Università degli Studi della Basilicata

Dipartimento di Scienze di Base e Applicate - DISBA

Calcolo Scientifico

Appunti delle lezioni ed esercizi rielaborati

Studente:
Donato Martinelli
Matr. 69060

Docente:
Prof.ssa **Maria Carmela De Bonis**

Anno Accademico 2025/2026

Contents

1 Introduzione

Il Ruolo della Matematica e la Modellizzazione La matematica è uno strumento indispensabile per l'interpretazione e la predizione dei fenomeni reali, ponendosi alla base di tutte le scienze. Gli scienziati cercano di ricavare un modello matematico (variabili, parametri, relazioni) che sia rigoroso e coerente con il fenomeno. La complessità del fenomeno determina la quantità di dati necessari e il numero di variabili del modello. Le proprietà incognite si deducono risolvendo tale modello.

Necessità dell'Approccio Numerico Spesso la soluzione non è disponibile in forma esplicita o non è calcolabile analiticamente, rendendo necessario un metodo numerico di approssimazione. Anche quando la risoluzione analitica è possibile, essa può risultare troppo onerosa computazionalmente all'aumentare delle dimensioni del problema.

Esempi Introduttivi Per chiarire la necessità dell'approssimazione, consideriamo alcuni esempi:

- **Equazione di secondo grado:** Data l'equazione $ax^2+bx+c=0$, le soluzioni sono $x_{1,2}=\frac{-b\pm\sqrt{b^2-4ac}}{2a}$. Se $a=1, b=1, c=-1$, le soluzioni sono $\frac{-1\pm\sqrt{5}}{2}$. Per individuare il valore numerico è necessario approssimare $\sqrt{5}$.
- **Equazione esponenziale:** La soluzione di $2^x=100$ è $x=\log_2(100)$. Anche qui, la risposta numerica richiede un'approssimazione.
- **Equazione trascendente:** L'equazione $2^x+x^2-3=0$ non è risolvibile analiticamente (non esplicitabile rispetto a x). Sebbene studiabile graficamente, il calcolo delle soluzioni richiede obbligatoriamente un metodo numerico.

Definizione di Metodo Numerico Un metodo numerico è un processo implementabile su un calcolatore che fornisce una soluzione numerica in un numero finito di passi. Esistono metodi matematicamente validi che però non sono implementabili in un calcolatore per via dei tempi di esecuzione.

Caso Studio: Sistemi Lineari (Cramer vs Gauss) Un esempio cruciale riguarda la risoluzione di sistemi lineari, problema centrale nella modellistica.

- **Il Metodo di Cramer (Non implementabile):** Se volessimo risolvere un sistema 20×20 ($n=20$) con Cramer, dovremmo calcolare 21 determinanti di ordine 20. Con la regola di Laplace, le operazioni sono circa $21\cdot 20!\approx 5.2\cdot 10^{19}$. Su un processore da 3GHz ($3\cdot 10^9$ op/sec), il tempo richiesto sarebbe superiore a 5 secoli. È una procedura con tempo di esecuzione proibitivo.
- **Il Metodo di Eliminazione di Gauss (Efficiente):** Questo metodo richiede circa $\frac{n^3}{3}$ operazioni. Per $n=20$, servono solo 2667 operazioni ($0.88\cdot 10^{-6}$ secondi). Per $n=1000$, servono 334 milioni di operazioni, eseguibili in un decimo di secondo.

Criteri di Scelta ed Errori La scelta del metodo si basa su due criteri fondamentali:

1. **Efficienza:** minor numero di operazioni richieste.
2. **Efficacia:** maggiore precisione di calcolo.

Nonostante la potenza dei calcolatori moderni, i matematici affrontano ancora difficoltà. Poiché i numeri sono rappresentati in binario su una memoria finita, ogni memorizzazione introduce un errore .

Propagazione dell'Errore Un tema centrale del Calcolo Scientifico è la consapevolezza dell'errore e della sua propagazione. Alcuni metodi propagano gli errori (introdotti nei dati iniziali) al punto da fornire soluzioni completamente sbagliate. Si studia quindi la stabilità degli algoritmi rispetto a questi errori.

2 Rappresentazione dei Numeri in un Calcolatore

Sistemi di Numerazione Il numero è un concetto che esiste indipendentemente dall'insieme dei simboli o segni che si utilizzano per rappresentarlo. Un sistema di numerazione non è altro che uno schema che permette di rappresentare i numeri "ideali" attraverso un insieme di simboli per i quali vengono definite delle tecniche di manipolazione (operazioni aritmetiche). E' indubbio che tutti i popoli impararono a contare, spinti dalle necessità della vita quotidiana, per quantificare un insieme di elementi. Gli uomini primitivi limitarono la propria conoscenza ai primi numeri naturali, per indicare i quali ricorsero a nomi di oggetti concreti, io per indicare 1, ali per indicare 2, mano per indicare 5. In seguito incominciarono a rappresentare i numeri con delle barrette verticali. Ogni intero era rappresentato con un numero di barrette pari alle unità in esso contenute. Questo sistema benché sia il più semplice possibile, presenta lo svantaggio di assegnare a ciascun numero una rappresentazione la cui lunghezza è proporzionale al numero stesso. Così, ben presto, si ebbe la necessità di rappresentare l'infinità dei numeri con un numero limitato di segni particolari, detti cifre e di fissare alcuni numeri fondamentali che facevano da riferimento. Nacque così l'idea di quella che viene chiamata base di numerazione. Quasi tutti i popoli scelsero come base di numerazione il 10. Scelta dovuta sicuramente al numero delle dita delle mani. I sistemi di numerazioni delle civiltà conosciute, a partire dalle popolazioni primitive, si classificano in:

- sistema di numerazione additivo
- sistema di numerazione posizionale

Un sistema di numerazione additivo è un sistema di numerazione basato su una legge additiva applicata a determinati simboli numerici fondamentali. Ogni numero è rappresentato attraverso una successione di tali simboli ed il suo valore è dato dalla somma dei valori attribuiti a ciascuno di essi. Nei sistemi additivi non serve un simbolo per lo zero. Tra i più famosi sistemi additivi, oltre a quello delle barrette, figurano quello romano e quello egizio. Solo verso il X secolo il sistema di numerazione additivo incominciò ad essere soppiantato dal sistema di numerazione posizionale introdotto dai matematici indiani ed arabi. Un sistema di numerazione si dice posizionale se i simboli usati per scrivere i numeri assumono valori diversi a seconda della posizione che occupano nella notazione.

