in \mathbb{R}^{n \times n}$  matrice triangolare inferiore non singolare (cioè con determinante diverso da zero  $\det(L) \neq 0$ )
- $\mathbf{b} \in \mathbb{R}^n$  vettore termine noto

La soluzione è data da  $Lx = \mathbf{b}$  con  $x \in \mathbb{R}^n$ . Più in generale si ha:

$$x_i = \frac{b_i - \sum_{j=1}^{i-1} L_{ij} x_j}{L_{ii}} \quad (18)$$

Per comprendere meglio la definizione del metodo di sostituzioni in avanti, è possibile visualizzare in modo generale la matrice triangolare inferiore  $L$  (non singolare):

$$L_{n \times n} = \begin{bmatrix} L_{11} & 0 & 0 & 0 & 0 \\ & \ddots & 0 & 0 & 0 \\ & & \ddots & 0 & 0 \\ & L_{ij} & & \ddots & 0 \\ & & & & L_{nn} \end{bmatrix}$$

Dove ovviamente  $n$  è la grandezza della matrice quadrata. Dalla matrice, è possibile rappresentare le prime tre iterazioni, ovvero  $x_1$ ,  $x_2$  e  $x_3$ :

- La prima riga è:

$$x_1 = \frac{b_1}{L_{11}}$$

Si può scrivere anche in linea nel seguente modo:  $L_{11}x_1 = b_1$ .

- La seconda riga:

$$x_2 = \frac{b_2 - (L_{21} \cdot x_1 + L_{22} \cdot \cancel{x_2})}{L_{22}}^0$$

In cui  $x_1$  è il risultato del punto precedente e  $x_2$  è il risultato che attualmente si sta calcolando, quindi uguale a zero.

- La terza riga:

$$x_3 = \frac{b_3 - (L_{31} \cdot x_1 + L_{32} \cdot x_2 + L_{33} \cdot \cancel{x_3})}{L_{33}}^0$$

Il **numero di operazioni** richieste dal metodo delle sostituzioni in avanti è dato da 1 sottrazione,  $i - 1$  moltiplicazioni,  $i - 2$  addizioni e 1 divisione:

$$\#op. = \sum_{i=1}^n (i - 1) + (i - 2) + 1 + 1 = \sum_{i=1}^n (2i - 1) = n^2 \quad (19)$$

Per completezza si presenta anche il metodo delle sostituzioni all'indietro.

### Definizione 2

Il seguente algoritmo rappresenta il **metodo delle sostituzioni all'indietro**. Dati:

- $U \in \mathbb{R}^{n \times n}$  matrice triangolare superiore non singolare (cioè con determinante diverso da zero  $\det(U) \neq 0$ )
- $\mathbf{b} \in \mathbb{R}^n$  vettore termine noto

La soluzione è data da  $Ux = \mathbf{b}$  con  $x \in \mathbb{R}^n$ . Più in generale si ha:

$$x_i = \frac{b_i - \sum_{j=i+1}^n U_{ij} x_j}{U_{ii}} \quad (20)$$

Come per il metodo precedente, anche in questo caso è utile visualizzare la matrice generale:

$$U_{n \times n} = \begin{bmatrix} U_{11} & & & & \\ 0 & \ddots & & & \\ 0 & 0 & \ddots & & \\ 0 & 0 & 0 & \ddots & \\ 0 & 0 & 0 & 0 & U_{nn} \end{bmatrix}$$

Anche da questa matrice è possibile rappresentare le iterazioni:

- La prima riga è:

$$x_1 = \frac{b_1 - (U_{12} \cdot x_2 + U_{13} \cdot x_3 + U_{14} \cdot x_4)}{U_{11}}$$

- La seconda riga è:

$$x_2 = \frac{b_2 - (U_{23} \cdot x_3 + U_{24} \cdot x_4)}{U_{22}}$$

- La terza riga è:

$$x_3 = \frac{b_3 - (U_{34} \cdot x_4)}{U_{33}}$$

- L'ultima riga è:

$$x_4 = \frac{b_4}{U_{44}}$$

Si deduce ovviamente che l'ultima riga può essere generalizzata nel seguente modo:

$$x_n = \frac{b_n}{U_{nn}}$$

Infine, il **numero di operazioni** è il medesimo del metodo delle sostituzioni in avanti, quindi  $n^2$ .

### 2.1.2 Fattorizzazione LRU: MEG e Cholesky

Sia  $A \in \mathbb{R}^{n \times n}$ . Si supponga che esistano due opportune matrici  $L$  ed  $U$ , triangolare inferiore e superiore, rispettivamente, tali che:

$$A = LU \quad (21)$$

L'equazione viene chiamata **fattorizzazione LU** (o **decomposizione LU**) di  $A$ .

La fattorizzazione LU è stata introdotta poiché se  $A$  **non è singolare** (quindi il determinante è diverso da zero) tali matrici devono essere **anch'esse non singolari**; questo assicura che i loro **elementi diagonali siano non nulli**. Da questa osservazione, si ottiene un risultato interessante perché la risoluzione di  $Ax = b$  è *equivalente* alla risoluzione dei due seguenti sistemi triangolari:

$$Ly = b \quad Ux = y \quad (22)$$

Dove  $y$  rappresenta la soluzione dell'equazione 18 a pagina 18, ovvero la risoluzione del metodo delle sostituzioni in avanti. Analogamente, la  $x$  rappresenta la soluzione dell'equazione 20 a pagina 19, ovvero la risoluzione del metodo delle sostituzioni all'indietro.

#### ❓ Chiaro, ma che algoritmi esistono per calcolare la fattorizzazione LU?

Esistono principalmente due algoritmi: il **Metodo di Eliminazione di Gauss (MEG)** e la **Fattorizzazione di Cholesky**.

- Senza entrare troppo nel dettaglio, la fattorizzazione LU viene chiamata anche fattorizzazione di Gauss poiché è dimostrato che è possibile applicare l'algoritmo di Gauss, ovvero il **Metodo di Eliminazione di Gauss (MEG)**.

Il **MEG** è possibile **applicarlo** per alcuni tipi di matrici:

1. Le **matrici a dominanza diagonale stretta**. Una matrice è detta **matrice a dominanza diagonale per righe** se:

$$|a_{ii}| \geq \sum_{j=1 \wedge j \neq i}^n |a_{ij}|, \quad i = 1, \dots, n \quad (23)$$

Una matrice è detta **matrice a dominanza diagonale per colonne** se:

$$|a_{ii}| \geq \sum_{j=1 \wedge j \neq i}^n |a_{ji}|, \quad i = 1, \dots, n \quad (24)$$

Quando nelle precedenti disuguaglianze è possibile sostituire il segno  $\geq$  con quello  $>$  si può dire che la matrice  $A$  è a dominanza diagonale **stretta**.

