

Calcolo Numerico - Appunti

260236

agosto 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

1	Equazioni non lineari	4
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
2	Metodi risolutivi per sistemi lineari e non lineari	17
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
2.1.4	Errori di arrotondamento nel MEG	22
2.1.5	Il pivoting totale	26
2.1.6	Il <i>fill-in</i> di una matrice	27
2.2	Metodi iterativi per sistemi lineari	29
2.2.1	Il metodo di Jacobi	31
2.2.2	Il metodo di Gauss-Seidel	33
2.2.3	Il metodo di Richardson	34
2.2.4	Il metodo del Gradiente e del Gradiente Coniugato	36
2.3	Metodi numerici per sistemi non lineari	40
3	Laboratorio	41
3.1	Introduzione al linguaggio MATLAB	41
3.1.1	Esercizio	58
3.2	Zeri di funzione	59
3.2.1	Grafici di funzione	59
3.3	Risoluzione di Sistemi di Equazioni Lineari	63
3.3.1	Metodi diretti	63
3.3.2	Metodi iterativi	68
3.3.2.1	Metodo di Jacobi	68
3.3.2.2	Metodo di Gauss-Seidel	68
3.3.2.3	Esercizio	69
3.3.2.4	Metodo di Richardson	76
3.3.2.5	Precondizionamento	77
3.3.2.6	Metodo del gradiente	78
3.3.2.7	Esercizi su Richardson e gradiente	79
	Index	90

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 α 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 α 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 α . Questo poiché la successione $\{x^{(k)}\}$ dei punti medi dei sotto-intervalli $I^{(k)}$ dovrà ineluttabilmente convergere a α , 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 α 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 $a^{(2)} = x^{(1)} = 0.875$

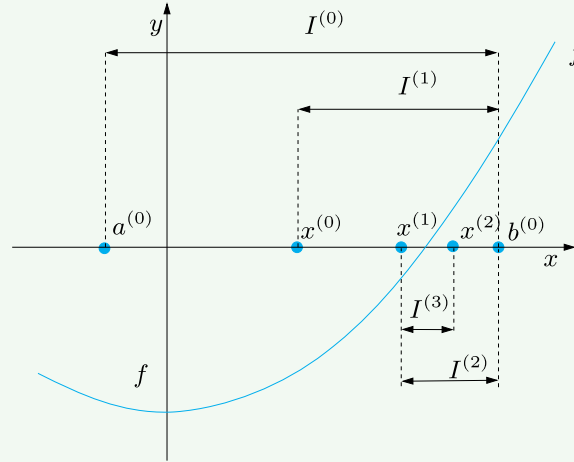
(f) Si pone $b^{(2)} = b^{(1)} = 1.25$

(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 α . Inoltre, la successione $\{x^{(k)}\}$ converge necessariamente allo zero α 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** ε , 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 α 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 α , ovvero **appartenenti ad un intorno $I(\alpha)$ sufficientemente piccolo** di α .

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 α 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 α 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 ε , 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_α dello zero α 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 α** , se:
 - α radice semplice¹;
 - $I(\alpha)$ è un opportuno intorno di α ;
 - $x^{(0)}$ e $x^{(1)}$ sono sufficientemente vicini ad α
 - $f'(x) \neq 0 \quad \forall x \in I(\alpha) \setminus \{\alpha\}$
- **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 α è multipla.

Come succederebbe usando il metodo di Newton.

¹ $f'(\alpha) \neq 0$

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 $x^{(0)} = 1$), α è tale che $\cos(\alpha) = \alpha$. Quindi, α viene detto punto fisso della funzione coseno.

? Perché è interessante?

Se α 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 α esiste, viene detto un **punto fisso** di ϕ 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 ϕ è detta **funzione di iterazione**.

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

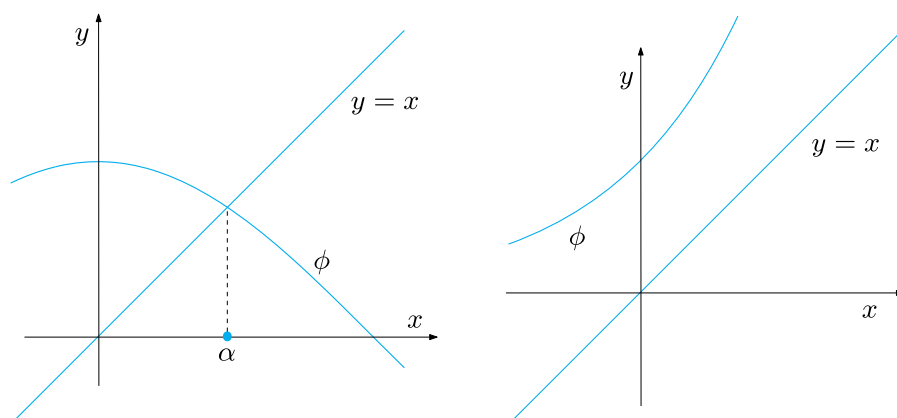


Figura 1: La funzione $\phi(x) = \cos(x)$ (sx) ammette un solo punto fisso, mentre la funzione $\phi(x) = e^x$ (dx) non ne ammette alcuno.

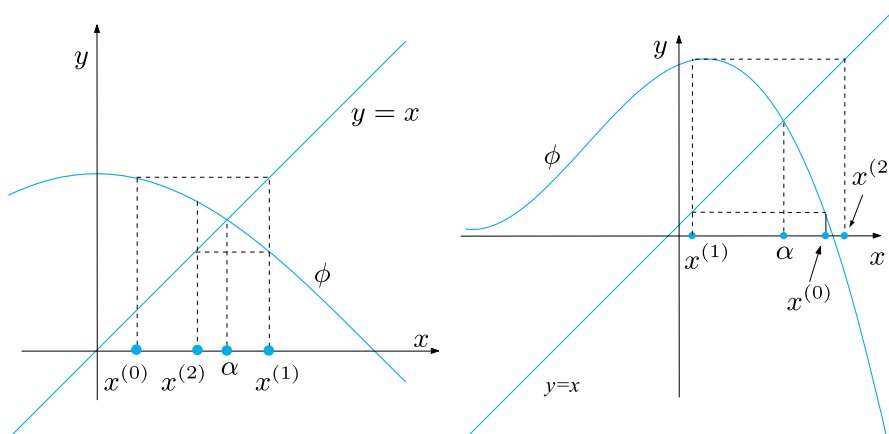


Figura 2: Rappresentazione delle prime iterazioni di punto fisso per due funzioni di iterazione. Le iterazioni convergono verso il punto fisso α (sx), mentre si allontanano da α (dx).

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 ϕ ha un **unico punto fisso** $\phi \in [a, b]$ e la successione definita nell'equazione 13 a pagina 13 converge a α , 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 α un punto fisso di una funzione ϕ continua e derivabile con continuità in un opportuno intorno \mathcal{I} di α . Se risulta $|\phi'(\alpha)| < 1$, allora esiste un $\delta > 0$ in corrispondenza del quale la successione $\{x^{(k)}\}$ converge ad α , 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, ϕ 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 α , 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:

$$U_{n \times n} = \begin{bmatrix} U_{11} & & & & \\ 0 & \ddots & & & U_{1j} \\ 0 & 0 & \ddots & & \\ 0 & 0 & 0 & \ddots & \\ 0 & 0 & 0 & 0 & U_{nn} \end{bmatrix}$$

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 simmetriche² e definite positive³**.

²Una matrice è simmetrica se coincide con la sua matrice trasposta.

³Una 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.

2.1.4 Errori di arrotondamento nel MEG

Prima di introdurre l'errore di arrotondamento nel Metodo di Eliminazione di Gauss, è necessario capire il problema alla fonte.

In generale, un elaboratore memorizza i numeri nel seguente modo:

$$x = (-1)^s \cdot (0.a_1a_2 \dots a_t) \cdot \beta^e = (-1)^s \cdot m \cdot \beta^{e-t} \quad \text{con } a_1 \neq 0 \quad (28)$$

Dove:

- s è il **segno** e può essere uguale a 0 o 1.
- β è la **base** e può essere un numero intero positivo maggiore od uguale a 2.
- m è la **mantissa**, un intero la cui **lunghezza** t è il **numero massimo di cifre** a_i (con $0 \leq a_i \leq \beta - 1$) **memorizzabili**.
- e è l'**esponente** ed è un intero.

I numeri con un formato identico all'equazione 28 sono detti numeri **floating-point normalizzati** essendo variabile la posizione del punto decimale. Inoltre, le cifre $a_1a_2 \dots a_p$ (con $p \leq t$) vengono generalmente chiamate le **prime p cifre significative** di x .

Si noti che la condizione $a_1 \neq 0$ nell'equazione 28 impedisce che lo stesso numero possa avere più rappresentazioni (per esempio $0.1 \cdot 10^0$ uguale a $0.01 \cdot 10^1$).

L'insieme \mathbb{F} è dunque l'**insieme dei numeri floating point** ed è completamente caratterizzato dalla base β , dal numero di cifre significative t e dall'intervallo (L, U) (con $L < 0$ ed $U > 0$) di variabilità dell'esponente e .

Sostituendo ad un numero reale $x \neq 0$ il suo rappresentante *floating point* $fl(x)$ in \mathbb{F} , è inevitabile un **errore di arrotondamento** uguale a:

$$\frac{|x - fl(x)|}{|x|} \leq \frac{1}{2} \epsilon_M \quad (29)$$

Dove:

- $\epsilon_M = \beta^{1-t}$ è la **epsilon macchina**, ovvero la distanza fra 1 ed il più piccolo numero *floating-point* maggiore di 1.
- $|x|$ è l'**errore relativo**.
- $|x - fl(x)|$ è l'**errore assoluto**.
- Il numero $u = \frac{1}{2} \epsilon_M$ è l'**unità di arrotondamento** poiché rappresenta il **massimo errore relativo che la macchina può commettere nella rappresentazione di un numero reale**.

Date queste osservazioni, si possono ricavare anche il **più grande** ed il **più piccolo numero positivo** di \mathbb{F} :

$$x_{\min} = \beta^{L-1} \quad x_{\max} = \beta^U \cdot (1 - \beta^{-t})$$

- Se un numero è minore del numero più piccolo positivo, allora si ha una situazione di **underflow**.
- Se un numero è maggiore del numero più grande positivo, allora si ha una situazione di **overflow**.

❓ **Quindi che cosa accade in una somma tra valori molto grandi?**

Ottima domanda. Quando si sommano tra loro numeri che hanno all'incirca lo stesso module, ma segno opposto, il risultato della somma può essere assai impreciso e ci si riferisce a questa situazione con l'espressione **cancellazione di cifre significative**.

Risulta quindi necessario fare una distinzione. L'aritmetica (o la logica) utilizzata dal calcolatore viene chiamata **aritmetica floating-point** (quella spiegata fin ora); al contrario, l'**aritmetica esatta** si basa sulla effettuazione esatta delle operazioni elementari (quindi senza tener conto degli errori di arrotondamento) su operandi noti esattamente (e non attraverso la loro rappresentazione *floating-point*).

❓ **Va bene, ma perché è importante considerare questo aspetto?**

Nonostante gli errori di arrotondamento sono generalmente piccoli, se ripetuti all'interno di algoritmi lunghi e complessi, possono portare ad effetti catastrofici (vedi per esempio [l'incidente del razzo Ariane 5](#)).

Adesso si prova a definire in modo formale questo comportamento.

- Con e_a si identificano i **tipi di errore che si manifestano a seguito di una serie di errori di arrotondamento**.
- Con e_t si identifica l'**errore di troncamento**. Tali errori sono assenti soltanto in quei modelli matematici che sono già di dimensione finita (per esempio nella risoluzione di un sistema lineare).
- Con e_c si identifica l'**errore computazionale**, ovvero l'insieme degli errori e_a e e_t .

Indicando con x la soluzione esatta del modello matematico e con \hat{x} la soluzione ottenuta al termine del processo numerico, allora l'**errore computazionale assoluto** sarà dunque:

$$e_c^{ass} = |x - \hat{x}| \quad (30)$$

Nel caso in cui l'errore relativo fosse diverso da zero ($x \neq 0$):

$$e_c^{rel} = \frac{|x - \hat{x}|}{|x|} \quad (31)$$

Relazione tra gli errori di arrotondamento e MEG

L'introduzione fatta nelle pagine precedenti è servita perché è possibile trovare errori di arrotondamento con il prodotto LU, la quale non ritorna esattamente la matrice A .

Come appena accennato, il **prodotto LU produce un errore di arrotondamento**. Esso può essere **ridotto usando la tecnica del pivoting** che consente di contenerlo.