Sistema decimale Il sistema di numerazione da noi comunemente usato per rappresentare i numeri è quello decimale. Esso è un sistema di numerazione posizionale basato sull'impiego di 10 cifre 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, ed è stato inventato dagli Indiani. Agli Indiani si deve l'introduzione dello zero che essi chiamavano sunya (nulla) anche se alcuni studiosi affermano che già i Cinesi conoscevano il metodo posizionale fin da tempi antichissimi. Questo sistema di numerazione venne introdotto in occidente dagli Arabi probabilmente nel XII secolo quando venne tradotta *Algoritmi de numero Indorum* del grande matematico arabo al-Khuwarizmi (IX secolo). La base del sistema decimale è 10 e ogni numero viene rappresentato come

$$a = \pm a_m a_{m-1} \dots a_1 a_0 . a_{-1} a_{-2} \dots a_{-M}$$

con

$$0 \leq a_i \leq 9$$

Il sistema è posizionale perché il valore di ogni cifra varia in funzione della sua posizione nella rappresentazione decimale del numero

$$z = \pm a_m 10^m + a_{m-1} 10^{m-1} + \dots + a_1 10^1 + a_0 10^0 + a_{-1} 10^{-1} + a_{-2} 10^{-2} + \dots$$

Esempio 2.1.

- $931.57 = 9 \times 10^2 + 3 \times 10^1 + 1 \times 10^0 + 5 \times 10^{-1} + 7 \times 10^{-2}$
- $34.002 = 3 \times 10^1 + 4 \times 10^0 + 0 \times 10^{-1} + 0 \times 10^{-2} + 2 \times 10^{-3}$
- $12600.09 = 1 \times 10^4 + 2 \times 10^3 + 6 \times 10^2 + 0 \times 10^1 + 0 \times 10^0 + 0 \times 10^{-1} + 9 \times 10^{-2}$

La rappresentazione decimale di ogni numero reale è unica, eccetto quando la parte frazionaria contiene una sequenza di 9 consecutivi.

Esempio 2.2.

- $0.319999...9... = 0.32$
- $10.99999...9... = 11$
- $199.99999...9... = 200$

Sistema in base N Qualunque intero $N > 1$ può essere scelto come base ed ogni numero reale a può essere scritto nella forma

$$z = \pm a_m N^m + a_{m-1} N^{m-1} + \cdots + a_1 N^1 + a_0 N^0 + a_{-1} N^{-1} + a_{-2} N^{-2} + \ldots$$

ovvero

$$a = \pm a_m a_{m-1} \ldots a_1 a_0 . a_{-1} a_{-2} \ldots a_{-M}$$

con $0 \leq a_i \leq N - 1$. La rappresentazione di ogni numero reale in base N è unica, eccetto quando la parte frazionaria contiene una sequenza di cifre $a_k = N - 1$ consecutive. Più piccola è la base scelta, più è lunga la stringa di caratteri necessari per rappresentare lo stesso numero.

Sistema binario La base del sistema binario è 2. Le cifre utilizzate da questo sistema sono 0 e 1 e vengono dette bit da *binary digit*. Ogni numero reale a è rappresentato come una sequenza di 0 e 1, ovvero

$$a = \pm a_m 2^m + a_{m-1} 2^{m-1} + \cdots + a_1 2^1 + a_0 2^0 + a_{-1} 2^{-1} + a_{-2} 2^{-2} + \ldots$$

con $0 \leq a_i \leq 1$.

Questo sistema è particolarmente interessante perché può essere realizzato con qualsiasi oggetto capace di assumere due stati diversi, uno per la cifra 0 e l'altro per la cifra 1.

Esempio 2.3.

- Stato di magnetizzazione e non di un nucleo di ferrite;
- conduttività e non di un diodo.

Per queste sue caratteristiche è stato adottato per la rappresentazione dei dati e, in particolare, dei numeri in un calcolatore.

Rappresentazione dei numeri in un calcolatore I numeri vengono rappresentati nel calcolatore secondo il sistema binario e, quindi, come una sequenza di bit. Per ovvie ragioni, ad ogni numero reale viene riservato uno spazio finito di memoria, capace di contenere un numero finito di bit detto **parola**. Di conseguenza, non tutti i numeri reali sono rappresentabili in modo esatto. Detta l la lunghezza della parola, si possono rappresentare esattamente solo quei numeri la cui rappresentazione binaria consta di un numero di bit inferiore o uguale ad l. Si parla di **numeri macchina**. Tutte le operazioni fra i numeri macchina vengono effettuate utilizzando l'aritmetica binaria.

Numeri interi Con parole di lunghezza l è possibile rappresentare tutti i numeri interi appartenenti all'intervallo

$$\left[-\frac{2^l}{2}, \frac{2^l}{2} - 1 \right]$$

Esempio: Se $l = 16$ sono rappresentabili tutti i numeri interi appartenenti all'intervallo

$$[-32768, 32767]$$

Numeri reali Ogni numero reale a può essere scritto nella forma

$$a = pN^q$$

dove p è un numero reale, N è la base del sistema di numerazione e q è un numero intero positivo o negativo. Questa rappresentazione, detta in virgola mobile (floating-point), non è unica, infatti

$$321.25 = 32.125 \times 10^1 = 0.32125 \times 10^3$$

La rappresentazione di a si dice **normalizzata** quando

$$N^{-1} \leq |p| < 1$$

Le cifre di p si dicono **cifre significative**.

Esempio 2.4.

- La rappresentazione normalizzata di $a = 92.25$ è $a = 0.9225 \times 10^2$;
- la rappresentazione normalizzata di $a = 0.000718$ è $a = 0.718000 \times 10^{-3}$;
- la rappresentazione normalizzata di $a = 4152.0002156$ è $a = 0.41520002156 \times 10^4$;

- la rappresentazione normalizzata di $a = 0.0215600000$ è $a = 0.2156 \times 10^{-1}$.

Fissata la base N, ogni numero reale a è univocamente definito dalla coppia

$$a = (p, q)$$

p viene detta **mantissa** di a, q viene detto **esponente** di a. I numeri reali in virgola mobile vengono rappresentati in un calcolatore in forma normalizzata secondo lo Standard IEEE 754 (Institute of Electrical and Electronical Engineering).