2. Le **matrici reali simmetriche<sup>2</sup> e definite positive<sup>3</sup>**.

---

<sup>2</sup>Una matrice è simmetrica se coincide con la sua matrice trasposta.

<sup>3</sup>Una matrice viene **definita positiva** se:

$$\forall \mathbf{x} \in \mathbb{R}^n \quad \text{con } \mathbf{x} \neq \mathbf{0}, \quad \mathbf{x}^T A \mathbf{x} > 0$$

Il calcolo dei coefficienti dei fattori  $L$  ed  $U$  richiede circa  $\frac{2n^3}{3}$  operazioni.

- Se la matrice  $A$ , cioè la matrice usata nella definizione, è **simmetrica e definita positiva**, è possibile trovare la **fattorizzazione di Cholesky**:

$$A = R^T R \quad (25)$$

Dove  $R$  è una matrice triangolare superiore con elementi positivi sulla diagonale. Inoltre, tale fattorizzazione è **unica**.

Il calcolo della matrice  $R$  richiede circa  $\frac{n^3}{3}$  operazioni (cioè la metà di quelle richieste per calcolare le due matrici della fattorizzazione LU).

---

### 2.1.3 La tecnica del pivoting

Si introduce un metodo che consenta di portare a compimento il processo di fattorizzazione LU per una qualunque matrice  $A$  non simmetrica ( $\det(A) \neq 0$ ).

La tecnica si basa sulla **permutazione** (cioè sullo scambio) opportuno di righe della matrice di partenza  $A$ . Purtroppo non è noto a priori quali siano le righe che dovranno essere tra loro scambiate; tuttavia questa decisione può essere presa ad ogni passo durante il quale si generino elementi nulli.

Dato che lo scambio tra righe comporta un cambiamento del pivot, questa tecnica viene chiamata **pivoting per righe**. La fattorizzazione che si trova restituisce la matrice  $A$  di partenza a meno di una permutazione fra le righe. Precisamente:

$$PA = LU \quad (26)$$

Dove  $P$  è un'opportuna **matrice di permutazione**. Ovvero, è una matrice uguale all'identità all'inizio del processo di fattorizzazione e se durante l'applicazione le righe di  $A$  vengono scambiate, allora deve essere fatto uno scambio analogo sulle righe di  $P$ . Per cui, alla fine sarà necessario risolvere i seguenti sistemi triangolari:

$$Ly = Pb \quad Ux = y \quad (27)$$

Dove  $y$  rappresenta la soluzione dell'equazione 18 a pagina 18, ovvero la risoluzione del metodo delle sostituzioni in avanti. Analogamente, la  $x$  rappresenta la soluzione dell'equazione 20 a pagina 19, ovvero la risoluzione del metodo delle sostituzioni all'indietro.

## 2.2 Metodi iterativi per sistemi lineari

## **2.3 Metodi numerici per sistemi non lineari**

## 3 Laboratorio

### 3.1 Introduzione al linguaggio MATLAB

L'introduzione al linguaggio di programmazione MATLAB sarà molto rapido. Si assume dunque che l'interfaccia grafica sia familiare e che concetti base di programmazione (per esempio "che cos'è una variabile?") siano ben noti.

In MATLAB, l'assegnazione di scalari a delle variabili è classica, quindi si utilizza il simbolo uguale: `a = 1` (assegnazione del valore 1 alla variabile `a`). Inoltre, il linguaggio è *case sensitive*, di conseguenza la variabile `a` è diversa dalla variabile `A`. Alcuni comandi utili e generali:

- `help nome-comando`, per avere informazioni in più riguardo al comando `nome-comando`;
- `clear nome-variabile`, per rimuovere la variabile `nome-variabile` dalla memoria. Se non viene inserito il `nome-variabile`, vengono rimosse tutte le variabili dalla memoria.
- `who`, per visualizzare le variabili attualmente in memoria.
- `clc`, per ripulire la *Command Window*.

Argomento	Pagina
Well-known variables	Pag. 24
Cambiare il formato delle variabili: <code>format</code>	Pag. 25
Assegnamento di vettori e matrici	Pag. 26
Operazioni su vettori e matrici	Pag. 29
Funzioni intrinseche per vettori e matrici	Pag. 33
Funzioni matematiche elementari	Pag. 38
Funzioni per definire vettori o matrici particolari	Pag. 39

Tabella 1: Argomenti trattati.

### Well-known variables

Esistono alcune variabili che sono ben note e hanno valori prestabiliti. Tra le più importanti:

- `pi`, che rappresenta il  $\pi$  e MATLAB gli assegna il valore `3.1416`
- `i`, che rappresenta l'unità immaginaria e MATLAB gli assegna il valore `0.0000 + 1.0000i`
- `eps`, che rappresenta il più piccolo valore rappresentabile nel calcolatore (PC) attualmente in uso. Solitamente, `eps` ritorna il valore `2.2204e-16`.

Questo tipo di variabili possono essere ridefinite, ma non è una *good practice*.



## Cambiare il formato delle variabili: `format`

Il comando `format` è utilizzato per cambiare il formato con cui sono rappresentate le variabili. MATLAB non cambia la precisione della variabile (quindi non si ottiene una precisione maggiore dopo la virgola), ma modifica soltanto la rappresentazione. Di default MATLAB utilizza una rappresentazione di tipo `short`. Tra i più utilizzati (di default `pi` è uguale a `3.1416`):

- `default` per reimpostare la rappresentazione di default.
- Decimale:

– `short`, rappresentazione a 5 cifre:

```
1 >> format short
2 >> pi
3
4 ans =
5     3.1416
```

– `long`, rappresentazione a 15 cifre:

```
1 >> format long
2 >> pi
3
4 ans =
5     3.141592653589793
```

- *Floating point*:

– `short e`, rappresentazione a 5 cifre floating point:

```
1 >> format short e
2 >> pi
3
4 ans =
5     3.1416e+00
```

– `long e`, rappresentazione a 15 cifre floating point:

```
1 >> format long e
2 >> pi
3
4 ans =
5     3.141592653589793e+00
```

Altri formati si possono trovare nella [documentazione ufficiale](#).