Inoltre, quando si risolve numericamente il sistema lineare $A\mathbf{x} = \mathbf{b}$ si può trovare la **soluzione esatta** $\hat{\mathbf{x}}$ di un **sistema perturbato** della forma:

$$(A + \delta A)\hat{\mathbf{x}} = \mathbf{b} + \delta \mathbf{b} \quad (32)$$

Dove δA e $\delta \mathbf{b}$ sono rispettivamente una matrice ed un vettore di perturbazione che dipendono dallo specifico metodo numerico impiegato nella risoluzione del sistema.

Usando le norme si ha:

$$\frac{\|\mathbf{x} - \hat{\mathbf{x}}\|}{\|\mathbf{x}\|} \leq \frac{\lambda_{\max}}{\lambda_{\min}} \cdot \frac{\|\delta \mathbf{b}\|}{\|\mathbf{b}\|} \quad (33)$$

Dove l'errore relativo sulla soluzione dipende dall'errore relativo sui dati attraverso la seguente costante (≥ 1):

$$K(A) = \frac{\lambda_{\max}}{\lambda_{\min}} \quad (34)$$

Essa viene chiamata **numero di condizionamento (spettrale) della matrice** A . Ovviamente, si ricorda che tale matrice A deve essere simmetrica e definita positiva.

Se la matrice A è una matrice simmetrica e definita positiva e δA una matrice non nulla simmetrica e definita positiva tale che $\lambda_{\max}(\delta A) < \lambda_{\min}(A)$, allora vale:

$$\frac{\|\mathbf{x} - \hat{\mathbf{x}}\|}{\|\mathbf{x}\|} \leq \frac{K(A)}{1 - \frac{\lambda_{\max}(\delta A)}{\lambda_{\min}(A)}} \cdot \left(\frac{\lambda_{\max}(\delta A)}{\lambda_{\min}(A)} + \frac{\|\delta \mathbf{b}\|}{\|\mathbf{b}\|} \right) \quad (35)$$

Infine, se le matrici A e δA non sono simmetriche e definite positive e δA è tale che $\|\delta A\|_2 \cdot \|A^{-1}\|_2 < 1$, vale la seguente stima:

$$\frac{\|\mathbf{x} - \hat{\mathbf{x}}\|}{\|\mathbf{x}\|} \leq \frac{K_2(A)}{1 - \frac{\|\delta A\|_2}{\|A\|_2}} \cdot \left(\frac{\|\delta A\|_2}{\|A\|_2} + \frac{\|\delta \mathbf{b}\|}{\|\mathbf{b}\|} \right) \quad (36)$$

Essendo $\|\delta A\|_2 = \sqrt{\lambda_{\max}(A^T \delta A)}$ e:

$$K_2(A) = \|\delta A\|_2 \cdot \|\delta A^{-1}\|_2 \quad (37)$$

Il **numero di condizionamento in norma 2**.

- Se $K_2(A)$ (o $K(A)$) è “piccolo” la matrice A viene detta **ben condizionata** ed a **piccoli errori sui dati corrisponderanno errori dello stesso ordine di grandezza sulla soluzione**.
- Se $K_2(A)$ (o $K(A)$) è “grande” la matrice A viene detta **mal condizionata** ed *potrebbe accadere* che a **piccole perturbazioni sui dati corrispondano grandi errori sulla soluzione**.

Infine, volendo l’equazione 33 può essere riscritta introducendo il **residuo** \mathbf{r} :

$$\mathbf{r} = \mathbf{b} - A\hat{\mathbf{x}} \quad (38)$$

Chiaramente se $\hat{\mathbf{x}}$ fosse la **soluzione esatta**, il **residuo sarebbe il vettore nullo**.

L’**efficacia** del residuo dipende dalla grandezza del numero di condizionamento di A . Infatti, sempre dall’equazione 33 si ricava:

$$\frac{\|\mathbf{x} - \hat{\mathbf{x}}\|}{\|\mathbf{x}\|} \leq K_2(A) \cdot \frac{\|\mathbf{r}\|}{\|\mathbf{b}\|} \quad (39)$$

- Se $K_2(A)$ è “piccolo” si avrebbe la **certezza che l’errore sarà piccolo quando lo è il residuo**.
- Se $K_2(A)$ è “grande” non si può avere la certezza che l’errore sarà piccolo quando lo è il residuo.

2.1.5 Il pivoting totale

Si opera un **pivoting totale** quando la ricerca del pivot è estesa alla sottomatrice $A^{(k)}$ costituita dagli elementi $a_{ij}^{(k)}$ con $i, j = k, \dots, n$. A differenza della tecnica del pivoting introdotta nel capitolo 2.1.3 a pagina 21, il parziale prevede il **coinvolgimento delle righe e delle colonne**, e conduce alla costruzione di due matrici di permutazione, chiamate P e Q , una sulle righe, l'altra sulle colonne:

$$PAQ = LU \quad (40)$$

Inoltre, la **soluzione** del sistema $Ax = b$ è ottenuta attraverso la **risoluzione di due sistemi triangolari e di una permutazione**:

$$Ly = Pb \quad Ux^* = y \quad x = Qx^* \quad (41)$$

Dal punto di vista **computazionale**, la tecnica del pivoting totale ha un **costo superiore rispetto a quello parziale** (capitolo 2.1.3 a pagina 21) in quanto ad ogni passo della fattorizzazione devono essere svolti molti più confronti. Tuttavia, può **apportare dei vantaggi in termini di risparmio di memoria e di stabilità**.

Quindi riassumendo:

❓ Come funziona?

Può essere visto come un pivoting parziale, ma in cui c'è un coinvolgimento anche delle colonne e non solo delle righe.

❓ Svantaggi

Più costoso rispetto al pivoting parziale poiché opera su più fronti e quindi devono essere eseguiti più confronti.

✓ Vantaggi

Non sempre, ma spesso si possono ottenere risparmi di memoria e un'alta stabilità.

2.1.6 Il *fill-in* di una matrice

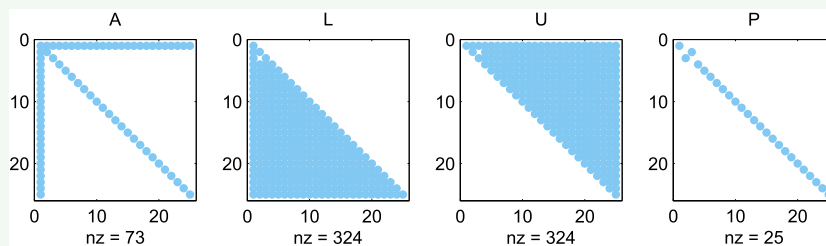
Un altro problema che è possibile riscontrare durante la fattorizzazione LU è il **fill-in**.

❓ Come si manifesta il fenomeno del *fill-in*?

Durante la fattorizzazione LU, non è certo che le matrici L ed U ottenute mantengano la struttura del corrispondente triangolo della matrice A iniziale. Al contrario, il **processo di fattorizzazione tende** generalmente a **riempire le matrici L ed U** . Tale fenomeno **dipende fortemente dalla struttura e dal valore dei singoli elementi non nulli** della matrice A .

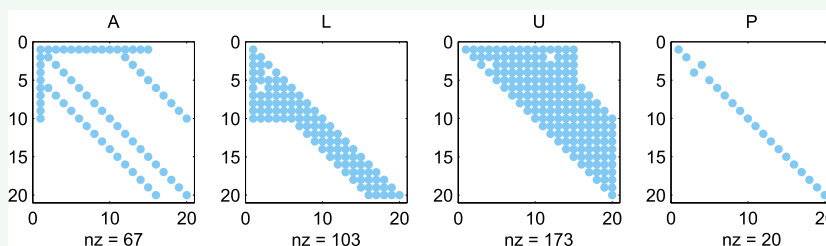
Esempio 1

Qua di seguito è possibile osservare un fenomeno di *fill-in* su una generica matrice.



Esempio 2

Un altro esempio di *fill-in* su una generica matrice. In questo caso, gli elementi non nulli della prima riga e della prima colonna di A inducono un riempimento totale delle corrispondenti colonne in U e righe in L , rispettivamente, mentre gli elementi non nulli sopra e sotto le diagonal di A comportano un riempimento delle diagonal superiori di U ed inferiori di L comprese tra quella principale e quelle non nulle di A .



❓ Come risolvere il *fill-in*?

Per ovviare al problema del *fill-in*, si possono adottare **tecniche di riordinamento che permutano righe e colonne della matrice prima di realizzare la fattorizzazione**. Tuttavia, in alcuni casi la **sola applicazione della tecnica di pivoting totale** (paragrafo 2.1.5 pagina 26) consente di raggiungere lo stesso obiettivo.

Prima di introdurre un nuovo esempio, si forniscono due definizioni.

Definizione 3

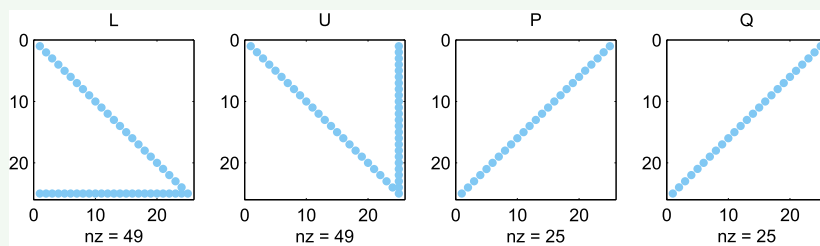
Una matrice quadrata di dimensione n è detta **matrice sparsa** se ha un numero di elementi non nulli dell'ordine di n (e non di n^2).

Definizione 4

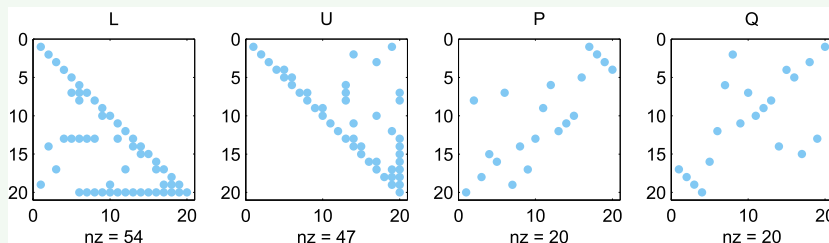
Si chiama **pattern** di una matrice sparsa l'insieme dei suoi elementi non nulli.

Esempio 3

Nella seguente figura sono mostrati i pattern delle matrici ottenute dalla fattorizzazione con pivotazione totale. Come si può vedere, il fenomeno di *fill-in* è contenuto.



Un altro esempio.



2.2 Metodi iterativi per sistemi lineari

Un **metodo iterativo** per la risoluzione del sistema lineare $Ax = b$ con:

- $A \in \mathbb{R}^{n \times n}$
- $b \in \mathbb{R}^n$
- $x \in \mathbb{R}^n$
- $\det(A) \neq 0$

Consiste nel costruire una successione di vettori del tipo:

$$\{\mathbf{x}^{(k)}, k \geq 0\}$$

Di \mathbb{R}^n che **converge** alla soluzione esatta \mathbf{x} , ossia tale che:

$$\lim_{k \rightarrow \infty} \mathbf{x}^{(k)} = \mathbf{x} \quad (42)$$

Per un qualunque vettore iniziale $\mathbf{x}^{(0)} \in \mathbb{R}^n$, ossia la **convergenza non deve dipendere dalla scelta di $\mathbf{x}^{(0)}$** .

Inoltre, si definisce ricorsivamente:

$$\mathbf{x}^{(k+1)} = B\mathbf{x}^{(k)} + \mathbf{g} \quad k \geq 0 \quad (43)$$

Essendo:

- B una matrice opportuna (dipendente da A)
- \mathbf{g} un vettore opportuno (dipendente da A e da \mathbf{b})

❓ E come si possono scegliere questi valori?

La scelta della matrice B e del vettore \mathbf{g} è piuttosto semplice. Devono essere rispettate due proprietà:

- **Condizione di consistenza.** I valori devono garantire che:

$$\mathbf{x} = B\mathbf{x} + \mathbf{g} \quad (44)$$

Essendo $\mathbf{x} = A^{-1}\mathbf{b}$, necessariamente dovrà aversi $\mathbf{g} = (I - B)A^{-1}\mathbf{b}$.

- **Condizione di convergenza.** Prima di dare la condizione, è necessario comprendere alcune cose. Prima di tutto, la seguente espressione:

$$\mathbf{e}^{(k)} = \mathbf{x} - \mathbf{x}^{(k)}$$

Viene identificata come l'**errore al passo k** . Adesso sottraendo l'equazione 43 dalla condizione di consistenza 44 si ottiene:

$$\mathbf{e}^{(k+1)} = B\mathbf{e}^{(k)}$$

Per tale ragione B viene detta **matrice di iterazione** del metodo 43.