Standard IEEE 754: Singola Precisione

Definizione 2.1. La struttura è definita dai seguenti parametri:

$$s = 1 \text{ bit, bit } 31$$

$$p = 23 \text{ bit}$$

$$q = 8 \text{ bit}$$

È possibile rappresentare tutti i numeri reali della seguente forma:

$$\pm 0.d_1d_2d_3d_4d_5d_6 \times 10^q$$

dove:

$$0 < d_1 \leq 9, \quad 0 \leq d_i \leq 9, \quad i \in \{2, \ldots, 6\}$$

e

$$-38 \leq q \leq 38$$

Si hanno **6 cifre significative**.

Standard IEEE 754: Doppia Precisione

Definizione 2.2. La struttura è definita dai seguenti parametri:

$$s = 1 \text{ bit (bit 63)}$$

$$q = 11 \text{ bit (bit 62-52)}$$

$$p = 52 \text{ bit (bit 51-0)}$$

È possibile rappresentare tutti i numeri reali della seguente forma:

$$\pm 0.d_1d_2 \ldots d_{16} \times 10^q$$

dove:

$$0 < d_1 \leq 9, \quad 0 \leq d_i \leq 9, \quad i \in \{2, \ldots, 16\}$$

e

$$-308 \leq q \leq 308$$

Si hanno **16 cifre significative**.

Esempio 2.5. Consideriamo due numeri reali da convertire nei rispettivi standard IEEE 754.

1. **Esempio in Singola Precisione (32 bit)** Rappresentiamo il numero $v = -13.625$.

   - **Segno:** Il numero è negativo, quindi $s = 1$.
   - **Conversione Binaria:**

     $$13_{10} = 1101_2$$

     $$0.625_{10} = 0.101_2 \quad (0.5 + 0.125)$$

     Unendo le parti: 1101.101_2.

- **Normalizzazione:** Spostiamo la virgola a sinistra fino ad avere $1.m$:

$$1.101101 \times 2^3$$

  L'esponente reale è 3.

- **Esponente Biased ($q = 8$):** Si aggiunge il bias 127 ($2^{q-1} - 1$):

$$3 + 127 = 130 \implies 10000010_2$$

- **Mantissa ($p = 23$):** Si prendono i bit dopo la virgola (il bit intero 1 è implicito):

$$10110100000000000000000$$

Rappresentazione finale:

$$\underbrace{1}_{s} \ \underbrace{10000010}_{q \text{ (esp)}} \ \underbrace{10110100000000000000000}_{p \text{ (mantissa)}}$$

2. Esempio in Doppia Precisione (64 bit) Rappresentiamo il numero $v = 0.15625$.

- **Segno:** Il numero è positivo, quindi $s = 0$.

- **Conversione Binaria:**

$$0.15625_{10} = \frac{5}{32} = \frac{1}{8} + \frac{1}{32} = 2^{-3} + 2^{-5} = 0.00101_2$$

- **Normalizzazione:** Spostiamo la virgola a destra:

$$1.01 \times 2^{-3}$$

  L'esponente reale è -3.

- **Esponente Biased ($q = 11$):** Si aggiunge il bias 1023 ($2^{q-1} - 1$):

$$-3 + 1023 = 1020 \implies 01111111100_2$$

- **Mantissa ($p = 52$):** Si prendono i bit dopo la virgola, riempiendo con zeri:

$$01000000 \ldots 0 \quad \text{(totale 52 bit)}$$

Rappresentazione finale:

$$\underbrace{0}_{s} \ \underbrace{01111111100}_{q \text{ (esp)}} \ \underbrace{0100000000000000000 \ldots 0}_{p \text{ (mantissa)}}$$

Osservazione Nello standard IEEE 754 in Doppia Precisione, benché lo spazio riservato alla mantissa di un numero reale sia di 52 bit, viene recuperato un bit in più non rappresentando il primo bit che è sempre uguale a 1. Pertanto le mantisse p dei numeri reali vengono rappresentate in doppia precisione con 53 bit.

Supponendo di utilizzare un calcolatore che lavora in doppia precisione, dato

$$a = \pm 0.d_1 d_2 \ldots 10^q, \quad d_1 \neq 0$$

un numero reale non nullo, si possono presentare i seguenti casi:

1. L'esponente q è tale che $-308 \leq q \leq 308$ e le cifre dopo la 16-esima sono tutte nulle ovvero $d_i = 0$ per ogni $i > 16$, cioè

$$a = \pm 0.d_1 d_2 \ldots d_{16} 10^q$$

   Allora a è **esattamente rappresentabile**.

2. L'esponente q non appartiene all'intervallo $[-308, 308]$. Se $q < -308$, si associa 0 ad a e il calcolatore segnala **underflow**. Se $q > 308$, a non viene rappresentato e il calcolatore segnala **overflow**.

Osservazione Nei computer di ultima generazione è possibile rappresentare anche numeri il cui esponente q è tale che

$$-324 < q < -308$$

Tali numeri riempiono l'intervallo tra lo zero ed il più piccolo numero normalizzato rappresentabile

$$\text{realmin} = 2^{-1022} \sim 2.22 \times 10^{-308}$$

e vengono chiamati **denormalizzati** (o subnormalizzati). Il più grande numero normalizzato rappresentabile in doppia precisione è

$$\text{realmax} \sim 1.79 \times 10^{308}$$

Esso è un bit meno di 2^{1024}.

3. L'esponente q appartiene all'intervallo $[-308, 308]$ ma le cifre d_i, $i > 16$, non sono tutte nulle. In questo caso è possibile associare al numero a un numero di macchina $fl(a)$ seguendo due criteri diversi:
   - troncamento
   - arrotondamento

Regola di troncamento Per troncare un numero

$$a = 0.d_1 d_2 \dots d_{t-1} d_t d_{t+1} d_{t+2} \dots N^q$$

fino a t cifre significative, si eliminano tutte le cifre $d_{t+1} d_{t+2} \dots$ a destra della t-esima. Dunque

$$fl(a) = trn(a) = 0.d_1 d_2 \dots d_{t-1} d_t N^q$$

Regola di arrotondamento Per arrotondare un numero

$$a = 0.d_1 d_2 \dots d_{t-1} d_t d_{t+1} d_{t+2} \dots N^q$$

fino a t cifre significative, si eliminano tutte le cifre $d_{t+1} d_{t+2} \dots$ a destra della t-esima e si rimpiazza d_t con la cifra c, dove

$$\begin{cases} c = d_t & \text{se } 0 \le d_{t+1} \le 4 \\ c = d_t + 1 & \text{se } 5 \le d_{t+1} \le 9 \end{cases}$$

Dunque

$$fl(a) = arr(a) = 0.d_1 d_2 \dots d_{t-1} c N^q$$

Esempio 2.6.