## Assegnamento di vettori e matrici

- **Vettore riga**, si può creare utilizzando uno spazio tra i valori o una virgola ,:

```
1 >> a = [1 2 3 4]
2
3 a =
4     1     2     3     4
5
6 >> b = [1, 2, 3, 4]
7
8 b =
9     1     2     3     4
```

- **Vettore colonna**, si crea usando il punto e virgola ;:

```
1 >> a = [1; 2; 3; 4]
2
3 a =
4     1
5     2
6     3
7     4
```

Talvolta può essere utile la generazione automatica di un vettore riga (sono ammessi anche i valori negativi e con la virgola ovviamente):

- **Vettore riga generato linearmente**, si crea usando i due punti e specificando il valore di inizio e il valore di fine:

```
1 >> a = [1 : 4]
2
3 a =
4     1     2     3     4
```

- **Vettore riga generato usando un passo**, si crea usando i due punti e specificando (in ordine) il valore di inizio, il “salto”, e il valore di fine. Nel caso in cui il salto sia troppo grande e si superi il valore di fine, MATLAB prenderà il primo valore ammissibile:

```
1 >> % Generazione con passo 1
2 >> a = [1 : 1 : 4]
3
4 a =
5     1     2     3     4
6
7 >> % Generazione con passo 2
8 >> a = [1 : 2 : 5]
9
10 a =
11     1     3     5
12
13 >> % Generazione con passo 2 (fine non raggiunta)
14 >> a = [1 : 2 : 6]
15
16 a =
17     1     3     5
18
19 >> % ... ma cambiando l'upper bound
```

```

20 >> a = [1 : 2 : 7]
21
22 a =
23     1     3     5     7

```

- **Vettore riga generato con valori uniformemente distanziati**, si crea usando la funzione `linspace`, la quale accetta tre parametri:

- `x1`, valore di partenza.
- `x2`, valore di fine.
- `n`, numero di valori da generare; se non specificato, di default è 100; se il valore inserito è zero o minore, viene creato un vettore vuoto.

```

1 >> % Vettore riga identico a: a = [1 : 4]
2 >> linspace(1, 4, 4)
3
4 ans =
5     1     2     3     4
6
7 >> % Chiedendo piu' valori, MATLAB andra' ad utilizzare i
   decimali
8 >> linspace(1, 4, 6)
9 ans =
10
11     1.000000000000000e+00     1.600000000000000e+00
12     2.200000000000000e+00     2.800000000000000e+00
13     3.400000000000000e+00     4.000000000000000e+00

```

Le matrici possono essere create a mano o usando la combinazione delle tecniche viste in precedenza:

- **Matrice**, le righe si creano usando gli spazi e le colonne si creano usando il punto e virgola:

```

1 >> a = [1 2 3 4; 5 6 7 8; 9 10 11 12]
2
3 a =
4     1     2     3     4
5     5     6     7     8
6     9    10    11    12
7
8 >> a = [1, 2, 3, 4; 5, 6, 7, 8; 9, 10, 11, 12]
9
10 a =
11     1     2     3     4
12     5     6     7     8
13     9    10    11    12

```

- **Matrice creata usando la generazione lineare dei vettori**, si possono utilizzare le tecniche precedenti e i punti e virgola:

```

1 >> % Usando a = [x : y]
2 >> a = [1 : 4; 5 : 8; 9 : 12]
3
4 a =
5     1     2     3     4
6     5     6     7     8
7     9    10    11    12
8

```

```
9 >> % Usando a = [x : y : z]
10 >> a = [1 : 2 : 7; 9 : 2 : 15; 17 : 2 : 23]
11 a =
12
13     1     3     5     7
14     9    11    13    15
15    17    19    21    23
16
17 >> % Usando linspace
18 >> a = [linspace(1, 4, 4); linspace(5, 8, 4); linspace(9, 12,
19         4)]
20 a =
21     1     2     3     4
22     5     6     7     8
23     9    10    11    12
```

## Operazioni su vettori e matrici

- **Trasposizione**, la classica operazione eseguita con le matrici o vettori, si esegue con la keyword `'` oppure usando la funzione `transpose`:

```
1 >> a = [1 2 3 4]
2
3 a =
4     1     2     3     4
5
6 >> a'
7
8 ans =
9     1
10    2
11    3
12    4
13
14 >> transpose(a)
15
16 ans =
17     1
18     2
19     3
20     4
```

- **Somma e sottrazione**

– Tra vettore e scalare:

```
1 >> a = [1 2 3 4]
2
3 a =
4     1     2     3     4
5
6 >> a + 1
7
8 ans =
9     2     3     4     5
10
11 >> a - 1
12
13 ans =
14     0     1     2     3
```

– Tra vettore e matrice:

```
1 >> a = [1 2 3 4]
2
3 a =
4     1     2     3     4
5
6 >> b = [1 2 3 4; 5 6 7 8; 9 10 11 12]
7
8 b =
9     1     2     3     4
10     5     6     7     8
11     9    10    11    12
12
13 >> a + b
14
15 ans =
```

```

16      2      4      6      8
17      6      8     10     12
18     10     12     14     16
19
20 >> a - b
21
22 ans =
23      0      0      0      0
24     -4     -4     -4     -4
25     -8     -8     -8     -8

```

— Tra matrice e scalare:

```

1 >> b = [1 2 3 4; 5 6 7 8; 9 10 11 12]
2
3 b =
4      1      2      3      4
5      5      6      7      8
6      9     10     11     12
7
8 >> b + 1
9
10 ans =
11      2      3      4      5
12      6      7      8      9
13     10     11     12     13
14
15 >> b - 1
16
17 ans =
18      0      1      2      3
19      4      5      6      7
20      8      9     10     11

```

— Tra matrice e matrice:

```

1 >> b = [1 2 3 4; 5 6 7 8; 9 10 11 12]
2
3 b =
4      1      2      3      4
5      5      6      7      8
6      9     10     11     12
7
8 >> c = [13 14 15 16; 17 18 19 20; 21 22 23 24]
9
10 c =
11     13     14     15     16
12     17     18     19     20
13     21     22     23     24
14
15 >> b + c
16
17 ans =
18     14     16     18     20
19     22     24     26     28
20     30     32     34     36
21
22 >> b - c
23
24 ans =
25    -12    -12    -12    -12
26    -12    -12    -12    -12
27    -12    -12    -12    -12

```

- Prodotto

- Prodotto matriciale:

```
1 >> b = [1 2 3 4; 5 6 7 8; 9 10 11 12]
2
3 b =
4     1     2     3     4
5     5     6     7     8
6     9    10    11    12
7
8 >> c = [13 14 15; 16 17 18; 19 20 21; 22 23 24]
9
10 c =
11    13    14    15
12    16    17    18
13    19    20    21
14    22    23    24
15
16 >> b * c
17
18 ans =
19    190    200    210
20    470    496    522
21    750    792    834
```

- Prodotto punto per punto, in MATLAB è possibile moltiplicare ogni cella di una matrice (o vettore) per la corrispondente cella della matrice (o vettore) moltiplicata. La keyword utilizzata è `.*`:

```
1 >> b = [1 2 3 4; 5 6 7 8; 9 10 11 12]
2
3 b =
4     1     2     3     4
5     5     6     7     8
6     9    10    11    12
7
8 >> c = [13 14 15 16; 17 18 19 20; 21 22 23 24]
9
10 c =
11    13    14    15    16
12    17    18    19    20
13    21    22    23    24
14
15 >> b .* c
16
17 ans =
18     13     28     45     64
19     85    108    133    160
20    189    220    253    288
21
22 >> d = [1 2 3 4]
23
24 d =
25     1     2     3     4
26
27 >> b .* d
28
29 ans =
30     1     4     9    16
31     5    12    21    32
32     9    20    33    48
```

- **Potenza**

- **Potenza matriciale:**

```
1 >> b = [1 2 3; 4 5 6; 7 8 9]
2
3 b =
4     1     2     3
5     4     5     6
6     7     8     9
7
8 >> b^2
9
10 ans =
11     30     36     42
12     66     81     96
13    102    126    150
```

- **Potenza punto per punto**, come per il prodotto, è possibile elevare al quadrato ogni valore della matrice (o vettore):

```
1 >> b = [1 2 3; 4 5 6; 7 8 9]
2
3 b =
4     1     2     3
5     4     5     6
6     7     8     9
7
8 >> b.^2
9
10 ans =
11     1     4     9
12    16    25    36
13    49    64    81
```



## Funzioni intrinseche per vettori e matrici

Qua di seguito si elencano le funzioni più importanti da utilizzare per i vettori e le matrici.