E adesso, si presenta la condizione di convergenza:

- La matrice B è simmetrica⁴ e definita positiva⁵, allora:

$$\|e^{(k+1)}\| = \|Be^{(k)}\| \leq \rho(B) \|e^{(k)}\| \quad \forall k \geq 0 \quad (45)$$

Dove $\rho(B)$ è il **raggio spettrale** di B ed è il massimo degli autovalori di B . Si tenga conto che nel caso di matrici simmetriche e definite positive esso coincide con il massimo autovalore.

- Iterando a ritroso il punto precedente, si ottiene:

$$\|e^{(k)}\| \leq [\rho(B)]^k \|e^{(0)}\| \quad k \geq 0 \quad (46)$$

Si noti che se $\rho(B) < 1$, allora $e^{(k)} \rightarrow \mathbf{0}$ per $k \rightarrow \infty$ per ogni possibile e^0 (e, conseguentemente, per ogni $x^{(0)}$).

In generale, la condizione sufficiente e necessaria di convergenza è la seguente.

Definizione 5: condizione sufficiente e necessaria di convergenza

Un metodo iterativo della forma dell'equazione 43, la cui matrice di iterazione B soddisfi la condizione di consistenza (eq. 44), è **convergente** per ogni $x^{(0)}$ *se e soltanto se* $\rho(B) < 1$ (raggio spettrale minore di 1).

Inoltre, minore è $\rho(B)$, minore è il numero di iterazioni necessarie per ridurre l'errore iniziale di un dato fattore.

❓ Perché introdurre i metodi iterativi?

L'esigenza di introdurre i metodi iterativi sorge nel momento in cui si ragiona sulla quantità di tempo spesa da un calcolatore per eseguire la fattorizzazione LU su matrici di grandi dimensioni. Difatti, con matrici con ordini di 10^7 , sono necessari circa 11 giorni.

⁴Quindi corrisponde con la sua trasposta

⁵Definizione a pagina: 20

2.2.1 Il metodo di Jacobi

Se i coefficienti diagonali della matrice A non sono nulli, allora:

$$D = \text{diag}(a_{11}, a_{22}, \dots, a_{nn})$$

Ovvero D è la matrice diagonale costruita a partire dagli elementi diagonali di A .

Il **metodo di Jacobi** corrisponde alla forma:

$$D\mathbf{x}^{(k+1)} = \mathbf{b} - (A - D)\mathbf{x}^{(k)} \quad k \geq 0$$

Che per componenti assume la forma:

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1, j \neq i}^n a_{ij} x_j^{(k)} \right) \quad i = 1, \dots, n \quad (47)$$

Dove $k \geq 0$ e $\mathbf{x}^{(0)} = (x_1^{(0)}, x_2^{(0)}, \dots, x_n^{(0)})^T$ è il vettore iniziale.

🔍 Quando converge?

In due casi:

1. Se la matrice $A \in \mathbb{R}^{n \times n}$ è a dominanza diagonale stretta per righe⁶, allora il metodo di Jacobi converge.
2. Con la seguente definizione (si veda il paragrafo 2.2.2 per comprendere meglio le definizioni):

Definizione 6: convergenza di Jacobi e Gauss-Seidel

Se $A \in \mathbb{R}^{n \times n}$ è una **matrice tridiagonale**, quindi una matrice quadrata che al di fuori della diagonale principale e delle linee immediatamente al di sopra e al di sotto di essa (la prima sovradiagonale e la prima sottodiagonale), ha solo valori nulli. Un esempio

⁶ In algebra lineare una **matrice a dominanza diagonale stretta per righe** o **matrice diagonale dominante stretta per righe** è una matrice quadrata $A \in \mathbb{C}^{n \times n}$ di ordine n i cui elementi diagonali sono maggiori (non stretta sarebbe maggiori o uguali) in valore assoluto della somma di tutti i restanti elementi della stessa riga in valore assoluto:

$$|a_{ii}| > \sum_{j=1, j \neq i}^n |a_{ij}| \quad (48)$$

In senso non stretto:

$$|a_{ii}| \geq \sum_{j=1, j \neq i}^n |a_{ij}| \quad (49)$$

di amtrice tridiagonale:

$$\begin{bmatrix} 1 & 4 & 0 & 0 \\ 3 & 4 & 1 & 0 \\ 0 & 2 & 3 & 4 \\ 0 & 0 & 1 & 3 \end{bmatrix}$$

Se tale matrice è non singolare, quindi $\det(A) \neq 0$, con valori della diagonale diversi da zero, allora i metodi di Jacobi e di Gauss-Seidel sono **entrambi convergenti o entrambi divergenti**.

- Se convergono, la velocità dei metodi:

$$\text{Gauss-Seidel} \gg \text{Jacobi}$$

Precisamente il raggio spettrale della matrice di iterazione del metodo di Gauss-Seidel è il quadrato del raggio spettrale di quella del metodo di Jacobi.

HPC curiosity: è interessante notare che il metodo di Jacobi viene facilmente parallelizzato per aumentare le prestazioni di calcolo.

2.2.2 Il metodo di Gauss-Seidel

Con il metodo di Jacobi, ogni componente $x_i^{(k+1)}$ del nuovo vettore $\mathbf{x}^{(k+1)}$ viene calcolata indipendentemente dalle altre. Questa può essere la base di partenza per costruire un nuovo metodo più ottimizzato, poiché se per il calcolo di $x_i^{(k+1)}$ venissero usate le nuove componenti già disponibili $x_j^{(k+1)}$, $j = 1, \dots, i-1$, assieme con le vecchie $x_j^{(k)}$, $j \geq i$.

Supponendo che gli elementi della diagonale non siano nulli, per $k \geq 0$, il **metodo di Gauss-Seidel**:

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^n a_{ij} x_j^{(k)} \right) \quad i = 1, \dots, n \quad (50)$$

HPC curiosity: a differenza di Jacobi, questo metodo non può essere parallelizzato, ma è necessario operare in modo sequenziale.

🔍 Quando converge?

In tre casi:

1. Se la matrice $A \in \mathbb{R}^{n \times n}$ è a dominanza diagonale stressa per righe (vedi pag. 31), allora il metodo di Gauss-Seidel converge.
2. Se la matrice A è simmetrica (uguale alla sua trasposta) e definita positiva (vedi pag. 20), allora Gauss-Seidel converge.
3. Con la definizione di convergenza data per il metodo di Jacobi a pagina 31

Da notare: **non esistono risultati generali che consentono di affermare che il metodo di Gauss-Seidel converga sempre più rapidamente di quello di Jacobi**, a parte il caso della definizione a pagina 20.

2.2.3 Il metodo di Richardson

Una tecnica generale per costruire un metodo iterativo è basata sulla seguente **decomposizione additiva** (o **splitting**) della matrice A :

$$A = P - (P - A)$$

In cui P è un'opportuna matrice non singolare ($\det(P) \neq 0$) chiamata **precondizionatore** di A . Di conseguenza:

$$P\mathbf{x} = (P - A)\mathbf{x} + \mathbf{b}$$

È un sistema purché si ponga:

$$\begin{aligned} B &= P^{-1}(P - A) = I - P^{-1}A \\ \mathbf{g} &= P^{-1}\mathbf{b} \end{aligned}$$

Questa identità suggerisce la definizione del seguente metodo iterativo:

$$P(\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}) = \mathbf{r}^{(k)} \quad k \geq 0$$

In cui:

$$\mathbf{r}^{(k)} = \mathbf{b} - A\mathbf{x}^{(k)} \quad (51)$$

Denota il vettore residuo alla k -esima iterazione. Una generalizzazione di questo metodo iterativo è

$$P(\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}) = \alpha_k \mathbf{r}^{(k)} \quad k \geq 0 \quad (52)$$

Dove $\alpha_k \neq 0$ è un parametro che può cambiare ad ogni iterazione e che, a priori, servirà a migliorare le proprietà di convergenza della successione $\{\mathbf{x}^{(k)}\}$.

L'equazione 52 è nota come **metodo di Richardson** o **metodo di Richardson dinamico**; se $\alpha_k = \alpha$ per ogni $k \geq 0$ esso si dice **metodo di Richardson stazionario**.

Questo metodo richiede ad ogni passo di trovare il cosiddetto **residuo preconditionato** $\mathbf{z}^{(k)}$ dato dalla soluzione del sistema lineare:

$$P\mathbf{z}^{(k)} = \mathbf{r}^{(k)} \quad (53)$$

Quindi, la nuova iterata è definita da $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha_k \mathbf{z}^{(k)}$.

Per questa ragione la matrice P deve essere scelta in modo tale che il costo computazionale richiesto dalla risoluzione di 53 sia modesto (ogni matrice P diagonale, tridiagonale o triangolare andrebbe bene a questo scopo).

❓ Quando converge?

Per studiare la convergenza, si definisce la **norma dell'energia** associata alla matrice A :

$$\|\mathbf{v}\|_A = \sqrt{\mathbf{v}^T A \mathbf{v}} \quad \forall \mathbf{v} \in \mathbb{R}^n \quad (54)$$

E si definisce la seguente definizione.

Definizione 7

Sia $A \in \mathbb{R}^{n \times n}$.

Per ogni matrice non singolare ($\det \neq 0$) $P \in \mathbb{R}^{n \times n}$ il **metodo di Richardson stazionario converge** se e solo se:

$$|\lambda_i|^2 < \frac{2}{\alpha} \operatorname{Re} \lambda_i \quad \forall i = 1, \dots, n$$

In cui λ_i sono gli autovalori della matrice risultato di $P^{-1}A$.

In particolare, se gli autovalori della matrice risultato di $P^{-1}A$ sono reali, allora esso converge se e solo se:

$$0 < \alpha \lambda_i < 2 \quad \forall i = 1, \dots, n$$

Se A e P sono entrambe simmetriche (quindi uguali alle loro rispettive trasposte) e definite positive (definizione pagina 20) il metodo di Richardson stazionario **converge per ogni possibile scelta di $\mathbf{x}^{(0)}$** se e solo se:

$$0 < \alpha < \frac{2}{\lambda_{\max}}$$

Dove λ_{\max} (che è maggiore di zero) è l'autovalore massimo della matrice risultato di $P^{-1}A$.

E ancora, il raggio spettrale $\rho(B_\alpha)$ della matrice di iterazione $B_\alpha = I - \alpha P^{-1}A$ è il minimo quando $\alpha = \alpha_{\text{opt}}$, dove:

$$\alpha_{\text{opt}} = \frac{2}{\lambda_{\min} + \lambda_{\max}} \quad (55)$$

Dove ovviamente λ_{\min} è l'autovalore minimo della matrice risultato di $P^{-1}A$.

Infine, se $\alpha = \alpha_{\text{opt}}$, vale la seguente **stima di convergenza**:

$$\|\mathbf{e}^{(k)}\|_A \leq \left(\frac{K(P^{-1}A) - 1}{K(P^{-1}A) + 1} \right)^k \|\mathbf{e}^{(0)}\|_A \quad k \geq 0 \quad (56)$$

Si noti che la massima velocità di convergenza è data anche dal raggio spettrale:

$$\rho_{\text{opt}} = \frac{K(P^{-1}A) - 1}{K(P^{-1}A) + 1} \quad (57)$$

2.2.4 Il metodo del Gradiente e del Gradiente Coniugato

Si definisce la funzione $\Phi : \mathbb{R}^n \rightarrow \mathbb{R}$:

$$\Phi(\mathbf{x}) = \frac{1}{2} \mathbf{x}^T A \mathbf{x} - \mathbf{x}^T \mathbf{b} \quad (58)$$

Nel caso in cui la matrice A è simmetrica⁷ e definita positiva⁸, Φ è una funzione convessa, cioè tale che ogni per ogni $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ e per ogni $\alpha \in [0, 1]$ vale:

$$\Phi(\alpha \mathbf{x} + (1 - \alpha) \mathbf{y}) \leq \alpha \Phi(\mathbf{x}) + (1 - \alpha) \Phi(\mathbf{y}) \quad (59)$$

Ed ammette un unico punto stazionario \mathbf{x}^* che è anche un **punto di minimo locale ed assoluto**. Quindi:

$$\mathbf{x}^* = \underset{\mathbf{x} \in \mathbb{R}^n}{\operatorname{argmin}} \Phi(\mathbf{x}) \quad (60)$$

È l'unica soluzione dell'equazione:

$$\nabla \Phi(\mathbf{x}) = A\mathbf{x} - \mathbf{b} = \mathbf{0} \quad (61)$$

Risolvere il problema di minimo dell'equazione 60 **equivale** pertanto a **risolvere il sistema lineare**⁹.