- Consideriamo il numero $\pi = 3.1415926536 \cdots = 0.31415926536 \dots 10^1$.
  - $t = 6$: $trn(\pi) = 0.314159 \cdot 10^1$, $arr(\pi) = 0.314159 \cdot 10^1$
  - $t = 5$: $trn(\pi) = 0.31415 \cdot 10^1$, $arr(\pi) = 0.31416 \cdot 10^1$
  - $t = 4$: $trn(\pi) = 0.3141 \cdot 10^1$, $arr(\pi) = 0.3142 \cdot 10^1$
- Consideriamo il numero $a = 2\sqrt{2} = 2.828427124746190 \cdots = 0.2828427124746190 \dots 10^1$.
  - $t = 6$: $trn(a) = 0.282842 \cdot 10^1$, $arr(a) = 0.282843 \cdot 10^1$
  - $t = 5$: $trn(a) = 0.28284 \cdot 10^1$, $arr(a) = 0.28284 \cdot 10^1$
  - $t = 4$: $trn(a) = 0.2828 \cdot 10^1$, $arr(a) = 0.2828 \cdot 10^1$

Vantaggi della rappresentazione normalizzata Consideriamo il numero

$$a = \frac{1}{7000}$$

Poniamo

$$\overline{a} = 0.000142857142857 \dots \quad \text{(rappresentazione non normalizzata)}$$
$$\overline{\overline{a}} = 0.142857142857142 \dots 10^{-3} \quad \text{(rappresentazione normalizzata)}$$

Applichiamo le regole di troncamento e arrotondamento ad $\overline{a}$:

- $t = 6$: $trn(\overline{a}) = 0.000142$, $arr(\overline{a}) = 0.000143$

- $t = 3$: $trn(\overline{a}) = 0$, $arr(\overline{a}) = 0$

Applichiamo le regole di troncamento e arrotondamento ad $\overline{\overline{a}}$:

- $t = 6$: $trn(\overline{\overline{a}}) = 0.142857 \cdot 10^{-3}$, $arr(\overline{\overline{a}}) = 0.142857 \cdot 10^{-3}$
- $t = 3$: $trn(\overline{\overline{a}}) = 0.142 \cdot 10^{-3}$, $arr(\overline{\overline{a}}) = 0.143 \cdot 10^{-3}$

Dunque, la normalizzazione permette di ridurre l'errore di rappresentazione dovuto al troncamento o all'arrotondamento ad un numero finito di cifre. Inoltre, l'informazione contenuta negli zeri dopo il punto può essere memorizzata, con minor spreco di memoria, in termini di esponente.

Conseguenze della rappresentazione dei numeri come parole di lunghezza fissa

Prima conseguenza: Limitatezza dell'insieme dei numeri rappresentabili L'insieme $\mathbb{R}$ dei numeri reali è infinito, ma, soltanto un sottoinsieme di $\mathbb{R}$, limitato inferiormente e superiormente, è rappresentabile in un calcolatore. In seguito denoteremo con $\mathbb{F}$ il sottoinsieme dei numeri reali rappresentabili in un calcolatore.

$$\mathbb{F} = \{0\} \cup \{\pm 0.d_1 d_2 \dots d_{16} \times 10^q \in \mathbb{R} : -308 \leq q \leq 308\}$$

$$0 < d_1 \leq 9, \quad 0 \leq d_i \leq 9, \quad i \in \{2, \dots, 16\}$$

Si ha l'intervallo contenente valori rappresentabili tra -realmax e -realmin, lo zero, e tra realmin e realmax. Al di fuori di questo intervallo si ha overflow o underflow.

Seconda conseguenza: Insieme dei numeri rappresentabili "bucato" L'insieme $\mathbb{R}$ dei numeri reali è denso, cioè tra due numeri reali r_1 e r_2 esiste sempre un numero reale r_3 tale che

$$r_1 < r_3 < r_2$$

Invece, il sottoinsieme $\mathbb{F}$, oltre ad essere limitato inferiormente e superiormente, è anche bucato.

Esempio Utilizzando la rappresentazione in doppia precisione, i numeri macchina

$$a = 1.123456789123456$$

e

$$b = 1.123456789123457$$

sono consecutivi e tra di essi non esiste alcun altro numero macchina. Tutti i numeri reali compresi tra a e b vengono approssimati con a o con b.

Il Software MatLab MatLab (Matrix Laboratory) è un ambiente di calcolo sviluppato a partire dagli anni 70. La struttura di base è la matrice, per la quale sono già predefinite numerosi tipi elementari (matrice identità, matrice nulla, matrice unità...), funzioni algebriche e di manipolazione (somma, prodotto, calcolo del determinante). Per lanciare MatLab da ambiente Windows basta cliccare con il mouse sull'icona corrispondente. Per entrare in confidenza con l'ambiente di lavoro è utile:

- lanciare il comando `demo` che illustra le potenzialità del software attraverso significativi esempi numerici e casi test;
- fare costante riferimento all'uso dell'`help`, ad esempio `help sqrt` (calcolo della radice quadrata di un numero).

Prime istruzioni in MatLab Il modo più immediato per interagire con MatLab è scrivere l'istruzione dal prompt `>>` seguita da Invio. Esempio: assegnazione del valore 3 alla variabile a:

```
>> a = 3
a =
     3
```

Possiamo usare MatLab come una semplice calcolatrice:

```
>> b = a * 2
b =
     6
```

o, come vedremo, come un vero e proprio ambiente di programmazione. Sia le istruzioni che esegue dal prompt che le routine devono essere scritti utilizzando una speciale sintassi.

Invece di digitare tutti i comandi al prompt, possiamo memorizzare una serie di istruzioni successive (script) sotto formato di file di testo, detto M-file e caratterizzato da estensione .m. A questo scopo possiamo utilizzare l'Editor di testo integrato. Per uscire da MatLab: comandi `quit` o `exit`.