- **size**, restituisce la dimensione del vettore o della matrice nel formato *righe colonne*. Specificando anche un valore (o vettore) come parametro, la funzione restituisce la dimensione (un vettore contenente le dimensioni richieste) nella “dimensione” richiesta:

```
1 >> a = [1 2 3 4]
2
3 a =
4     1     2     3     4
5
6 >> size(a)
7
8 ans =
9     1     4
10
11 >> size(a, 2)
12 ans =
13
14     4
15
16 >> b = [1 2 3; 4 5 6; 7 8 9]
17
18 b =
19     1     2     3
20     4     5     6
21     7     8     9
22
23 >> size(b)
24
25 ans =
26     3     3
27
28 >> size(b, [2, 3])
29
30 ans =
31     3     1
```

- **length**, restituisce la lunghezza del vettore e per le matrici restituisce il numero degli elementi per ogni riga:

```
1 >> a = [1 2 3 4]
2
3 a =
4     1     2     3     4
5
6 >> length(a)
7
8 ans =
9     4
10
11 >> b = [1 2 3 4 5 6; 7 8 9 10 11 12]
12
13 b =
14     1     2     3     4     5     6
15     7     8     9    10    11    12
16
```

```
17 >> length(b)
18
19 ans =
20     6
```

- **max**, **min**, calcolano rispettivamente il massimo e il minimo valore delle componenti di un vettore; per le matrici viene presa in considerazione ogni colonna e calcolato il massimo o minimo:

```
1 >> a = [ 1 2 3 4]
2
3 a =
4     1     2     3     4
5
6 >> max(a)
7
8 ans =
9     4
10
11 >> min(a)
12
13 ans =
14     1
15
16 >> b = [7 1 1 4; 2 3 9 10; 8 1 7 1]
17
18 ans =
19     7     1     1     4
20     2     3     9    10
21     8     1     7     1
22
23 >> max(b)
24
25 ans =
26     8     3     9    10
27
28 >> min(b)
29
30 ans =
31     2     1     1     1
```

- **sum**, **prod**, calcola rispettivamente la somma e il prodotto degli elementi che compongono il vettore; nel caso di una matrice, viene presa in considerazione ogni colonna e calcolata la somma o il prodotto. Inoltre, i due comandi possono prendere un argomento in più per eseguire il calcolo in una dimensione specifica (cosa sensata con le matrici):

```
1 >> a = [ 1 2 3 4]
2
3 a =
4     1     2     3     4
5
6 >> sum(a)
7
8 ans =
9    10
10
11 >> prod(a)
12
13 ans =
14    24
```

```

15
16 >> b = [7 1 1 4; 2 3 9 10; 8 1 7 1]
17
18 b =
19      7      1      1      4
20      2      3      9     10
21      8      1      7      1
22
23 >> sum(b) % per colonne
24
25 ans =
26     17      5     17     15
27
28 >> sum(b, 2) % per righe
29
30 ans =
31     13
32     24
33     17
34
35 >> prod(b)
36
37 ans =
38    112      3     63     40

```

- **norm**, la norma di un vettore o di una matrice. Passando un vettore o un matrice, viene calcolata di default la norma euclidea (norma 2):

$$\|v\|_2 = \sqrt{\sum_{i=2}^{\text{length}(v)} v_i^2}$$

Passando un valore aggiuntivo, esso rappresenterà l'ordine della norma:

$$\|v\|_n = \left( \sum_{i=2}^{\text{length}(v)} |v_i|^n \right)^{\frac{1}{n}}$$

Infine, con **inf** viene calcolata la norma infinito:

$$\|v\|_\infty = \max_{1 \leq i \leq \text{length}(v)} |v_i|$$

```

1 >> a = [1 2 3 4]
2
3 a =
4      1      2      3      4
5
6 >> norm(a)
7
8 ans =
9     5.477225575051661e+00
10
11 >> norm(a, 2)
12
13 ans =
14     5.477225575051661e+00
15
16 >> norm(a, 3)

```

```
17
18 ans =
19     4.641588833612779e+00
20
21 >> norm(a, inf)
22
23 ans =
24     4
25
26 >> b = [7 1 1 4; 2 3 9 10; 8 1 7 1]
27
28 b =
29     7     1     1     4
30     2     3     9    10
31     8     1     7     1
32
33 >> norm(b, 2)
34
35 ans =
36     1.711222312384884e+01
37
38 >> norm(b, inf)
39
40 ans =
41    24
```

- **abs**, rappresenta il valore assoluto e restituisce il vettore o matrice dopo aver applicato il valore assoluto a ciascun elemento:

```
1 >> a = [-1 -2 -3 -4]
2
3 a =
4    -1    -2    -3    -4
5
6 >> abs(a)
7
8 ans =
9     1     2     3     4
10
11 >> b = [7 1 1 -4; 2 3 -9 10; 8 1 -7 1]
12 b =
13
14     7     1     1    -4
15     2     3    -9    10
16     8     1    -7     1
17
18 >> abs(b)
19
20 ans =
21     7     1     1     4
22     2     3     9    10
23     8     1     7     1
```

- **diag**, estrae la diagonale di una matrice esistente, oppure ne crea una con i valori dati come input. Inoltre, può creare una matrice con la diagonale spostata a seconda del valore dato (si veda l'esempio):

```
1 >> b = [7 1 1 4; 2 3 9 10; 8 1 7 1]
2
3 b =
4     7     1     1     4
5     2     3     9    10
6     8     1     7     1
7
8 >> diag(b)
9
10 ans =
11     7
12     3
13     7
14
15 >> diag(b, 1)
16
17 ans =
18     1
19     9
20     1
21
22 >> diag(b, -1)
23
24 ans =
25
26     2
27     1
28
29 >> diag([1 2 3])
30
31 ans =
32     1     0     0
33     0     2     0
34     0     0     3
35
36 >> diag([1 2 3], -1)
37
38 ans =
39     0     0     0     0
40     1     0     0     0
41     0     2     0     0
42     0     0     3     0
```

## Funzioni matematiche elementari

Qua di seguito una lista di alcune funzioni matematiche elementari. Gli esempi e la sintassi non verranno mostrati poiché è sempre la medesima:

*funzione(parametro)*

Funzione	Comando
Radice quadrata	sqrt
Esponenziale	exp
Logaritmo Naturale	log
Logaritmo In Base 2	log2
Logaritmo In Base 10	log10
Seno	sin
Arcoseno	asin
Coseno	cos
Arcocoseno	acos
Tangente	tan

Tabella 2: Funzioni matematiche elementari.