Per un generico $\bar{\mathbf{x}} \in \mathbb{R}^n$ diverso da \mathbf{x}^* , $\nabla \Phi(\bar{\mathbf{x}})$ è un vettore non nullo di \mathbb{R}^n che individua la direzione lungo cui avviene:

- La **massima crescita** di Φ ;
- Di conseguenza $-\nabla \Phi(\bar{\mathbf{x}})$ individua la direzione di **massima decrescita** di Φ a partire da $\bar{\mathbf{x}}$.

Inoltre, ricordando che il vettore residuo nel punto $\bar{\mathbf{x}}$ è $\bar{\mathbf{r}} = \mathbf{b} - A\bar{\mathbf{x}}$, grazie alla direzione lungo cui avviene la massima crescita⁶¹, si ha che il **vettore residuo** è uguale a:

$$\bar{\mathbf{r}} = -\nabla \Phi(\bar{\mathbf{x}}) \quad (62)$$

Cioè il residuo **individua una possibile direzione lungo cui muoversi per avvicinarsi al punto di minimo \mathbf{x}^*** .

In generale, un vettore \mathbf{d} rappresenta una **direzione di discesa per Φ** nel punto $\bar{\mathbf{x}}$ se si verificano le seguenti condizioni:

$$\begin{aligned} \nabla \Phi(\bar{\mathbf{x}}) \neq \mathbf{0} &\Rightarrow \mathbf{d}^T \nabla \Phi(\bar{\mathbf{x}}) < 0 \\ \nabla \Phi(\bar{\mathbf{x}}) = \mathbf{0} &\Rightarrow \mathbf{d} = \mathbf{0} \end{aligned} \quad (63)$$

Il **residuo** è pertanto una **direzione di discesa**. È banale dire che quindi per **avvicinarsi al punto di minimo** ci si **muoverà lungo opportune direzioni di discesa**. I **metodi di discesa** sono definiti nel seguente modo.

⁷Quindi corrisponde con la sua trasposta

⁸Definizione a pagina: 20

⁹Il gradiente è un argomento del calcolo differenziale vettoriale e il vettore gradiente di una funzione scalare punta secondo la direzione di massima crescita della funzione stessa. Se si è un po' arrugginiti a riguardo, si può dare un'occhiata su [Wikipedia](#) o su [YouMath](#)

Definizione 8: metodi di discesa

Assegnato un vettore $\mathbf{x}^{(0)} \in \mathbb{R}^n$, per $k = 0, 1, \dots$ fino a convergenza:

1. Si determina una direzione di discesa $\mathbf{d}^{(k)} \in \mathbb{R}^n$
2. Si determina un passo $\alpha_k \in \mathbb{R}$
3. Si pone $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha_k \mathbf{d}^{(k)}$

❓ Ma quindi quale è la differenza tra Gradiente e Gradiente Coniugato?

I metodi del gradiente e del gradiente coniugato sono metodi di discesa che si **differenziano nella scelta delle direzioni di discesa**, mentre la scelta dei passi α_k è *comune* ad entrambi. Per cui, qua di seguito si esamina la scelta dei passi, supponendo di aver già calcolato la nuova direzione $\mathbf{d}^{(k)}$.

La **scelta del passo** α_k è comune ad entrambi, quindi:

$$\alpha_k = \underset{\alpha \in \mathbb{R}^n}{\operatorname{argmin}} \Phi(\mathbf{x}^{(k)} + \alpha \mathbf{d}^{(k)}) \quad (64)$$

Ovvero un passo α_k tale che $\Phi(\mathbf{x}^{(k+1)})$ sia il **minimo** di Φ lungo la direzione $\mathbf{d}^{(k)}$ passante per $\mathbf{x}^{(k)}$.

📖 Definizione metodo del gradiente

Adesso si può dare la definizione del Gradiente. Il **metodo del gradiente** è caratterizzato dalla scelta:

$$\mathbf{d}^{(k)} = \mathbf{r}^{(k)} = -\nabla \Phi(\mathbf{x}^{(k)}) \quad k = 0, 1, \dots \quad (65)$$

Quindi la **direzione di discesa ad ogni passo k è la direzione opposta a quella del gradiente** (come si vede dal meno davanti a ∇) della funzione Φ .

L'**algoritmo del metodo del gradiente** è: dato $\mathbf{x}^{(0)} \in \mathbb{R}^n$, si definisce $\mathbf{r}^{(0)} = \mathbf{b} - A\mathbf{x}^{(0)}$ e si calcola:

$$\begin{aligned} &\text{per } k = 0, 1, \dots \\ &\alpha_k = \frac{(\mathbf{r}^{(k)})^T \mathbf{r}^{(k)}}{(\mathbf{r}^{(k)})^T A \mathbf{r}^{(k)}} \\ &\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha_k \mathbf{r}^{(k)} \\ &\mathbf{r}^{(k+1)} = \mathbf{r}^{(k)} - \alpha_k A \mathbf{r}^{(k)} \end{aligned} \quad (66)$$

❓ Quando converge il metodo del gradiente?

Se $A \in \mathbb{R}^{n \times n}$ è simmetrica¹⁰ e definita positiva¹¹ il metodo del gradiente converge alla soluzione del sistema lineare $A\mathbf{x} = \mathbf{b}$ per ogni dato iniziale $\mathbf{x}^{(0)} \in \mathbb{R}^n$ e inoltre vale la seguente **stima dell'errore**:

$$\|\mathbf{e}^{(k)}\|_A \leq \left(\frac{K(A) - 1}{K(A) + 1} \right)^k \|\mathbf{e}^{(0)}\|_A \quad k \geq 0 \quad (67)$$

Dove $\|\cdot\|$ è la **norma dell'energia** definita a pagina 35 e $K(A)$ è il numero di condizionamento (spettrale) della matrice A definito a pagina 24.

⚠ Ma si può fare di meglio! Ed ecco che arriva il metodo del gradiente coniugato...

È possibile costruire delle direzioni di discesa “migliori” al fine di **giungere a convergenza in un numero di iterazioni inferiore rispetto a quelle del metodo del gradiente**. Partendo dunque dal metodo del gradiente e sfruttando la definizione ricorsiva dei vettori $\mathbf{x}^{(k)}$, si può ottenere:

$$\begin{aligned} \mathbf{x}^{(k+1)} &= \mathbf{x}^{(k)} + \alpha_k \mathbf{d}^{(k)} \\ &= \mathbf{x}^{(k-1)} + \alpha_{k-1} \mathbf{d}^{(k-1)} + \alpha_k \mathbf{d}^{(k)} \\ &= \dots \\ &= \mathbf{x}^{(0)} + \sum_{j=0}^k \alpha_j \mathbf{d}^{(j)} \end{aligned} \quad (68)$$

Si deduce che il vettore $(\mathbf{x}^{(k+1)} - \mathbf{x}^{(0)}) \in \mathbb{R}^n$ è una combinazione lineare delle direzioni di discesa $\{\mathbf{d}^{(j)}\}_{j=0}^k$.

Il **metodo del gradiente coniugato** costruisce un sistema di direzioni di discesa in \mathbb{R}^n che siano tutte linearmente indipendenti fra di loro (quindi $\{\mathbf{d}^{(k)}\}_{k=0}^{n-1}$ sarà una base di \mathbb{R}^n) e tali che i valori $\{\alpha_k\}_{k=0}^{n-1}$ siano proprio i coefficienti dello sviluppo di $(\mathbf{x}^* - \mathbf{x}^{(0)})$ rispetto alla base $\{\mathbf{d}^{(k)}\}_{k=0}^{n-1}$. Questo comporta che l' n -simo termine $\mathbf{x}^{(n)}$ della successione (eq. 68) coincide con la soluzione esatta \mathbf{x}^* .

❓ Quando converge il metodo del gradiente coniugato?

Sia $A \in \mathbb{R}^{n \times n}$ una matrice simmetrica e definita positiva. Il metodo del gradiente coniugato per risolvere un sistema lineare **converge al più in n iterazioni** (in aritmetica esatta). Inoltre, il residuo $\mathbf{r}^{(k)}$ alla k -esima iterazione (con $k < n$) è ortogonale a $\mathbf{d}^{(j)}$ per $j = 0, \dots, k-1$ e:

$$\|\mathbf{e}^{(k)}\|_A \leq \frac{2c^k}{1+c^{2k}} \|\mathbf{e}^{(0)}\|_A \quad (69)$$

¹⁰Quindi corrisponde con la sua trasposta

¹¹Definizione a pagina: 20

Con:

$$c = \frac{\sqrt{K(A)} - 1}{\sqrt{K(A)} + 1} \quad (70)$$

In aritmetica esatta il metodo del gradiente coniugato è pertanto un **metodo diretto in quanto termina dopo un numero finito di iterazioni!**

Il preconditionamento migliora il metodo del gradiente

Le iterazioni del metodo del gradiente, quando convergono alla soluzione esatta \mathbf{x}^* , sono caratterizzate da una **velocità di convergenza dipendente dal numero di condizionamento** della matrice A : **più grande è il numero di condizionamento di A e tanto più lenta sarà la convergenza**. Quindi, invece di risolvere il sistema lineare, si può risolvere il cosiddetto **sistema preconditionato** (equivalente):

$$P_L^{-1} A P_R^{-1} \hat{\mathbf{x}} = P_L^{-1} \mathbf{b} \quad \text{con } \hat{\mathbf{x}} = P_R \mathbf{x} \quad (71)$$

Dove P_L e P_R sono opportune matrici non singolari tali che:

1. La matrice:

$$P = P_L P_R \quad (72)$$

è detta **precondizionatore**, è simmetrica e definita positiva;

2. $K(P^{-1}A) \ll K(A)$

3. La risoluzione di sistemi lineari della forma:

$$P_L \mathbf{s} = \mathbf{t} \quad P_R \mathbf{v} = \mathbf{w} \quad (73)$$

è computazionalmente molto meno costosa della risoluzione del sistema originario $A\mathbf{x} = \mathbf{b}$.

Se si sceglie P_R uguale alla matrice identità, la matrice $P = P_L$ è detta **precondizionatore sinistro**, ed il sistema preconditionato assume la forma:

$$P^{-1} A \mathbf{x} = P^{-1} \mathbf{b} \quad (74)$$

Analogamente, se P_L è la matrice identità, la matrice $P = P_R$ è detta **precondizionatore destro**, ed il sistema preconditionato assume la forma:

$$A P^{-1} \hat{\mathbf{x}} = \mathbf{b} \quad \text{con } \hat{\mathbf{x}} = P \mathbf{x} \quad (75)$$

Il **metodo del gradiente preconditionato** si ottiene applicando il metodo del gradiente al sistema preconditionato in cui si prenda $P_R = P_L^T$ e quindi $P_R^{-1} = P_L^{-T} \equiv (P_L^{-1})^T$. Più precisamente, riscrivendo:

$$\underbrace{P_L^{-1} A P_L^{-T}}_{\hat{A}} \underbrace{P_L^T \mathbf{x}}_{\hat{\mathbf{x}}} = \underbrace{P_L^{-1} \mathbf{b}}_{\hat{\mathbf{b}}} \quad (76)$$

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. 41
Cambiare il formato delle variabili: <code>format</code>	Pag. 42
Assegnamento di vettori e matrici	Pag. 43
Operazioni su vettori e matrici	Pag. 46
Funzioni intrinseche per vettori e matrici	Pag. 50
Funzioni matematiche elementari	Pag. 55
Funzioni per definire vettori o matrici particolari	Pag. 56

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 π 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	<code>sqrt</code>
Esponenziale	<code>exp</code>
Logaritmo Naturale	<code>log</code>
Logaritmo In Base 2	<code>log2</code>
Logaritmo In Base 10	<code>log10</code>
Seno	<code>sin</code>
Arcoseno	<code>asin</code>
Coseno	<code>cos</code>
Arcocoseno	<code>acos</code>
Tangente	<code>tan</code>

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:

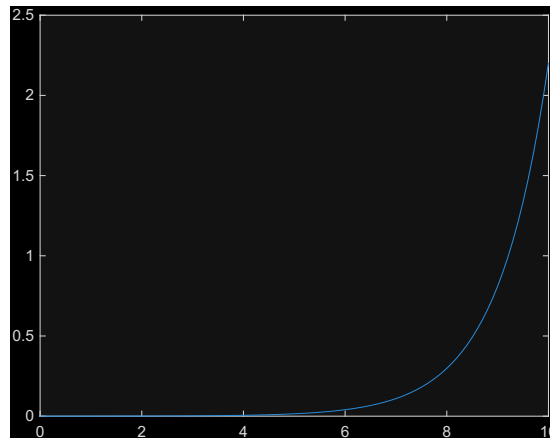
```
1 f(2)
2
3 ans =
4
5     8.3891
6
7 f(0:3)
8
9 ans =
10
11     2.0000     3.7183     8.3891    21.0855
```

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 $[0, 10]$.