Alcuni trucchi per risparmiare tempo

1. Durante la sessione di lavoro è possibile richiamare i comandi precedentemente digitati utilizzando i tasti $\leftarrow, \rightarrow, \uparrow, \downarrow$;
2. immettendo i primi caratteri di un'istruzione già digitata e poi premendo il tasto $\uparrow$, viene completata la riga con l'ultima istruzione che inizia con quegli stessi caratteri;
3. i tasti $\leftarrow$ e $\rightarrow$ permettono di riposizionare sulla linea di comando il cursore e di modificare il testo scritto;
4. con il tasto sinistro del mouse sulla finestra di calcolo si possono selezionare parti di testo che è poi possibile copiare, tagliare ed incollare sulla linea di comando.

Le variabili in MatLab In MatLab tutte le variabili sono in doppia precisione, ovvero sono rappresentate internamente con 64 bit, cui corrispondono 16 cifre significative. Quando assegnamo un valore ad una variabile, MatLab risponde con un'eco:

```
s = 10
s =
    10
```

Per sopprimere l'eco, usiamo la sintassi:

```
s = 10;
```

Quando non assegnamo il valore di un'operazione ad una variabile, MatLab assegna tale valore alla variabile `ans` (che viene così ogni volta sovrascritta):

```
>> 3^2
ans =
     9
```

Un'osservazione sulla precisione di calcolo Tutti i calcoli vengono effettuati in doppia precisione, mentre diversa è la visualizzazione delle variabili che viene determinata con il comando `format`:

- `format short`: virgola fissa con 5 cifre (è il formato di default):

  ```
  >> pi
  ans =
      3.1416
  ```

- `format long`: virgola fissa con 16 cifre:

  ```
  >> pi
  ans =
      3.141592653589793
  ```

- `format short e`: virgola mobile con 5 cifre:

  ```
  >> pi
  ans =
      3.1416e+00
  ```

- `format long e`: virgola mobile con 16 cifre:

  ```
  >> pi
  ans =
      3.141592653589793e+00
  ```

format long e

Noi useremo sempre il formato `format long e` perché è quello più vicino alla rappresentazione normalizzata del numero. Dunque, prima di incominciare qualunque esercizio digitiamo dal prompt:

```
>> format long e
```

Esercizi in MatLab

- Digitare il numero 0.95842567894152678954126354 e verificare che il programma lo arrotonda a 16 cifre significative fornendo

$$9.584256789415268e-001 = 0.9584256789415268$$

```
>> 0.95842567894152678954126354
ans =
     9.584256789415268e-001
```

- Digitare il numero 0.00000052135658365124858985 e verificare che il programma lo arrotonda a 16 cifre significative fornendo

$$5.21356583651249e-007 = 0.521356583651249e-006$$

```
>> 0.00000052135658365124858985
ans =
     5.213565836512486e-007
```

- Digitare il numero 12356.00056256984154646 e verificare che il programma lo arrotonda a 16 cifre significative fornendo

$$1.23560005625698e+004 = 0.123560005625698e+005$$

```
>> 12356.00056256984154646
ans =
     1.235600056256984e+004
```

- Calcolare `realmin` $= 2^{-1022} \sim 2.225073858507201e-308$

```
>> 2^(-1022)
ans =
     2.225073858507201e-308
```

- Calcolare i valori 2^{-k} con $k = 1040, 1050, 1060, 1070$ e osservare che il programma riesce a rappresentare anche numeri con esponente compreso tra -308 e -324 (numeri denormalizzati).

```
>> k = 1040;
>> 2^(-k)
ans =
     8.487983163861089e-314
>> k = 1050;
>> 2^(-k)
ans =
     8.289046058458095e-317
>> k = 1060;
>> 2^(-k)
ans =
     8.094771541462983e-320
>> k = 1070;
>> 2^(-k)
ans =
     7.90505033459945e-323
```

- Verificare che $2^{-1074} \sim 4.94065645841247e-324$ e che calcolando 2^{-1075} il programma restituisce come valore 0, dunque non riesce a rappresentare numeri con esponente più piccolo di -324.

```
>> k = 1074;
>> 2^(-k)
ans =
     4.940656458412465e-324
>> k = 1075;
>> 2^(-k)
ans =
     0
```

- Verificare che $2^{1023} \sim 8.988465674311580e+307$ e che calcolando 2^{1024} il programma restituisce Inf, dove Inf denota l'overflow.

```
>> k = 1023;
>> 2^k
ans =
     8.988465674311580e+307
>> k = 1024;
>> 2^k
ans =
     Inf
```

2.1 Errori

Cause principali di errori nella risoluzione di problemi matematici con un calcolatore Nella risoluzione di un problema matematico con un calcolatore le sorgenti di errore possono essere, in generale, suddivise in due gruppi.

1. **Errori di rappresentazione dei dati o di arrotondamento**
   Abbiamo visto che non tutti i numeri reali possono essere rappresentati in un calcolatore. Se il calcolatore utilizza una rappresentazione con t cifre per la mantissa, tutti i numeri reali compresi tra l'estremo inferiore e l'estremo superiore di $\mathbb{F}$ la cui mantissa ha un numero di cifre superiore a t dovranno essere arrotondati o troncati a t cifre.
2. **Errori nei calcoli**
   Effettuando calcoli tra numeri macchina approssimati, gli errori di rappresentazione dei dati iniziali si propagano in qualche misura sui risultati di tali calcoli. L'entità della propagazione dipende anche dall'algoritmo utilizzato.

Errori di rappresentazione dei dati Nel rappresentare un numero reale $a \neq 0$, con un corrispondente numero macchina $fl(a)$ occorre valutare l'errore commesso che sarà ovviamente nullo se a è esso stesso un numero di macchina, cioè $a = fl(a)$. A tale scopo, dato un valore reale esatto a e un sua approssimazione A diamo alcune definizioni.

Errore Assoluto

Definizione 2.3. La quantità

$$\Delta a = |a - A|$$

si chiama errore assoluto.

```
function e = errAbs(valExact, valApprox)
%ERRABS Calcola l'errore assoluto con validazione degli input.
%   E = ERRABS(VALEXACT, VALAPPROX) calcola |valExact - valApprox|.
%   Supporta scalari, vettori e matrici multidimensionali.

    arguments
        valExact  {mustBeNumeric}
        valApprox {mustBeNumeric}
    end

    % Controllo manuale delle dimensioni
    if ~isequal(size(valExact), size(valApprox))
        error('Input arguments must be of the same size.');
    end

    e = abs(valExact - valApprox);
end
```

Funzione errore assoluto

Errore Relativo L'errore assoluto non riesce a caratterizzare ciò che intuitivamente si può ritenere una misura della precisione del valore approssimato. Per misurare la bontà di un'approssimazione A di a occorre confrontare l'errore assoluto con l'ordine di grandezza a del dato da approssimare. Cioè fare il rapporto tra l'errore assoluto e il valore assoluto del dato esatto.