## Iterazione con il ciclo for

In MATLAB il ciclo for viene eseguito con la seguente sintassi.

```
1 for index = values
2     statements
3 end
```

Di seguito si riporta un ciclo for che itera sulla diagonale secondaria di una matrice:

```
1 >> b = [7 1 1 4; 2 3 9 10; 8 1 7 1]
2
3 b =
4     7     1     1     4
5     2     3     9    10
6     8     1     7     1
7
8 >> res = []
9
10 res =
11     []
12
13 >> for i = 1 : size(b, 1)
14     res(i) = b(i, size(b, 1) - i + 1);
15 end
16
17 >> res
18
19 res =
20     1     3     8
```

## Funzioni per definire vettori o matrici particolari

In queste pagine vengono presentate alcune funzioni utili che consentono di creare matrici o vettori “particolari”.

- **Vettore/Matrice nulla**, con la funzione **zeros** è possibile creare una matrice o un vettore di tutti zeri. I parametri ammessi corrispondono alla dimensione del vettore o matrice:

```
1 >> zeros(1, 4)
2
3 ans =
4      0      0      0      0
5
6 >> zeros(4, 1)
7
8 ans =
9      0
10     0
11     0
12     0
13
14 >> zeros(4, 4)
15
16 ans =
17      0      0      0      0
18      0      0      0      0
19      0      0      0      0
20      0      0      0      0
```

- **Vettore unario/Matrice unaria**, con la funzione **ones** è possibile creare una matrice o un vettore di tutti uni. I parametri ammessi corrispondono alla dimensione del vettore o matrice:

```
1 >> ones(1, 4)
2
3 ans =
4      1      1      1      1
5
6 >> ones(4, 1)
7
8 ans =
9      1
10     1
11     1
12     1
13
14 >> ones(4, 4)
15
16 ans =
17      1      1      1      1
18      1      1      1      1
19      1      1      1      1
20      1      1      1      1
```

- **Matrice identità**, con la funzione `eye` è possibile creare una matrice identità. I parametri ammessi corrispondono alla dimensione del vettore o matrice:

```

1 >> eye(3)
2
3 ans =
4     1     0     0
5     0     1     0
6     0     0     1
7
8 >> eye(2, 3)
9
10 ans =
11     1     0     0
12     0     1     0
13
14 >> eye(1, 4)
15
16 ans =
17     1     0     0     0

```

- **Matrice/Vettore riga di numeri casuali interi e non**, con il comando `rand` si genera una matrice di numeri casuali nell'intervallo  $[0, 1]$  con la virgola, mentre con il comando `randi` si genera una matrice di numeri casuali interi (primo parametro deve essere specificato il range dei valori):

```

1 >> rand(3, 5)
2
3 ans =
4     0.9157     0.6557     0.9340     0.7431     0.1712
5     0.7922     0.0357     0.6787     0.3922     0.7060
6     0.9595     0.8491     0.7577     0.6555     0.0318
7
8 >> % Matrice di valori interi random da 1 a 5
9 >> randi([1, 5], 3, 4)
10
11 ans =
12     2     5     5     2
13     1     4     1     4
14     1     2     3     4
15
16 >> % Errore! L'intervallo e' sbagliato
17 >> randi([-1, -50], 3, 4)
18 Error using randi
19 First input must be a positive scalar integer value IMAX, or
    two integer values [IMIN IMAX] with IMIN less than or
    equal to IMAX.
20
21 >> randi([-50, -1], 3, 4)
22
23 ans =
24    -41    -18    -37    -42
25    -26    -15    -17    -45
26    -28    -13    -18    -26

```



### 3.1.1 Esercizio

Creare una funzione (file) chiamato `mat_hilbert.m` che fornisca la matrice di Hilbert avente una generica dimensione `n`. Ogni cella della matrice di Hilbert deve rispettare la seguente condizione:

$$a_{ij} = \frac{1}{i + j - 1}$$

Dopo aver creato la funzione, utilizzare la funzione nativa di MATLAB `hilb`, per verificare il risultato ottenuto.

### Soluzione

Il codice non ha bisogno di grandi spiegazioni. Vi è un controllo iniziale per verificare l'argomento inserito dall'utente e successivamente due cicli `for` per popolare la matrice:

```

1 function hilbert_matrix = mat_hilbert(n)
2
3 if n < 0
4     error("n can't be 0 or less than zero")
5 end
6
7 hilbert_matrix = zeros(n);
8 for i = 1 : n
9     for j = 1 : n
10        hilbert_matrix(i, j) = 1 / (i + j - 1);
11    end
12 end

```

Il risultato:

```

1 mat_hilbert(5)
2
3 ans =
4
5     1.0000     0.5000     0.3333     0.2500     0.2000
6     0.5000     0.3333     0.2500     0.2000     0.1667
7     0.3333     0.2500     0.2000     0.1667     0.1429
8     0.2500     0.2000     0.1667     0.1429     0.1250
9     0.2000     0.1667     0.1429     0.1250     0.1111
10
11 hilb(5)
12
13 ans =
14
15     1.0000     0.5000     0.3333     0.2500     0.2000
16     0.5000     0.3333     0.2500     0.2000     0.1667
17     0.3333     0.2500     0.2000     0.1667     0.1429
18     0.2500     0.2000     0.1667     0.1429     0.1250
19     0.2000     0.1667     0.1429     0.1250     0.1111

```

## 3.2 Zeri di funzione

### 3.2.1 Grafici di funzione

In MATLAB una funzione  $f(x)$  viene memorizzata come un vettore. In particolare, il vettore  $y$  ottenuto valutando  $f$  nel vettore delle ascisse  $x$ . Per cui la rappresentazione della funzione  $f(x)$  è di fatto la rappresentazione del vettore  $y$  contro il vettore  $x$ .

Per introdurre i concetti di funzione e grafici di funzione, si presentano qua di seguito alcuni esempi di caso d'uso.