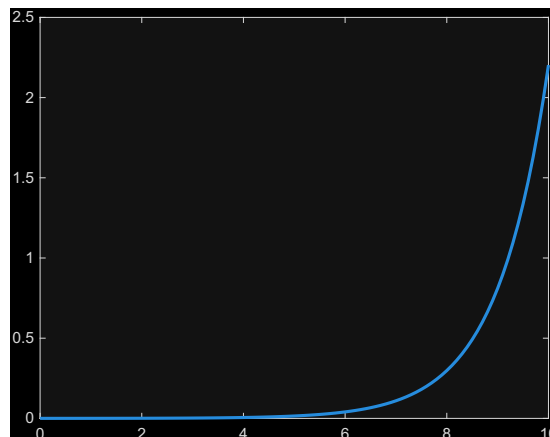
Per disegnare il grafico si utilizza il comando `plot`. Di default questa funzione disegna i valori in un piano cartesiano usando segmenti rettilinei (retta spezzata):

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



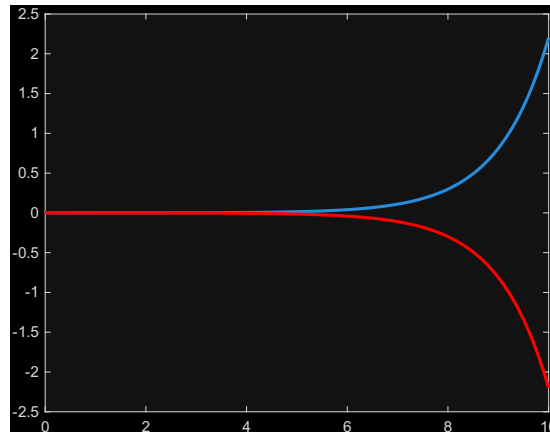
Per evitare che MATLAB sovrascriva la figura nella finestra aperta, è possibile numerarle usando la funzione `figure` (e.g. `figure(1); plot(x,y); figure(2); plot(0:3, 0:3)`).

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:

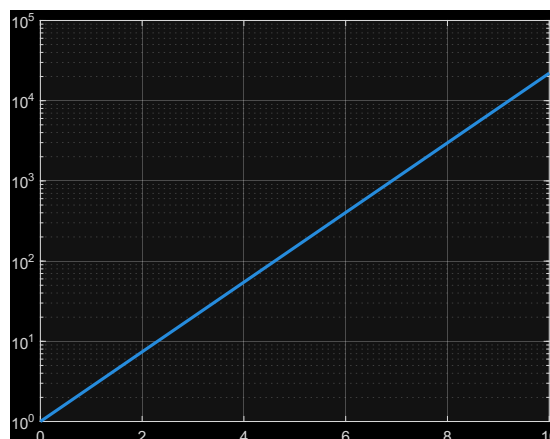
```
1 figure(1)
2 plot(x, f(x), 'linewidth', 2)
3 hold on
4 plot(x, -f(x), 'r', 'linewidth', 2)
```



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 52.

```

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.

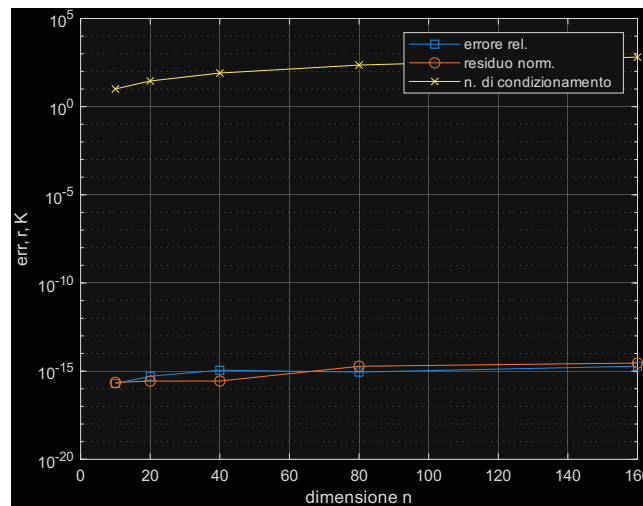


Figura 3: Andamento dell'errore relativo, del residuo normalizzato e del numero di condizionamento in funzione di n .

3.3.2 Metodi iterativi

I metodi iterativi stazionari sono considerati in genere nella seguente forma:

$$\mathbf{x}^{(k+1)} = B\mathbf{x}^{(k)} + \mathbf{f} \quad k \geq 0$$

Dove B è detta matrice di iterazione. B e \mathbf{f} identificano il metodo.

3.3.2.1 Metodo di Jacobi

Si consideri la matrice diagonale D degli elementi diagonali di A . Tale matrice è facilmente invertibile, se gli $a_{ii} \neq 0, i = 1, \dots, n$, in quanto:

$$D = \begin{pmatrix} a_{11} & 0 & 0 & \cdots & 0 \\ 0 & a_{22} & 0 & \cdots & 0 \\ \vdots & 0 & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & 0 \\ 0 & 0 & \cdots & 0 & a_{nn} \end{pmatrix} \Rightarrow D^{-1} = \begin{pmatrix} \frac{1}{a_{11}} & 0 & 0 & \cdots & 0 \\ 0 & \frac{1}{a_{22}} & 0 & \cdots & 0 \\ \vdots & 0 & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & 0 \\ 0 & 0 & \cdots & 0 & \frac{1}{a_{nn}} \end{pmatrix}$$

E il metodo può essere scritto direttamente in forma matriciale:

$$\begin{aligned} &\mathbf{x}^{(0)} \text{ assegnato} \\ &\mathbf{x}^{(k+1)} = B_J \mathbf{x}^{(k)} + \mathbf{f}_J \end{aligned}$$

Dove $B_J = I - D^{-1}A = D^{-1}(D - A)$ è la matrice di iterazione di Jacobi e $\mathbf{f}_J = D^{-1}\mathbf{b}$

3.3.2.2 Metodo di Gauss-Seidel

Questo metodo si differenzia dal metodo di Jacobi per il fatto che considera, oltre alla matrice D , anche le due matrici $-E$ e $-F$ triangolari superiore e inferiore della matrice A , ovvero:

$$-E = \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 \\ a_{21} & 0 & 0 & \cdots & 0 \\ \vdots & a_{32} & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & 0 \\ a_{n1} & a_{n2} & \cdots & a_{nn-1} & 0 \end{bmatrix} \quad -F = \begin{bmatrix} 0 & a_{12} & a_{13} & \cdots & a_{1n} \\ 0 & 0 & a_{23} & \cdots & a_{2n} \\ \vdots & 0 & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & a_{n-1n} \\ 0 & 0 & \cdots & 0 & 0 \end{bmatrix}$$

Dunque, il seguente algoritmo o le seguenti istruzioni:

$$\begin{aligned} &\mathbf{x}^{(0)} \text{ assegnato} \\ &\mathbf{x}^{(k+1)} = B_{GS} \mathbf{x}^{(k)} + \mathbf{f}_{GS} \end{aligned}$$

Dove $B_{GS} = (D - E)^{-1}F$ è la matrice d'iterazione di Gauss-Seidel e $\mathbf{f}_{GS} = (D - E)^{-1}\mathbf{b}$.

3.3.2.3 Esercizio

Si considerino la matrice:

$$A = \begin{bmatrix} 9 & -3 & 1 & & & & \\ -3 & 9 & -3 & 1 & & & \\ 1 & -3 & 9 & -3 & 1 & & \\ & 1 & -3 & 9 & -3 & 1 & \\ & & 1 & -3 & 9 & -3 & 1 \\ & & & 1 & -3 & 9 & -3 \\ & & & & 1 & -3 & 9 \end{bmatrix}$$

E il termine noto:

$$\mathbf{b} = [7 \ 4 \ 5 \ 5 \ 5 \ 4 \ 7]^T$$

1. Costruire la matrice A (utilizzando i comandi Matlab `diag` e `ones`) e determinare il numero di elementi non nulli tramite il comando `nnz`. La matrice A è a dominanza diagonale per righe? È simmetrica e definita positiva?

È stato richiesto di utilizzare i comandi `diag` e `ones` per costruire la matrice A . Quindi per farlo, si controlla rapidamente la documentazione dei due comandi:

```

1 >> help ones
2 ones - Create array of all ones
3 This MATLAB function returns the scalar 1.
4
5 Syntax
6 X = ones
7 X = ones(n)
8 X = ones(sz1,...,szN)
9 X = ones(sz)
10
11 X = ones(___,typename)
12 X = ones(___, 'like', p)
13
14 Input Arguments
15 n - Size of square matrix
16 integer value
17 sz1,...,szN - Size of each dimension
18 two or more integer values
19 sz - Output size
20 row vector of integer values
21 typename - Output class
22 'double' (default) | 'single' | 'logical' | 'int8' | '
uint8' | ...
23 p - Prototype
24 variable
25
26 >> help diag
27 diag - Create diagonal matrix or get diagonal elements of
matrix
28 This MATLAB function returns a square diagonal matrix with
the elements
29 of vector v on the main diagonal.
30
31 Syntax
32 D = diag(v)
33 D = diag(v,k)

```

```

34     x = diag(A)
35     x = diag(A,k)
36
37     Input Arguments
38     v - Diagonal elements
39     vector
40     A - Input matrix
41     matrix
42     k - Diagonal number
43     integer
44

```

Adesso si osservi la matrice A . È possibile notare che la diagonale principale ha tutti i valori uguale a 9, mentre sopra e sotto la diagonale principale, altre due diagonaloni con valore pari a -3 e allo stesso modo due diagonaloni con valore uguale a 1.

Usando entrambi i comandi, si può giungere al seguente risultato parziale:

```

1 >> diag(9*ones(1, n)) + diag(-3*ones(1,n-1), 1) + diag(1*ones
2     (1,n-2), 2)
3 ans =
4
5     9     -3     1     0     0     0     0
6     0     9     -3     1     0     0     0
7     0     0     9     -3     1     0     0
8     0     0     0     9     -3     1     0
9     0     0     0     0     9     -3     1
10    0     0     0     0     0     9     -3
11    0     0     0     0     0     0     9

```

Ed eseguendo con la stessa logica anche sotto la diagonale principale, si ottiene la matrice A richiesta:

```

1 n = 7;
2 A = diag(9*ones(1, n)) + ... % diagonale principale
3     diag(-3*ones(1,n-1), 1) + diag(-3*ones(1,n-1), -1) + ...
4     diag(1*ones(1,n-2), 2) + diag(1*ones(1,n-2), -2)
5
6 % ans =
7 %
8 %     9     -3     1     0     0     0     0
9 %     -3     9     -3     1     0     0     0
10 %     1     -3     9     -3     1     0     0
11 %     0     1     -3     9     -3     1     0
12 %     0     0     1     -3     9     -3     1
13 %     0     0     0     1     -3     9     -3
14 %     0     0     0     0     1     -3     9

```

Il numero di elementi non nulli si calcola con il comando `nnz`:

```

1 >> help nnz
2 nnz - Number of nonzero matrix elements
3 This MATLAB function returns the number of nonzero
4     elements in matrix X.
5
6 Syntax
7     N = nnz(X)
8
9 Input Arguments
10    X - Input matrix

```

```

10         matrix
11
12 >> nnz(A)
13
14 ans =
15
16     29

```

E infine, per confermare che la matrice sia a dominanza diagonale per righe, simmetrica e definita positiva:

```

1 % applicando la definizione
2 n = 7;
3 A = diag(9*ones(1, n)) + ... % diagonale principale
4     diag(-3*ones(1, n-1), 1) + diag(-3*ones(1, n-1), -1) + ...
5     diag(1*ones(1, n-2), 2) + diag(1*ones(1, n-2), -2);
6 A_diag = diag(abs(A));
7 A_no_diag = diag(-3*ones(1, n-1), 1) + ...
8             diag(-3*ones(1, n-1), -1) + ...
9             diag(1*ones(1, n-2), 2) + diag(1*ones(1, n-2), -2);
10 % definizione:
11 % https://it.wikipedia.org/wiki/Matrice_a_diagonale_dominante
12 for index = n
13     if not(A_diag(index) >= sum(abs(A_no_diag(index:index, 1:n)
14 )))
15         error("La matrice non e' a diagonale dominante")
16     end
17 end
18 % oppure in modo piu' rapido:
19 if (2 * abs(diag(A)) - sum(abs(A), 2) > 0)
20     disp("La matrice e' a diagonale dominante")
21 end
22
23 % una matrice e' simmetrica se e' uguale alla sua trasposta
24 if not(A == transpose(A))
25     disp("La matrice non e' simmetrica")
26 end
27
28 % il metodo piu' efficiente per controllare se una matrice
29 % e' simmetrica e definita positiva, e' con l'utilizzo
30 % della fattorizzazione di Cholesky
31 % (studiata nella parte di teoria)
32 % source: https://rb.gy/uko7gs
33 try chol(A);
34     disp("La matrice e' simmetrica e definita positiva")
35 catch ME
36     error("La matrice non e' simmetrica definita positiva")
37 end

```

2. Si calcolino le matrici di iterazione:

$$B_J = D^{-1}(D - A)$$

$$B_{GS} = (D - E)^{-1}F$$

Associate rispettivamente ai metodi di Jacobi e Gauss-Seidel e i relativi raggi spettrali. La condizione necessaria e sufficiente per la convergenza del metodo iterativo è soddisfatta in entrambi i casi?

Le matrici di iterazione dei due metodi si calcolano a partire dalla definizione:

```

1 D = diag(diag(A));
2 Bj = D \ (D - A); % matrice di iterazione di Jacobi
3
4 E = -tril(A, -1);
5 F = -triu(A, 1);
6 Bgs = (D - E) \ F; % matrice di iterazione di Gauss-Seidel
7
8 rho_j = max(abs(eig(Bj)))
9 rho_gs = max(abs(eig(Bgs)))

```

Si noti l'istruzione $D = \text{diag}(\text{diag}(A))$; il comando interno estrae la diagonale principale di A , restituendo un vettore, il quale viene elaborato dal comando più esterno che crea una seconda matrice quadrata identica alla dimensione di A ma con solo la diagonale principale.

Dal calcolo del raggio spettrale delle matrici si può concludere che in questo caso entrambi i metodi convergono, in quanto l'autovalore massimo risulta in modulo strettamente minore di 1. Si osservi che il raggio spettrale della matrice di iterazione del metodo di Gauss-Seidel è più basso di quello della matrice del metodo di Jacobi.

3. Scrivere la funzione Matlab che implementi il metodo di Jacobi inversione *matriciale* per il sistema lineare $A\mathbf{x} = \mathbf{b}$. L'intestazione della funzione sarà la seguente:

$$[\mathbf{x}, k] = \text{jacobi}(A, \mathbf{b}, \mathbf{x}_0, \text{toll}, \text{nmax}).$$

Il processo iterativo si arresta quando:

$$\frac{\|\mathbf{r}^{(k)}\|}{\|\mathbf{b}\|} \leq \text{toll}$$

(criterio d'arresto del residuo normalizzato).

```

1 function [x,k]=jacobi(A,b,x0,toll,nmax)
2
3 % Metodo di Jacobi
4 %
5 % A: matrice del sistema
6 % b: termine noto
7 % x0: vettore iniziale
8 % toll: tolleranza sul residuo normalizzato
9 % nmax: massimo numero di iterazioni
10 %
11 % x: soluzione ottenuta
12 % k: numero di iterazioni effettuate
13
14 n = size(b,1);
15
16 % Controlliamo che la matrice A sia quadrata e che,
17 % insieme al guess iniziale x0,
18 % abbia dimensioni compatibili con b.
19 if ((size(A,1) ~= n) || (size(A,2) ~= n) || (size(x0,1) ~= n))
20     error('Dimensioni incompatibili')
21 end
22
23 % Controlliamo che la matrice A non abbia elementi
24 % diagonali nulli.
25 if (prod(diag(A)) == 0)

```



```

26     error('res_normore: elementi diagonali nulli')
27 end
28
29 % Estraiamo la matrice D da A e calcoliamo la matrice
30 % d'iterazione e il termine noto g
31 D = diag(diag(A));
32 Bj = eye(n) - D\A;
33 g = D\b;
34
35 % Inizializziamo x come x0, calcoliamo il residuo
36 % e l'res_normore normalizzato
37 x = x0;
38 r = b - A*x;
39 res_norm = norm(r) / norm(b);
40
41 % Inizializziamo l'indice d'iterazione
42 k = 0;
43
44 while (res_norm > toll && k < nmax)
45     k = k + 1;
46
47     % Calcoliamo il nuovo x
48     x=Bj*x+g;
49
50     % Calcoliamo residuo e res_normore
51     r = b - A*x;
52     res_norm = norm(r)/norm(b);
53 end

```

4. Scrivere una funzione Matlab che implementi il metodo di Gauss-Seidel inversione *matriciale* per il sistema lineare $Ax = b$. L'intestazione della funzione sarà la seguente:

$$[x,k] = \text{gs}(A,b,x0,toll,nmax).$$

```

1 function [x,k]=gs(A,b,x0,toll,nmax)
2
3 % Metodo di Gauss-Seidel
4 %
5 % A: matrice del sistema
6 % b: termine noto
7 % x0: vettore iniziale
8 % toll: tolleranza sul residuo normalizzato
9 % nmax: massimo numero di iterazioni
10 %
11 % x: soluzione ottenuta
12 % k: numero di iterazioni effettuate
13
14 n = size(b,1);
15
16 % Controlliamo che la matrice A sia quadrata e che,
17 % insieme al guess iniziale x0,
18 % abbia dimensioni compatibili con b.
19 if ((size(A,1)~=n) || (size(A,2)~=n) || (size(x0,1) ~= n) )
20     error('dimensioni incompatibili')
21 end
22
23 % Controlliamo che la matrice A non abbia
24 % elementi diagonali nulli.
25 if (prod(diag(A)) == 0)
26     error('errore: elementi diagonali nulli')

```

```

27 end
28
29 % Decomponiamo la matrice in D,E e F,
30 % e calcoliamo la matrice d'iterazione e il termine g
31 D=diag(diag(A));
32 E=-tril(A,-1);
33 F=-triu(A,1);
34 Bgs=(D-E)\F;
35 g=(D-E)\b;
36
37 % Inizializziamo x come x0, calcoliamo il residuo
38 % e l'errore normalizzato
39 x = x0;
40 r = b - A * x;
41 err = norm(r) / norm(b);
42
43 % Inizializziamo l'indice d'iterazione
44 k = 0;
45
46 while ( err > toll && k < nmax )
47     k = k + 1;
48
49     % Calcoliamo il nuovo x
50     x=Bgs*x+g;
51
52     % Calcoliamo residuo e errore
53     r = b - A*x;
54     err = norm(r)/norm(b);
55 end

```

5. Costruire il termine noto \mathbf{b} . Utilizzando le funzioni costruite nei punti 3 e 4, risolvere il sistema $A\mathbf{x} = \mathbf{b}$ ponendo $x^{(0)} = [0, 0, \dots, 0]^T$, $\text{toll} = 10^{-6}$ e $\text{nmax} = 1000$. Confrontare il numero di iterazioni necessarie per arrivare a convergenza per i due metodi e commentare i risultati ottenuti.

Il metodo di Gauss-Seidel converge più velocemente alla soluzione esatta in accordo con il corrispondente raggio spettrale che è più basso di quello della matrice del metodo di Jacobi.

```

1 b = transpose([7 4 5 5 5 4 7]);
2 toll = 1e-6;
3 x0 = zeros(n, 1);
4 nmax = 1000;
5
6 [xJ, kJ] = jacobi(A, b, x0, toll, nmax);
7 [xGS, kGS] = gs(A, b, x0, toll, nmax);
8
9
10 xJ
11 kJ
12 xGS
13 kGS
14
15 % xJ =
16 %
17 %     1.0000
18 %     1.0000
19 %     1.0000
20 %     1.0000
21 %     1.0000
22 %     1.0000

```

```
23 %      1.0000
24 %
25 %
26 % kJ =
27 %
28 %      49
29 %
30 %
31 % xGS =
32 %
33 %      1.0000
34 %      1.0000
35 %      1.0000
36 %      1.0000
37 %      1.0000
38 %      1.0000
39 %      1.0000
40 %
41 %
42 % kGS =
43 %
44 %      12
```

3.3.2.4 Metodo di Richardson

Il metodo di Richardson stazionario è basato sulla seguente legge. Dati $\mathbf{x}^{(0)}$ e $\alpha \in \mathbb{R}$ si calcola:

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha \mathbf{r}^{(k)} \quad k \geq 0 \quad (77)$$

Dove $\alpha \neq 0$ è un parametro costante per ogni iterazione. Questo metodo richiede, ad ogni passo k , di calcolare il *residuo* $\mathbf{r}^{(k)}$ definito come:

$$\mathbf{r}^{(k)} = \mathbf{b} - A\mathbf{x}^{(k)}$$

Il metodo di Richardson *stazionario* converge solo per $0 < \alpha < \frac{2}{\lambda_{\max}}$, in cui λ_{\max} è il massimo degli autovalori della matrice A . Inoltre, è possibile calcolare un valore di α ottimale che massimizza la velocità di convergenza. Questo valore è dato da:

$$\alpha_{\text{opt}} = \frac{2}{\lambda_{\min} + \lambda_{\max}}$$

In cui λ_{\min} è il minimo degli autovalori della matrice A . Per questo valore, la velocità di convergenza è data da:

$$\rho_{\text{opt}} = \frac{K(A) - 1}{K(A) + 1}$$

Dove $K(A)$ è il numero di condizionamento, definito anche come:

$$K(A) = \|A^{-1}\| \cdot \|A\|$$

Dove $\|\cdot\|$ è una opportuna norma introdotta per la matrice. Vale sempre $K(A) \geq 1$. Se la matrice A è simmetrica e definita positiva, utilizzando la sua norma 2, vale:

$$K(A) = \|A^{-1}\|_2 \cdot \|A\|_2 = \frac{\lambda_{\max}}{\lambda_{\min}}$$

Si noti che, dalla sua definizione, il metodo di Richardson (eq. 77) può essere riscritto nella seguente forma (utilizzando la definizione di \mathbf{r}):

$$\mathbf{x}^{(k+1)} = (I - \alpha A) \mathbf{x}^{(k)} + \alpha \mathbf{b} \quad k \geq 0 \quad (78)$$

Segue che il metodo di Richardson è un metodo iterativo caratterizzato dalla matrice di iterazione $B_\alpha = I - \alpha A$ e da $\mathbf{f} = \alpha \mathbf{b}$.

3.3.2.5 Precondizionamento

Il numero di condizionamento di una matrice governa il rapporto tra l'errore relativo commesso dalla soluzione numerica nella risoluzione di un sistema lineare e il corrispondente residuo normalizzata, alla iterata k :

$$\frac{\|\mathbf{x}^{(k)} - \mathbf{x}\|}{\|\mathbf{x}^{(k)}\|} \leq K(A) \frac{\|\mathbf{r}^{(k)}\|}{\|\mathbf{b}\|}$$

Dove:

- $\mathbf{x}^{(k)}$ è la soluzione numerica
- $\mathbf{r}^{(k)}$ è il residuo, ovvero $\mathbf{b} - A\mathbf{x}^{(k)}$
- \mathbf{x} è la soluzione esatta del sistema lineare

Inoltre, nel caso del metodo di Richardson, si ottiene:

$$\|\mathbf{e}^{(k+1)}\| \leq \frac{K(A) - 1}{K(A) + 1} \cdot \|\mathbf{e}^{(k)}\|$$

Come migliore stima ottenibile usando α_{opt} .

Per problemi ben condizionati ($K(A)$ non molto più grande di 1), la soluzione del problema con piccoli residui non differisce molto dalla soluzione del problema originale; al contrario, in problemi con la matrice mal condizionata ($K(A) \gg 1$) a piccoli residui possono corrispondere grandi errori e la convergenza è molto lenta.

L'idea del precondizionamento consiste nel cercare di ridurre il numero di condizionamento della matrice del sistema, pre-moltiplicandola per una matrice P^{-1} (P è chiamata *precondizionatore*). Si ottiene così il sistema equivalente:

$$P^{-1}A\mathbf{x}^{(k)} = P^{-1}\mathbf{b}$$

Ovviamente il precondizionatore è efficace se $K(P^{-1}A) \ll K(A)$ e se la soluzione del sistema lineare in P che sorge ad ogni iterazione non è troppo onerosa. La prima proprietà è solitamente verificata quando $P^{-1} \approx A^{-1}$, cioè quando P e A hanno uno spettro simile, ma dovendo tener conto della seconda condizione è opportuno scegliere P con una struttura speciale che mantenga basso il costo computazionale (ad esempio diagonale o triangolare).

Ad esempio, nel caso di Richardson, per alleviare la dipendenza della convergenza ottimale dal numero di condizionamento, si introduce la tecnica di precondizionamento, che consiste nel sostituire A con $P^{-1}A$. Questo metodo richiede, ad ogni iterata, di trovare il cosiddetto *residuo precondizionato* $\mathbf{z}^{(k)}$ dato dalla soluzione del sistema lineare:

$$P\mathbf{z}^{(k)} = \mathbf{r}^{(k)} \quad (79)$$

Di conseguenza, la nuova iterata è definita da $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha\mathbf{z}^{(k)}$. Si noti che la matrice P , oltre ad essere non singolare (determinante diverso da zero),

simmetrica (coincide con la sua trasposta) e definita positiva, deve essere scelta in modo tale che il costo computazionale richiesto dalla risoluzione del sistema (eq. 79) sia basso. Per il caso preconditionato, valgono i precedenti risultati su α_{opt} e ρ_{opt} a patto che si considerino gli autovalori di $P^{-1}A$ invece di quelli di A .

3.3.2.6 Metodo del gradiente

È possibile generalizzare il metodo di Richardson preconditionato tramite l'introduzione di un parametro di accelerazione dinamico α_k :

$$P(\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}) = \alpha_k \mathbf{r}^{(k)} \quad k \geq 0, \alpha_k \neq 0$$

Con P il preconditionatore. Lo scopo dell'utilizzo di α_k è quello di poter calcolare facilmente il parametro di accelerazione evitando il calcolo (spesso oneroso) degli autovalori di A come per α_{opt} . Questo metodo è detto *metodo di Richardson dinamico*. Se α_k è scelto in modo ottimale, allora il metodo è detto del *gradiente* se $P = I$, oppure del *gradiente preconditionato* se $P \neq I$.

A partire dal metodo di Richardson, è possibile ottenere i metodi del *gradiente* ($P = I$) e del *gradiente preconditionato* ($P \neq I$) tramite l'aggiunta del parametro α_k : dato $\mathbf{x}^{(0)}$ assegnato, si ponga $\mathbf{r}^{(0)} = \mathbf{b} - A\mathbf{x}^{(0)}$ e per $k = 0, 1, \dots$

$$\begin{aligned} P\mathbf{z}^{(k)} &= \mathbf{r}^{(k)} \\ \alpha_k &= \frac{(\mathbf{z}^{(k)})^T \mathbf{r}^{(k)}}{(\mathbf{z}^{(k)})^T A\mathbf{z}^{(k)}} \\ \mathbf{x}^{(k+1)} &= \mathbf{x}^{(k)} + \alpha_k \mathbf{z}^{(k)} \\ \mathbf{r}^{(k+1)} &= \mathbf{r}^{(k)} + \alpha_k A\mathbf{z}^{(k)} \end{aligned} \tag{80}$$

3.3.2.7 Esercizi su Richardson e gradiente

Si considerino la matrice:

$$A = \begin{bmatrix} 9 & -3 & 1 & & & & \\ -3 & 9 & -3 & 1 & & & \\ 1 & -3 & 9 & -3 & 1 & & \\ & 1 & -3 & 9 & -3 & 1 & \\ & & 1 & -3 & 9 & -3 & 1 \\ & & & 1 & -3 & 9 & -3 \\ & & & & 1 & -3 & 9 \end{bmatrix}$$

E il termine noto:

$$\mathbf{b} = [7 \ 4 \ 5 \ 5 \ 5 \ 4 \ 7]^T$$

1. Costruire la matrice A (utilizzando i comandi Matlab `diag` e `ones`).