Definizione 2.4. La quantità

$$\delta a = \frac{|a - A|}{|a|}$$

si chiama errore relativo.

```
function e = errRel(valExact, valApprox)
%ERRREL Calcola l'errore relativo richiamando errAbs.
%   E = ERRREL(VALEXACT, VALAPPROX) sfrutta la funzione errAbs per il
%   numeratore e normalizza rispetto al valore esatto.

    arguments
        valExact  {mustBeNumeric}
        valApprox {mustBeNumeric}
    end

    % Controllo manuale delle dimensioni
    if ~isequal(size(valExact), size(valApprox))
        error('Input arguments must be of the same size.');
    end

    e = errAbs(valExact, valApprox) ./ abs(valExact);
end
```

Funzione errore relativo

Cifre corrette Siano

$$a = pN^q \quad \text{e} \quad A = \overline{p}N^q$$

con

$$N^{-1} \leq |p|, |\overline{p}| < 1$$

Cifre decimali corrette

Definizione 2.5. Se

$$\Delta a \leq \frac{1}{2} N^{-t}, \quad t \geq 1,$$

si dice che A ha almeno t cifre decimali corrette nella base N.

Partendo dalla disuguaglianza che lega l'errore assoluto Δa al numero di cifre decimali t:

$$\begin{aligned} \Delta a &\leq \frac{1}{2} \cdot 10^{-t} \\ 2 \cdot \Delta a &\leq 10^{-t} \\ \log_{10}(2 \cdot \Delta a) &\leq -t \\ t &\leq -\log_{10}(2 \cdot \Delta a) \end{aligned}$$

Poiché t deve essere un intero, si assume il massimo valore che soddisfa la condizione (parte intera inferiore):

$$t = \lfloor -\log_{10}(2 \cdot \Delta a) \rfloor .$$

```
function n = cifDecCorr(valExact, valApprox)
%CIFDECCORR Calcola il numero di cifre decimali corrette.
%   n = floor(-log10(2 * errAbs(valExact, valApprox)))

    arguments
```

```
        valExact  {mustBeNumeric}
        valApprox {mustBeNumeric}
    end

    % Controllo manuale delle dimensioni
    if ~isequal(size(valExact), size(valApprox))
        error('Input arguments must be of the same size.');
    end

    % Calcolo l'errore assoluto
    err = errAbs(valExact, valApprox);

    n = floor(-log10(2 * err));
end
```

Funzione cifre decimali corrette

Cifre significative corrette

Definizione 2.6. Se

$$\delta a \leq \frac{1}{2} N^{-t+1}, \quad t \geq 1,$$

si dice che A ha almeno t cifre significative corrette nella base N.

Partendo dalla disuguaglianza che lega l'errore relativo δa al numero di cifre significative t:

$$\begin{aligned} \delta a &\leq \frac{1}{2} \cdot 10^{1-t} \\ 2 \cdot \delta a &\leq 10^{1-t} \\ \log_{10}(2 \cdot \delta a) &\leq 1 - t \\ t &\leq 1 - \log_{10}(2 \cdot \delta a) \end{aligned}$$

Poiché t deve essere un intero, si assume il massimo valore che soddisfa la condizione (parte intera inferiore):

$$t = \lfloor 1 - \log_{10}(2 \cdot \delta a) \rfloor .$$

```
function n = cifSigCorr(valExact, valApprox)
%CIFSIGCORR Calcola il numero di cifre significative corrette.
%   n = floor(1-log10(2 * errRel(valExact, valApprox)))

    arguments
        valExact  {mustBeNumeric}
        valApprox {mustBeNumeric}
    end

    % Controllo manuale delle dimensioni
    if ~isequal(size(valExact), size(valApprox))
        error('Input arguments must be of the same size.');
    end

    % Calcolo l'errore assoluto
    err = errRel(valExact, valApprox);

    n = floor(1-log10(2 * err));
end
```

Funzione cifre significative corrette

Osservazione Le cifre significative del numero $a = pN^q$ (scritto nella forma normalizzata) sono le cifre decimali corrette della sua mantissa p. Infatti, supponendo che la mantissa $\overline{p}$ di A approssimi la mantissa p di a con t cifre decimali corrette, tenendo conto che $N^{-1} \leq |p|, |\overline{p}| < 1$, si ha:

$$\delta a = \frac{|p - \overline{p}|N^q}{|p|N^q} = \frac{|p - \overline{p}|}{|p|} \leq \frac{\frac{1}{2}N^{-t}}{N^{-1}} = \frac{1}{2}N^{-t+1}$$

Esempio 2.7. 1. $a = 15.2000$, $A = 15.1997$, $N = 10$

$$\Delta a = 0.3\ldots 10^{-3} < \frac{1}{2}10^{-3}$$

$$\delta a = 0.197\ldots 10^{-4} < \frac{1}{2}10^{-4}$$

Dunque A ha almeno 3 cifre decimali corrette e almeno 5 cifre significative corrette. In realtà ha 3 cifre decimali corrette e 5 cifre significative corrette.

```
a = 15.2000;    % Valore Esatto
A = 15.1997;    % Valore Approssimato

d = cifDecCorr(a, A);
s = cifSigCorr(a, A);

fprintf('Il valore %g, approssimazione di %g, ha almeno %d cifre decimali corrette.\n', A
    , a, d);
fprintf('Il valore %g, approssimazione di %g, ha almeno %d cifre significative corrette.\
    n', A, a, s);
```

```
Il valore 15.1997, approssimazione di 15.2, ha almeno 3 cifre decimali corrette.
Il valore 15.1997, approssimazione di 15.2, ha almeno 5 cifre significative corrette.
```