*Definire le seguenti variabili:*

- $x$ : vettore di estremi 0 e 10 con passo 0.1
- $y = e^x + 1$

Il vettore delle ascisse  $x$  può essere costruito banalmente con il seguente costrutto:

```
1 x = [0 : 0.1 : 10];
```

Per quanto riguarda la **funzione**, si utilizza la keyword `@` per indicare che  $f$  ha come input un valore ( $x$ ) e rappresenta la funzione  $\exp(x)+1$ . In questo caso, la funzione si dice anonima. Per dichiarare funzioni esplicite, si rimanda alla [documentazione ufficiale](#).

```
1 f = @(x) exp(x) + 1
```

Una volta definita una **funzione**, per **valutarla in uno o più punti**, si utilizzerà banalmente la sintassi matematica:

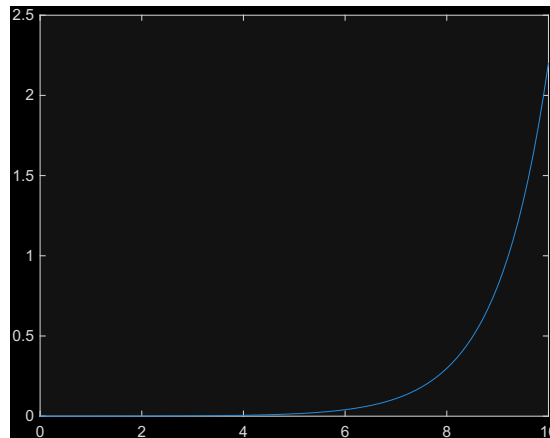
```
1 f(2)
2
3 ans =
4
5     8.3891
6
7 f(0:3)
8
9 ans =
10
11     2.0000     3.7183     8.3891    21.0855
```

Da notare che se l'argomento è un vettore, allora il risultato sarà un vettore della medesima lunghezza del vettore dato in input.

*Utilizzando le variabili precedentemente definite, disegnare il grafico della funzione  $y = e^x + 1$  nell'intervallo  $[0, 10]$ .*

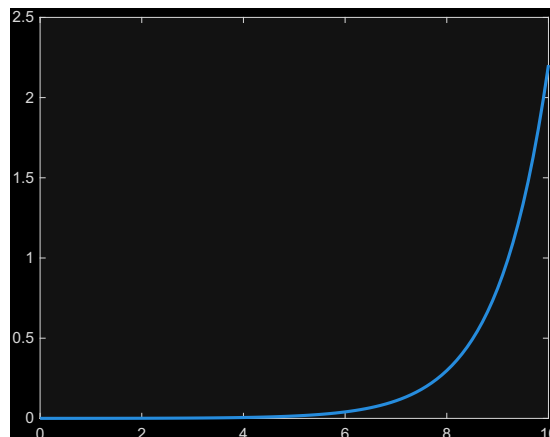
Per disegnare il grafico si utilizza il comando `plot`. Di default questa funzione disegna i valori in un piano cartesiano usando segmenti rettilinei (retta spezzata):

```
1 y = f(x);  
2 plot(x, y)
```



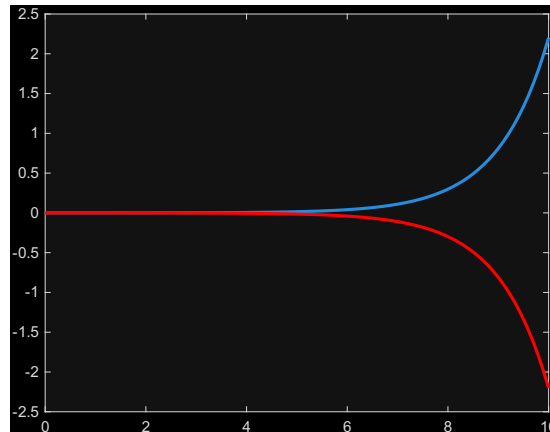
Per evitare che MATLAB sovrascriva la figura nella finestra aperta, è possibile numerarle usando la funzione `figure` (e.g. `figure(1); plot(x,y); figure(2); plot(0:3, 0:3)`).

La funzione `plot` accetta determinati valori per modificare il grafico finale. Nella [documentazione ufficiale](#) è possibile trovare l'intera lista e alcuni esempi. Scrivendo `plot(x, f(x), 'linewidth', 2)`, il parametro `'linewidth'` consente di definire lo spessore delle curve. Il valore che viene specificato in questo caso è 2 e il risultato:



Usando il comando `hold on` per fare un confronto tra i vari grafici e invocando di nuovo la funzione `plot` ma con parametri differenti, si ottiene:

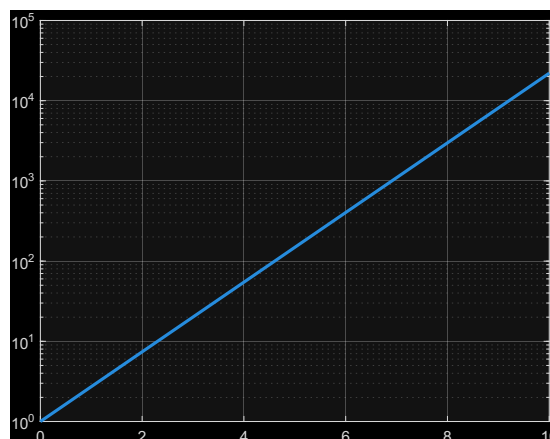
```
1 figure(1)
2 plot(x, f(x), 'linewidth', 2)
3 hold on
4 plot(x, -f(x), 'r', 'linewidth', 2)
```



*Disegnare il grafico in scala semi-logaritmica (logaritmica solo per le ordinate) della funzione  $y = e^x$  nell'intervallo  $[0, 10]$ . È possibile prevedere come sarà il grafico in scala semi-logaritmica della funzione  $y = e^{2x}$ ? Verificare la risposta tracciando sulla medesima finestra le due funzioni utilizzando colori diversi per i due grafici.*

Per disegnare il grafico in scala semi-logaritmica (logaritmica sulle ordinate) si utilizza il comando `semilogy` e si aggiunge anche la griglia:

```
1 semilogy(x, exp(x), 'linewidth', 2)
2 grid on
```



È una retta poiché  $\log_{10}(y) = \log_{10}(e^x) = x \log_{10}(e)$ .

- Il comando `semilogy` è l'equivalente di `plot` ma traccia un **grafico con l'asse delle ordinate in scala logaritmica**.
- Il comando `semilogx` traccia un **grafico con l'asse delle ascisse logaritmico**.
- Il comando `loglog` traccia un grafico in cui entrambi gli assi sono in scala logaritmica.