```

1 n = 7;
2 A = diag(9*ones(1, n)) + ... % diagonale principale
3     diag(-3*ones(1,n-1), 1) + diag(-3*ones(1,n-1), -1) + ...
4     diag(1*ones(1,n-2), 2) + diag(1*ones(1,n-2), -2)
5
6 % ans =
7 %
8 %      9      -3       1       0       0       0       0
9 %     -3       9      -3       1       0       0       0
10 %      1      -3       9      -3       1       0       0
11 %      0       1      -3       9      -3       1       0
12 %      0       0       1      -3       9      -3       1
13 %      0       0       0       1      -3       9      -3
14 %      0       0       0       0       1      -3       9

```

2. Calcolare l'intervallo di valori di α per cui il metodo di Richardson stazionario non preconditionato converge. Determinare il valore ottimale di α per avere massima velocità di convergenza.

```

1 % calcolo di eigenvalues e eigenvectors
2 % V: right eigenvectors (matrice quadrata)
3 % D: eigenvalues (matrice diagonale)
4 [V,D] = eig(A);
5
6 % si ottiene l'autovalore massimo
7 lambda_max = max(diag(D))
8 % lambda_max =
9 %
10 %      16.0403
11
12 % il limite (supponendo lambda_min = 0)
13 Lim = 2/lambda_max
14 % Lim =
15 %
16 %      0.1247
17
18 % si calcola infine alpha opt
19 lambda_min = min(diag(D))
20 alpha_opt = 2/(lambda_min+lambda_max)
21 % lambda_min =
22 %
23 %      4.9042
24 %

```

```

25 %
26 % alpha_opt =
27 %
28 %      0.0955

```

3. Scrivere una funzione Matlab che implementi il metodo di Richardson stazionario non preconditionato per il sistema lineare $A\mathbf{x} = \mathbf{b}$. I parametri in ingresso richiesti dalla funzione sono la matrice A , il termine noto \mathbf{b} , il guess iniziale $\mathbf{x}^{(0)}$, il coefficiente α , la tolleranza per il criterio d'arresto `toll` e il numero massimo di iterazioni ammesse `nmax`. La funzione restituisce la soluzione numerica \mathbf{x} e il numero di iterazioni effettuate k .

L'intestazione della funzione sarà la seguente:

$$[\mathbf{x}, k] = \text{richardson}(A, \mathbf{b}, \mathbf{x}_0, \alpha, \text{toll}, \text{nmax})$$

Il processo iterativo si arresta quando:

$$\frac{\|\mathbf{r}^{(k)}\|}{\|\mathbf{b}\|} \leq \text{toll}$$

(criterio d'arresto del residuo normalizzato).

```

1 function [x, k, err] = richardson(A, b, x0, alpha, toll, nmax)
2
3 % Metodo di Richardson stazionario
4 %
5 % A: matrice del sistema
6 % b: termine noto
7 % x0: vettore iniziale
8 % alpha: coefficiente di Richardson
9 % toll: tolleranza sul residuo normalizzato
10 % nmax: massimo numero di iterazioni
11 %
12 % x: soluzione ottenuta
13 % it: numero di iterazioni effettuate
14
15 n = size(b,1);
16 k = 0;
17
18 if ((size(A,1) ~= n) || (size(A,2) ~= n) || (size(x0,1) ~= n))
19     error('Dimensioni incompatibili')
20 end
21
22 x = x0;
23 r = b - A*x;
24 errk = norm(r) / norm(b);
25 err = errk;
26
27 while (errk > toll && k < nmax)
28     k = k + 1;
29     x = x + alpha*r;
30     r = b - A*x;
31     errk = norm(r)/norm(b);
32     err = [err; errk];
33 end

```

4. Costruire il termine noto \mathbf{b} . Utilizzando la funzione scritta al punto precedente, determinare la soluzione del sistema lineare $A\mathbf{x} = \mathbf{b}$, con A data dal punto 1.

Si ponga $\mathbf{x}^{(0)} = [0, 0, \dots, 0]^T$, $\text{toll} = 10^{-6}$ e $\text{nmax} = 1000$. Si verifichi sperimentalmente che il metodo di Richardson stazionario non preconditionato converge solo se α appartiene all'intervallo trovato nel punto 2. In particolare, si scelga un α al di fuori dell'intervallo e si verifichi che il metodo diverge. Si provi, inoltre, sperimentalmente, che per α_{opt} il metodo converge più velocemente. In particolare, si scelga un α nell'intervallo di convergenza e si osservi il numero di iterazioni per aggiungere a convergenza è maggiore di quello ottenuto utilizzando α_{opt} .

```

1 % calcolo della trasposta
2 b = transpose([7 4 5 5 5 4 7]);
3
4 % si pongono i dati richiesti
5 x0 = zeros(n,1);
6 toll = 1e-6;
7 nmax = 1000;
8
9 % si utilizza prima un alpha fuori dal limite del punto 2
10 alpha = 2;
11 [xR1,kR1] = richardson(A,b,x0,alpha,toll,nmax)
12 % xR1 =
13 %
14 % -Inf
15 % Inf
16 % -Inf
17 % Inf
18 % -Inf
19 % Inf
20 % -Inf
21 %
22 %
23 % kR1 =
24 %
25 % 208
26
27 % adesso ci si avvicina al valore alpha
28 alpha = 0.11;
29 [xR2,kR2] = richardson(A,b,x0,alpha,toll,nmax)
30 % xR2 =
31 %
32 % 1.0000
33 % 1.0000
34 % 1.0000
35 % 1.0000
36 % 1.0000
37 % 1.0000
38 % 1.0000
39 %
40 %
41 % kR2 =
42 %
43 % 45
44
45 % e infine si utilizza alpha opt per verificare
46 % che sia il migliore
47 [xR3,kR3] = richardson(A,b,x0,alpha_opt,toll,nmax)
48 % xR3 =
49 %
50 % 1.0000
51 % 1.0000
52 % 1.0000

```

```

53 %      1.0000
54 %      1.0000
55 %      1.0000
56 %      1.0000
57 %
58 %
59 % kR3 =
60 %
61 %      22

```

Adesso si passa al gradiente. Si consideri il problema lineare $A\mathbf{x} = \mathbf{b}$, dove la matrice $A \in \mathbb{R}^{n \times n}$ è pentadiagonale:

$$A = \begin{bmatrix} 4 & -1 & -1 & & & \\ -1 & 4 & -1 & -1 & & \\ -1 & 4 & -1 & -1 & -1 & \\ & \ddots & \ddots & \ddots & \ddots & \ddots \\ & & -1 & -1 & 4 & -1 & -1 \\ & & & -1 & -1 & 4 & -1 \\ & & & & -1 & -1 & 4 \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} 0.2 \\ 0.2 \\ 0.2 \\ \vdots \\ 0.2 \\ 0.2 \\ 0.2 \end{bmatrix}$$

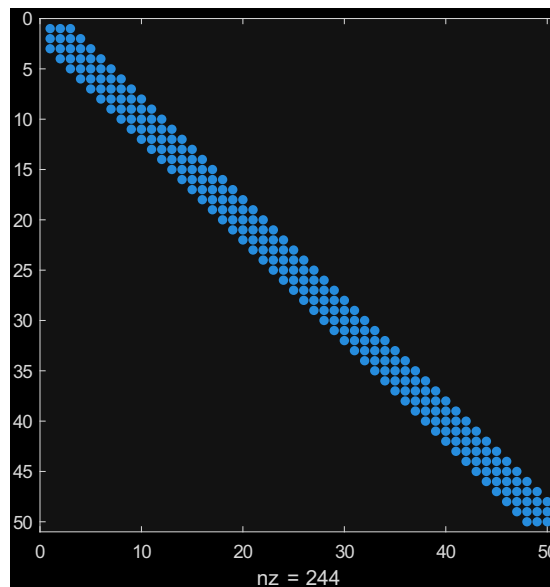
Si vuole risolvere tale problema con il metodo del gradiente, soddisfacendo una tolleranza di 10^{-5} per il criterio relativo al residuo normalizzato, a partire dal vettore iniziale $\mathbf{x}_0 = [0, \dots, 0]^T$.

1. Costruire la matrice A con $n = 50$, il termine noto \mathbf{b} ed il vettore soluzione iniziale \mathbf{x}_0 . La matrice A è simmetrica e definita positiva? Se ne calcoli il numero di condizionamento, senza utilizzare il comando `cond` di Matlab.

```

1 n = 50;
2 A = diag(4*ones(n,1)) + ...
3     diag(-ones(n-1,1),-1) + ....
4     diag(-ones(n-1,1),1) + ...
5     diag(-ones(n-2,1),-2) + ...
6     diag(-ones(n-2,1),2);
7 % si visualizza la sparsity pattern della matrice
8 spy(A)
9 % si costruiscono i restanti vettori
10 b = 0.2*ones(n, 1);
11 x0 = zeros(n, 1);
12
13 % si controlla se e' simmetrica e definita positiva
14 % sfruttando la velocita' della fattorizzazione di Cholesky
15 try chol(A);
16     disp("La matrice e' simmetrica definita positiva")
17 catch ME
18     disp("La matrice non e' simmetrica definita positiva")
19 end
20 eigA = eig(A);
21 KA = max(eigA)/min(eigA)
22 % KA =
23 %
24 %      336.2412

```



2. Si scriva una funzione che implementi il metodo del gradiente e la si usi per risolvere il sistema lineare $Ax = b$. L'intestazione della funzione sarà ad esempio:

```
[x, iter, err] = graddyn(A, b, x0, nmax, toll)
```

Dove **err** è il vettore contenente il residuo normalizzato ad ogni iterazione.