2. $a = 199.2000$, $A = 199.1997$, $N = 10$

$$\Delta a = 0.3\ldots 10^{-3} < \frac{1}{2}10^{-3}$$

$$\delta a = 0.15\ldots 10^{-5} < \frac{1}{2}10^{-5}$$

Dunque A ha almeno 3 cifre decimali corrette e almeno 6 cifre significative corrette. In realtà ha 3 cifre decimali corrette e 6 cifre significative corrette. *Nota:* Gli errori assoluti degli esempi 1 e 2 sono uguali!

```
a = 199.2000;    % Valore Esatto
A = 199.1997;    % Valore Approssimato

d = cifDecCorr(a, A);
s = cifSigCorr(a, A);

fprintf('Il valore %g, approssimazione di %g, ha almeno %d cifre decimali corrette.\n', A
    , a, d);
fprintf('Il valore %g, approssimazione di %g, ha almeno %d cifre significative corrette.\
    n', A, a, s);
```

```
Il valore 199.2, approssimazione di 199.2, ha almeno 3 cifre decimali corrette.
Il valore 199.2, approssimazione di 199.2, ha almeno 6 cifre significative corrette.
```

3. $a = 1$, $A = 0.9999$, $N = 10$

$$\Delta a = 10^{-4} = 0.1 \cdot 10^{-3} < \frac{1}{2}10^{-3}$$

$$\delta a < \frac{1}{2}10^{-3}$$

Dunque A ha almeno 3 cifre decimali corrette e almeno 4 cifre significative corrette. In realtà ha 4 cifre decimali corrette e 4 cifre significative corrette ma nessuna cifra coincidente!

```
a = 1;    % Valore Esatto
A = 0.9999;    % Valore Approssimato

d = cifDecCorr(a, A);
s = cifSigCorr(a, A);

fprintf('Il valore %g, approssimazione di %g, ha almeno %d cifre decimali corrette.\n', A
    , a, d);
fprintf('Il valore %g, approssimazione di %g, ha almeno %d cifre significative corrette.\
    n', A, a, s);
```

```
Il valore 0.9999, approssimazione di 1, ha almeno 3 cifre decimali corrette.
Il valore 0.9999, approssimazione di 1, ha almeno 4 cifre significative corrette.
```

4. $a = 332.122$, $A = 332.129$, $N = 10$

$$\Delta a = 0.7 \ldots 10^{-2} = \frac{1}{2} 0.14 \ldots 10^{-1} < \frac{1}{2} 10^{-1}$$

$$\delta a = 0.2 \ldots 10^{-4} < \frac{1}{2} 10^{-4}$$

Dunque A ha almeno 1 cifra decimale corretta e almeno 5 cifre significative corrette. In realtà ha 2 cifre decimali corrette e 5 cifre significative corrette.

```
a = 332.122;    % Valore Esatto
A = 332.129;    % Valore Approssimato

d = cifDecCorr(a, A);
s = cifSigCorr(a, A);

fprintf('Il valore %g, approssimazione di %g, ha almeno %d cifre decimali corrette.\n', A
    , a, d);
fprintf('Il valore %g, approssimazione di %g, ha almeno %d cifre significative corrette.\
    n', A, a, s);
```

```
Il valore 332.129, approssimazione di 332.122, ha almeno 1 cifre decimali corrette.
Il valore 332.129, approssimazione di 332.122, ha almeno 5 cifre significative corrette.
```

Unità di roundoff

Teorema 2.1. *Denotando con $A = fl(a)$ il numero macchina che si ottiene arrotondando il numero reale a a t cifre significative nella base N, si ha*

$$\frac{|fl(a) - a|}{|a|} \leq u$$

dove

$$u = \frac{1}{2} N^{1-t}$$

La quantità u è detta ***unità di roundoff****.*

Il teorema precedente ci dice che ogni numero può essere rappresentato su un calcolatore con un errore che non eccede u. Infatti, ponendo $\delta = \frac{fl(a)-a}{a}$, si ha

$$fl(a) = a(1 + \delta), \quad |\delta| \leq u$$

La quantità u è una costante propria di ciascuna aritmetica floating-point e rappresenta la massima precisione di calcolo raggiungibile con un calcolatore su cui tale aritmetica è implementata.

Epsilon-macchina Si chiama invece **precisione di macchina** o **epsilon di macchina** la distanza tra due numeri macchina consecutivi. È possibile dimostrare che l'epsilon di macchina è esattamente il doppio della unità di roundoff, cioè $\varepsilon = 2u$. Dunque parleremo indifferentemente di unità di roundoff o di precisione di macchina. Nel caso dell'insieme $\mathbb{F}$ (doppia precisione standard IEEE 754, con 53 bit di mantissa inclusa quella nascosta), l'unità di roundoff è uguale a:

$$u = \frac{1}{2} 2^{1-53} = 2^{-53} \approx 1.110223024625157e - 016$$

e quindi

$$\varepsilon = 2^{-52} \approx 2.220446049250313e - 016$$

La precisione di macchina ε rappresenta l'ampiezza del buco esistente tra i numeri macchina consecutivi:

$$1.000000000000000$$

e

$$1.000000000000001$$

Esercizi in MatLab Dato il numero reale a e la sua approssimazione A calcolare l'errore assoluto Δa e l'errore relativo δa e stabilire quante sono le cifre decimali corrette e le cifre significative corrette di A.

Esempio 2.8.

$$a = 123.1256 \quad \text{e} \quad A = 123.12555551$$

```
>> a=123.1256; A=123.12555551;
>> errAss=abs(a-A)
errAss =
     4.449000000761316e-005
>> errRel=errAss/abs(a)
errRel =
     3.613383407480911e-007
```

Usando la rappresentazione normalizzata si ha:

$$\text{errAss} = 4.449\ldots10^{-5} = 0.4449\ldots10^{-4}$$

$$\text{errRel} = 3.613\ldots10^{-7} = 0.3613\ldots10^{-6}$$

Dunque A ha almeno 4 cifre decimali corrette e almeno 7 cifre significative corrette. In realtà ha proprio 4 cifre decimali corrette e 7 cifre significative corrette.

Esempio 2.9. $a = 0.11$ e $A = 0.11111112$

```
>> a=0.11; A=0.11111112;
>> errAss=abs(a-A)
errAss =
     1.111119999999993e-003
>> errRel=errAss/abs(a)
errRel =
     1.010109090909085e-002
```

Usando la rappresentazione normalizzata si ha:

$$\text{errAss} = 1.111\ldots10^{-3} = 0.1111\ldots10^{-2}$$

$$\text{errRel} = 1.010\ldots10^{-2} = 0.1010\ldots10^{-1}$$

Dunque A ha almeno 2 cifre decimali corrette e almeno 2 cifre significative corrette. In realtà ha proprio 2 cifre decimali corrette e 2 cifre significative corrette.