Passando alla risoluzione dell'esercizio, dato che  $\log_{10}(e^{2x}) = 2x \log_{10}(e)$ , disegnando in scala semi-logaritmica la funzione  $y = e^{2x}$ , si otterrà una retta con pendenza doppia rispetto alla retta precedentemente disegnata.

```
1 hold on
2 semilogy(x, exp(2 * x), 'r', 'linewidth', 2)
3 % oppure in un solo comando senza usare hold on
4 % semilogy(x, exp(x), 'b', x, exp(2*x), 'r', 'linewidth', 2)
5 title('Grafico di exp(x) e di exp(2x)')
6 xlabel('Scala lineare')
7 ylabel('Scala logaritmica')
8 grid on
9 legend('exp(x)', 'exp(2*x)', 'Location', 'NorthWest')
```



Il comando `legend` attribuisce alle curve disegnate da `plot` le stringhe di testo che gli vengono passate. Attenzione che alcune stringhe, come `'Location'` e `'NorthWest'`, vengono interpretate dalla funzione come comandi veri e propri. In questo caso si chiede di inserire una legenda in alto a sinistra.

### 3.3 Risoluzione di Sistemi di Equazioni Lineari

#### 3.3.1 Metodi diretti

Si consideri la matrice di dimensione  $n \times n$ :

$$A = \begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & -1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & -1 & 0 & \cdots & 0 \\ \vdots & 0 & \ddots & \ddots & & \vdots \\ \vdots & \vdots & & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & -1 \end{bmatrix}$$

E  $\mathbf{b}$  il vettore di dimensione  $n$ :

$$\mathbf{b} = [2, 0, 0, \dots, 0]^T$$

1. Si ponga  $n = 20$  e si assegnino in MATLAB la matrice  $A$  e il vettore dei termini noti  $\mathbf{b}$ .

```

1 n = 20;
2 % crea il vettore colonna composto da soli uno
3 R = ones(n, 1);
4 % crea una matrice di valori negativi (-1)
5 % sulla diagonale principale
6 % e sommala alla matrice creata come:
7 % - vettore R specificando il range per evitare
8 % una matrice troppo grande;
9 % - -1 per indicare un livello sotto la diagonale principale:
10 A = -diag(R) + diag(R(1:n-1), -1);
11 % si riempi la prima riga della matrice A di uni
12 A(1, :) = 1;
13 % si crea un altro vettore di zeri
14 b = zeros(n, 1);
15 % e si sostituisce il primo valore con un 2
16 b(1) = 2;
```

La funzione `diag` ha un parametro particolare, [vedi la documentazione](#).

2. Si calcoli la fattorizzazione LU della matrice  $A$ , mediante la funzione MATLAB `lu`. Verificare che la tecnica del pivoting non è stata usata in questo caso.

```

1 % la funzione lu puo' essere utilizzata nel seguente modo
2 [L, U, P] = lu(A);
3 % in cui L ed U sono la fattorizzazione,
4 % mentre la matrice P indica la matrice permutazione.
5 % quest'ultima potrebbe avere delle permutazioni sulle righe
6 % dovute alla tecnica del pivoting.
7 % per controllare si puo' procedere controllando manualmente,
8 % quindi stampando la matrice P e controllare che sia
9 % una matrice identita',
10 % oppure invocare la funzione eye e verificare che siano
11 % uguali con un if statement
12 if P == eye(n)
13     disp('pivoting effettuato!');
14 end
```

Si veda a pagina 21 la spiegazione della matrice di permutazione.

3. Scrivere una funzione MATLAB `fwsb.m` che, dati in ingresso una matrice triangolare inferiore  $L \in \mathbb{R}^{n \times n}$  e un vettore  $\mathbf{f} \in \mathbb{R}^n$  restituisca in uscita il vettore  $\mathbf{x}$ , soluzione del sistema  $L\mathbf{x} = \mathbf{f}$ , calcolata mediante l'algoritmo della sostituzione in avanti (*forward substitution*). L'intestazione della funzione sarà ad esempio: `[x] = fwsb(L, f)`.

Analogamente, scrivere la funzione `bksb.m` che implementi l'algoritmo della sostituzione all'indietro (*backward substitution*) per matrici triangolari superiori ( $U$ ). Per controllare che le matrici  $L$  e  $U$  passate a `fwsb.m` e `bksb.m` siano effettivamente triangolari, è possibile utilizzare i comandi MATLAB `triu` e `tril` che, data una matrice, estraggono rispettivamente la matrice triangolare superiore e la matrice triangolare inferiore.

Per creare una funzione, in MATLAB viene utilizzata la seguente sintassi:

```
1 function output_params = function_name(input_params)
2     % Statements
3 end
```

Introdotta la sintassi, si introduce il codice della funzione `fwsb.m`:

```
1 % si dichiara la funzione fwsb, che ha come input A e b
2 % e restituisce come output x
3 function x = fwsb(A,b)
4
5     % ~ algoritmo di sostituzione in avanti ~
6     % A: matrice quadrata triangolare inferiore
7     % b: termine noto
8     % x: soluzione del sistema Ax=b
9
10    % si controlla che la matrice sia quadrata e
11    % per farlo si calcola le dimensioni di b
12    n = length(b);
13    % se il numero di righe (size(A,1)) della matrice A
14    % o se il numero di colonne (size(A,2)) della matrice A
15    % sono diverse da n, allora le dimensioni non sono ammesse
16    if (size(A, 1) ~= n || size(A, 2) ~= n)
17        error("Dimensioni non ammesse");
18    end
19
20    % inoltre si controlla che la matrice sia una matrice
21    % triangolare inferiore;
22    % si utilizza la funzione tril per ottenere la
23    % matrice triangolare inferiore
24    if (A ~= tril(A))
25        error("La matrice non e' triangolare inferiore");
26    end
27
28    % infine, si controlla che la matrice sia NON singolare,
29    % ovvero che il determinante deve essere diverso da zero
30    if (det(A) == 0)
31        error("La matrice e' singolare");
32    end
33
34    % adesso l'algoritmo puo' iniziare;
35    % si inizializza una matrice risultato, ovvero x,
36    % nella quale verranno salvati i risultati
37    x = zeros(n,1);
38    % si applica la formula della sostituzione in avanti,
39    % prima per la posizione (1,1)
40    x(1) = b(1) / A(1,1);
41    % e dopodiche' per tutte le posizioni della matrice
```

```

42     for i = 2:n
43         x(i) = (b(i) - A(i, 1:i-1) * x(1:i-1)) / A(i,i);
44     end