```
1 function [xn, iter, err] = graddyn (A, b, x0, nmax, tol)
2
3 % Metodo del gradiente per sistemi lineari
4 %
5 % Parametri di ingresso:
6 %   A       Matrice del sistema
7 %   b       Termine noto
8 %   x0      Vettore iniziale
9 %   nmax    Numero massimo di iterazioni
10 %   tol     Tolleranza sul test d'arresto
11 %
12 % Parametri in uscita
13 %   xn      Vettore soluzione
14 %   iter    Iterazioni effettuate
15 %   err     Vettore contenente gli errori relativi sul residuo
16
17 [n, m] = size (A);
18 if not(n == m)
19     error ('matrice non quadrata')
20 end
21
22 iter = 0;
23 xn = zeros(n,1);
24
25 % Iterazioni
26 bnorm2 = norm (b);
27 r = b - A * x0;
28 errk = norm (r) / bnorm2;
29 err = errk;
```

```

30 xn = x0;
31
32 while (errk > tol) && (iter < nmax)
33     z = r;
34     transpose_z = transpose(z);
35     alpha = transpose_z*r / (transpose_z*A*z);
36     xn = xn + alpha * r;
37     r = (eye(n)-alpha*A)*r;
38     errk = norm(r) / bnorm2;
39     err = [err errk];
40     % xv = xn;
41     iter = iter + 1;
42 end
43
44 if (iter == nmax)
45     fprintf('Il metodo graddyn non converge in %d iterazioni \n', iter);
46 end

```

3. Utilizzare la funzione scritta al punto precedente per determinare la soluzione del sistema lineare $Ax = b$.

```

1 % si impostano i valori
2 toll = 1e-5;
3 nmax = 10000;
4
5 % inizia il timer
6 tic
7 % si invoca il metodo del gradiente
8 [xGD, iterGD, errGD] = graddyn(A, b, x0, nmax, toll);
9 % si salva il tempo finale di esecuzione
10 timeGD=toc;

```

4. A partire dal `graddyn.m`, si scriva una funzione `gradprec.m` che implementi il metodo del gradiente preconditionato. L'intestazione della funzione sarà ad esempio:

$$[x, iter, err] = gradprec(A, b, x0, nmax, toll)$$

Si risolva il sistema lineare $Ax = b$ applicando il metodo del gradiente preconditionato con il preconditionatore:

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

```

1 function [xn, iter, err] = gradprec (A, b, P, x0, nmax, tol)
2
3 % Metodo del gradiente preconditionato
4 %
5 % Parametri di ingresso:
6 % A      Matrice del sistema
7 % b      Termine noto
8 % P      Precondizionatore
9 % x0     Vettore iniziale

```

```

10 % nmax Numero massimo di iterazioni
11 % tol Tolleranza sul test d'arresto
12 %
13 % Parametri in uscita
14 % xn Vettore soluzione
15 % iter Iterazioni effettuate
16 % err Vettore contenente gli errori relativi sul residuo
17
18 [n, m] = size (A);
19 if not(n == m)
20     error ('matrice non quadrata')
21 end
22
23 iter = 0;
24 xn = zeros(n,1);
25
26 % Iterazioni
27 bnorm2 = norm (b);
28 r = b - A * x0;
29 errk = norm (r) / bnorm2;
30 err = errk;
31 xv = x0;
32
33 while (errk > tol) && (iter < nmax)
34     z = P\r;
35     transpose_z = transpose(z);
36     alpha = transpose_z*r / (transpose_z*A*z);
37     xn = xv + alpha * z;
38     r = r-alpha*A*z;
39     errk = norm (r) / bnorm2;
40     err = [err errk];
41     xv = xn;
42     iter = iter + 1;
43 end
44
45 if (iter == nmax)
46     fprintf('Il metodo gradprec non converge in %d iterazioni\n', iter);
47 end

```

```

1 P = diag(2*ones(n,1)) - ...
2     diag(ones(n-1,1),-1) - ...
3     diag(ones(n-1,1),1);
4
5 tic
6 [xPG, iterPG, errPG] = gradprec(A, b, P, x0, nmax, toll);
7 timePG=toc;

```

5. Disegnare su un grafico l'andamento del residuo normalizzato:

$$\frac{\|\mathbf{r}^{(k)}\|}{\|\mathbf{b}\|}$$

In funzione delle iterazioni k nei due casi (gradiente e gradiente preconditionato), e confrontare le curve ottenute.

```

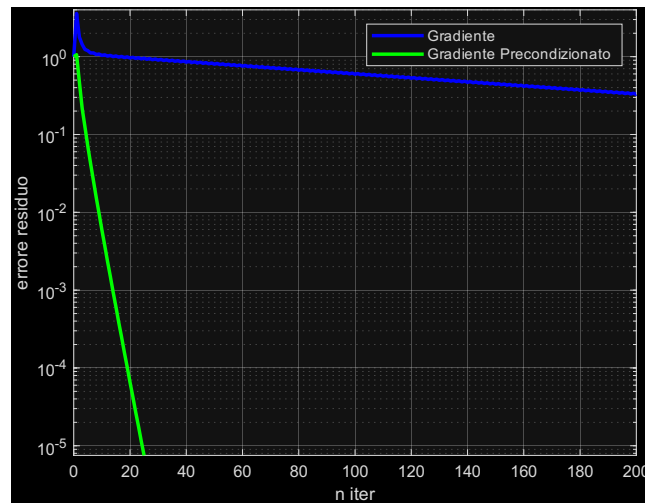
1 figure
2 semilogy(...
3     [0:iterGD], ...
4     errGD, ...
5     'b-', ...

```

```

6     [0:iterPG], ...
7     errPG, ...
8     'g-', ...
9     'Linewidth', ...
10    2 ...
11 )
12 grid on
13 axis([0 200 0 4])
14 xlabel('n iter')
15 ylabel('errore residuo')
16 legend('Gradiente', 'Gradiente Precondizionato')

```



6. A partire da `richardson.m`, si scriva una funzione `richprec.m` che implementi il metodo di Richardson preconditionato. L'intestazione della funzione sarà ad esempio:

```
[x, iter, err] = richprec(A, b, P, alpha, x0, nmax, toll)
```

Si risolva il sistema lineare $Ax = b$ applicando il metodo di Richardson preconditionato utilizzando la relativa α_{opt} e il preconditionatore P implementato al punto 4.

```

1 function [x, it, err] = richprec(A,b,P, alpha, x0,nmax, toll)
2
3 % Metodo di Richardson stazionario preconditionato
4 %
5 % A: matrice del sistema
6 % b: termine noto
7 % P: preconditionatore
8 % x0: vettore iniziale
9 % alpha: coefficiente di Richardson
10 % toll: tolleranza sul residuo normalizzato
11 % nmax: massimo numero di iterazioni
12 %
13 % x: soluzione ottenuta
14 % it: numero di iterazioni effettuate
15 % err: vettore contenente gli errori relativi sul residuo
16
17

```

```

18 n = size(b,1);
19 if ( ...
20     not(size(A,1) == size (A,2)) || ...
21     not(size(A,1) == n) || ...
22     not(size(x0,1) == n) || ...
23     not(size (P,2) == n) || ...
24     not(size(P,1) == n) ...
25 )
26     error('Dimesioni incompatibili')
27 end
28
29 it = 0;
30 x = x0;
31 r = b - A*x;
32 errk = norm(r)/norm(b);
33 err = errk;
34
35 % loop
36 while (it < nmax && errk > toll)
37     it = it + 1;
38     z = P\r;
39     x = x + alpha*z;
40     r = b - A*x;
41     errk = norm(r)/norm(b);
42     err = [err; errk];
43 end

1 lambda_minRP = min(eig(P\A));
2 lambda_maxRP = max(eig(P\A));
3 alpha_optRP = 2/(lambda_maxRP + lambda_minRP);
4 tic
5 [xRP, iterRP, errRP] = richprec(A, b, P, alpha_optRP, x0, nmax
6     , toll);
7 timeRP=toc;

```

7. Si risolva il sistema lineare $Ax = b$ con il metodo di Richardson. Disegnare su un grafico l'andamento del residuo normalizzato:

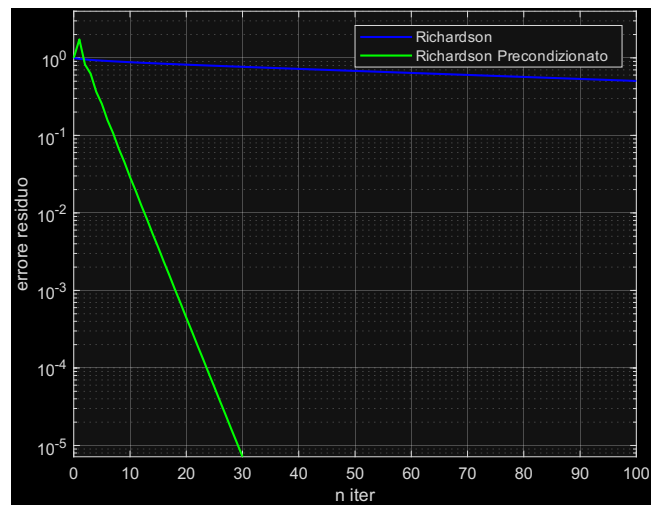
$$\frac{\|r^{(k)}\|}{\|b\|}$$

In funzione delle iterazioni k nei due casi (Richardson e Richardson preconditionato), e confrontare le curve ottenute.

```

1 lambda_minR = min(eig(A));
2 lambda_maxR = max(eig(A));
3 alpha_optR = 2/(lambda_maxR + lambda_minR);
4
5 tic
6 [xR, iterR, errR] = richardson(A, b, x0, alpha_optR, toll,
7     nmax);
8 timeR=toc;
9
10 figure
11 semilogy([0:iterR], errR, 'b-', [0:iterRP], errRP, 'g-', '
12     Linewidth', 1.2)
13 grid on
14 axis([0 100 0 4])
15 xlabel('n iter')
16 ylabel('errore residuo')
17 legend('Richardson', 'Richardson Precondizionato')

```



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

A

algoritmo del metodo del gradiente 37

aritmetica esatta 23

aritmetica floating-point 23

B

ben condizionata 25

C

cancellazione di cifre significative 23

D

decomposizione additiva 34

definita positiva 20

differenza fra due iterate consecutive 9

direzione di discesa per Φ 36

E

epsilon macchina 22

errore assoluto 22

errore computazionale 23

errore computazionale assoluto 23

errore di arrotondamento 22

errore di troncamento 23

errore relativo 22

F

fattore di convergenza 15

fattorizzazione di Cholesky 21

fattorizzazione LU 20

fill-in 27

floating-point normalizzati 22

formula di Cramer 17

funzione di iterazione 13

I

iterazioni di punto fisso 13

M

mal condizionata 25

matrice a dominanza diagonale per colonne 20

matrice a dominanza diagonale per righe 20

matrice a dominanza diagonale stretta per righe 31

matrice di iterazione 29

matrice di permutazione 21

matrice diagonale dominante stretta per righe 31

matrice Jacobiana	11
matrice sparsa	28
matrice tridiagonale	31
matrici a dominanza diagonale stretta	20
metodi di discesa	36
metodo del gradiente	37
metodo del gradiente coniugato	38
metodo del gradiente preconditionato	39
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 Gauss-Seidel	33
metodo di Jacobi	31
metodo di Newton	8
metodo di Richardson	34
metodo di Richardson dinamico	34
metodo di Richardson stazionario	34, 35
metodo iterativo	4, 29
N	
norma dell'energia	35, 38
numero di condizionamento (spettrale) della matrice	24
numero di condizionamento in norma 2	24
O	
overflow	23
P	
pattern	28
pivoting per righe	21
pivoting totale	26
precondizionatore	34, 39
precondizionatore destro	39
precondizionatore sinistro	39
punto fisso	13
R	
raggio spettrale	30
residuo	9, 25
residuo preconditionato $\mathbf{z}^{(k)}$	34
S	
sistema preconditionato	39
splitting	34
T	
teorema di Ostrowski	15
U	
underflow	23

unità di arrotondamento	22
-------------------------	----