```

Analogamente, si presenta il codice della funzione `bksub.m`:

```

1  % la funzione bksub avra' la stessa signature della funzione
2  % fwsb, ma la formula chiaramente sara' differente
3  function x = bksub(A, b)
4
5      % ~ algoritmo di sostituzione all'indietro ~
6      % A: matrice quadrata triangolare superiore
7      % b: termine noto
8      % x: soluzione del sistema Ax = b
9
10     % 1. si esegue lo stesso controllo della funzione
11     % fwsb, si verifica che A sia quadrata
12     n = length(b);
13     if (size(A, 1) ~= n || size(A, 2) ~= n)
14         error("Dimensioni non ammesse");
15     end
16
17     % 2. si controlla che sia effettivamente una
18     % matrice triangolare superiore,
19     % usando questa volta la funzione triu
20     if (A ~= triu(A))
21         error("La matrice non e' triangolare superiore");
22     end
23
24     % 3. l'ultimo controllo riguarda la "non singolarita'"
25     % ovvero, determinante diverso da zero
26     if (det(A) == 0)
27         error("La matrice e' singolare");
28     end
29
30     % 4. si parte con l'algoritmo e per farlo si inizia
31     % con l'ultima posizione;
32     % ovviamente si inizializza la matrice risultato
33     x = zeros(n,1);
34     x(n) = b(n) / A(n,n);
35     % 5. si ricorda la sintassi del for statement
36     % initVal : step : endVal
37     % l'indice parte con un valore uguale a initVal,
38     % incrementa o decrementa a seconda dello step,
39     % termina quando raggiunge la condizione endVal
40     for i = n-1 : -1 : 1
41         x(i) = (b(i) - A(i, i+1:n) * x(i+1:n)) / A(i,i);
42     end

```

4. Risolvere numericamente, utilizzando le funzioni `fwsb.m` e `bksub.m` implementate al punto precedente, i due sistemi triangolari necessari per ottenere la soluzione del sistema di partenza  $Ax = b$  mediante la fattorizzazione LU.

Si utilizza la tecnica del pivoting e l'equazione 27 a pagina 21:

```

1  y = fwsb(L, P*b);
2  x = bksub(U, y);

```



5. Si calcoli la norma 2 dell'errore relativo

$$\|\mathbf{err}_{\text{rel}}\| = \frac{\|\mathbf{x} - \hat{\mathbf{x}}\|}{\|\mathbf{x}\|}$$

E la norma 2 del residuo normalizzata:

$$\|\mathbf{r}\| = \frac{\|\mathbf{b} - A\hat{\mathbf{x}}\|}{\|\mathbf{b}\|}$$

Sapendo che la soluzione esatta è il vettore di componenti:

$$\mathbf{x}(i) = \frac{2}{n} \quad i = 1, \dots, n$$

Si commenti il risultato ottenuto basandosi sul valore del numero di condizionamento della matrice  $A$  (si utilizzino i comandi `norm` e `cond`).

Il comando `norm` è stato spiegato a pagina 35.

```

1 x_ex = 2 / n * ones(n, 1);
2
3 err_rel = norm(x_ex - x) / norm(x_ex)
4 % il risultato: 5.1554e-16
5
6 r_nor = norm(b - A*x) / norm(b)
7 % il risultato: 2.7318e-16
8
9 K = cond(A)
10 % il risultato: 28.4998

```

6. Si ripeta il punto precedente per  $n = 10, 20, 40, 80, 160$ . Si rappresentino su un grafico in scala semi-logaritmica gli andamenti dell'errore relativo, del residuo normalizzato (si usa dire residuo normalizzato per la norma normalizzata del residuo) e del numero di condizionamento in funzione di  $n$ . Commentare il grafico ottenuto.

```

1 % si crea il vettore n
2 N = [10 20 40 80 160];
3 % e si inizializzano le variabili
4 K = [];
5 err_rel = [];
6 r_nor = [];
7
8 % per ogni n, si applicano i pezzi di codice precedenti
9 for n = N
10     R = ones(n, 1);
11     A = -diag(R) + diag(R(1:n-1), -1);
12     A(1, :) = 1;
13
14     b = zeros(n, 1);
15     b(1) = 2;
16
17     [L, U, P] = lu(A);
18
19     y = fwsb(L, P*b);
20     x_1 = bksub(U, y);
21
22     x_ex = 2 / n * ones(n, 1);
23     err_rel = [err_rel; norm(x_ex - x_1) / norm(x_ex)];
24     r_nor = [r_nor; norm(b - A*x_1) / norm(b)];

```

```

25     K = [K; cond(A)];
26 end
27
28 % Semilog plot (y-axis has log scale)
29 semilogy(N, err_rel, '-s', N, r_nor, '-o', N, K, '-x')
30 legend('errore rel.', 'residuo norm.', 'n. di condizionamento',
31        )
32 xlabel('dimensione n')
33 ylabel('err, r, K')
34 grid on

```

La seguente figura mostra l'andamento dell'errore relativo, del residuo normalizzato e del numero di condizionamento in funzione di  $n$ , in scala semi-logaritmica. Si noti che sia il residuo normalizzato sia l'errore relativo sono molto piccoli, dall'ordine di  $10^{-16}$ , conseguenza del fatto che il numero di condizionamento  $K(A)$  è in questo caso relativamente piccolo.

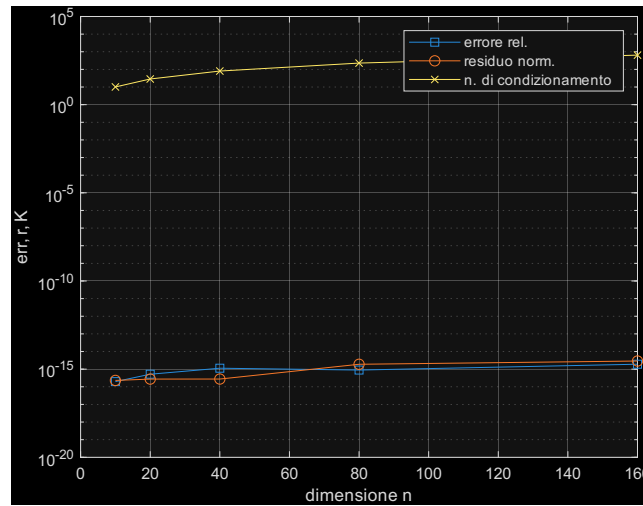


Figura 3: Andamento dell'errore relativo, del residuo normalizzato e del numero di condizionamento in funzione di  $n$ .

## Riferimenti bibliografici

- [1] A. Quarteroni, F. Saleri, and P. Gervasio. *Calcolo Scientifico: Esercizi e problemi risolti con MATLAB e Octave*. UNITEXT. Springer Milan, 2017.

## Index

### Symbols

$p$  super-lineare 10

### D

definita positiva 20

differenza fra due iterate consecutive 9

### F

fattore di convergenza 15

fattorizzazione di Cholesky 21

fattorizzazione LU 20

formula di Cramer 17

funzione di iterazione 13

### I

iterazioni di punto fisso 13

### M

matrice a dominanza diagonale per colonne 20

matrice a dominanza diagonale per righe 20

matrice di permutazione 21

matrice Jacobiana 11

matrici a dominanza diagonale stretta 20

metodo delle secanti 10

metodo delle sostituzioni all'indietro 19

metodo delle sostituzioni in avanti 18

metodo di bisezione 4

Metodo di Eliminazione di Gauss (MEG) 20

metodo di Newton 8

metodo iterativo 4

### P

pivoting per righe 21

punto fisso 13

### R

residuo 9

### T

teorema di Ostrowski 15