Esempio 2.10. $a = 1000.212324$ e $A = 1000.2123231$

```
>> a=1000.212324; A=1000.2123231;
>> errAss=abs(a-A)
errAss =
     8.999999181469320e-007
>> errRel=errAss/abs(a)
errRel =
     8.998088671290278e-010
```

Usando la rappresentazione normalizzata si ha:

$$\text{errAss} = 8.999\ldots10^{-7} = \frac{1}{2}0.1799\ldots10^{-5} < \frac{1}{2}10^{-5}$$

$$\text{errRel} = 8.998\ldots10^{-10} = \frac{1}{2}0.1799\ldots10^{-8} < \frac{1}{2}10^{-8}$$

Dunque A ha almeno 5 cifre decimali corrette e almeno 9 cifre significative corrette. In realtà ha proprio 5 cifre decimali corrette e 9 cifre significative corrette.

Variabile `eps` in MatLab In MatLab la variabile `eps` contiene il valore

$$\varepsilon = 2^{-52} = 2.22044604925031e - 016$$

```
>> eps
ans =
     2.22044604925031e-016
```

Verifichiamo che ε è il buco tra i numeri macchina 1.000000000000000 e 1.000000000000001 utilizzando la sua caratterizzazione e ricordando che il programma fornisce come risposta 1 per vero e 0 per falso.

```
>> 1+eps>1
ans =
     1
>> 1+eps/2>1
ans =
     0
```

Osserviamo, però, che, contrariamente a quanto ci aspettiamo:

```
>> 1+eps
ans =
     1.00000000000000e+000
```

Ciò è dovuto ad un errore di visualizzazione del numero $1 + eps$. Invece, rispettando le nostre aspettative:

```
>> 1+eps/2
ans =
     1
```

In MatLab esiste una routine che permette di calcolare il massimo errore relativo che si commette approssimando un qualunque numero macchina A.

```
>> x=10.23;
>> eps(x)
ans =
     1.77635683940025e-015
>> x+eps(x)>x
ans =
     1
>> x+eps(x)/2>x
ans =
     0
```

Ovviamente:

```
>> eps(1)
ans =
     2.22044604925031e-016
```

cioè `eps(1) = eps`.

2.2 Operazioni Macchina e Cancellazione Numerica

Terza conseguenza della rappresentazione dei numeri come parole di lunghezza fissa - Errori nei calcoli Non sempre una operazione tra due o più numeri macchina produce un risultato che è un numero macchina. Si può verificare una situazione di overflow o underflow. Inoltre la forma normalizzata del risultato potrebbe avere un numero di cifre decimali superiore alla precisione con cui si sta lavorando. Pertanto in un calcolatore è impossibile implementare esattamente le operazioni aritmetiche.

Dovremo accontentarci delle cosiddette operazioni macchina: a due numeri macchina viene associato un terzo numero macchina ottenuto arrotondando l'esatto risultato dell'operazione aritmetica.

Siano a e b due numeri reali e siano $A = fl(a)$ e $B = fl(b)$ i corrispondenti numeri macchina. Denotando con $\oplus, \ominus, \otimes, \oslash$ le operazioni macchina corrispondenti rispettivamente alle operazioni aritmetiche $+, -, \times, /$, si ha:

$$A \oplus B = fl(A + B) = (A + B)(1 + \delta_1)$$

$$A \ominus B = fl(A - B) = (A - B)(1 + \delta_2)$$
$$A \otimes B = fl(A \times B) = (A \times B)(1 + \delta_3)$$
$$A \oslash B = fl(A/B) = (A/B)(1 + \delta_4)$$

con

$$|\delta_i| \leq u \leq \varepsilon, \quad i \in \{1, 2, 3, 4\}$$

Dunque, quando si effettua una operazione macchina si commette un errore dell'ordine della precisione di macchina ε.

Osservazione I precedenti risultati valgono per lo standard IEEE 754 che utilizza i cosiddetti bit di guardia, cioè per eseguire le operazioni i numeri macchina vengono memorizzati in opportuni registri con dei bit aggiuntivi allo scopo di ridurre gli effetti degli errori di arrotondamento. In tale sistema si ha anche:

$$fl(\sqrt{A}) = \sqrt{A}(1 + \delta), \quad |\delta| \leq \varepsilon$$

Utilizzando i precedenti risultati è possibile stimare, in via teorica, gli errori in tutte le espressioni. Per esempio, se X, Y e Z sono numeri macchina si ha:

$$X \otimes (Y \oplus Z) = X \otimes fl(Y + Z) = fl(X \times fl(Y + Z))$$
$$= (X \times fl(Y + Z)) \times (1 + \delta_1)$$
$$= (X \times (Y + Z) \times (1 + \delta_2)) \times (1 + \delta_1)$$
$$\approx X \times (Y + Z) \times (1 + \delta_1 + \delta_2)$$

avendo trascurato il prodotto $\delta_1\delta_2$ perché piccolo. Dunque

$$X \otimes (Y \oplus Z) \approx X \times (Y + Z)(1 + \delta_3), \quad \delta_3 = \delta_1 + \delta_2, \quad |\delta_3| \leq \varepsilon$$

Domanda Per le operazioni macchina valgono ancora le note proprietà (commutativa, associativa, distributiva, etc) delle operazioni aritmetiche? La risposta non è sempre affermativa.

La proprietà commutativa permane:

$$A \oplus B = B \oplus A \quad \text{e} \quad A \otimes B = B \otimes A$$

mentre, in generale,

$$(A \oplus B) \oplus C \neq A \oplus (B \oplus C)$$
$$(A \otimes B) \otimes C \neq A \otimes (B \otimes C)$$
$$A \otimes (B \oplus C) \neq (A \otimes B) \oplus (A \otimes C)$$

Un'ulteriore relazione anomalia è:

$$A \oplus B = A, \quad \text{se } |B| \ll |A|$$

dunque, l'elemento neutro della somma non è unico.

Esempio 2.11. $N = 10, t = 16$

$$a = \exp(18)$$
$$b = \exp(-20)$$
$$A = fl(a) = 0.6565996913733051 \cdot 10^8$$
$$B = fl(b) = 0.2061153622438558 \cdot 10^{-8}$$
$$A \oplus B = fl(A + B) = 0.6565996913733051 \cdot 10^8 = A$$

Ciò accade perché, nell'eseguire la somma tra numeri macchina aventi esponenti diversi, il calcolatore esegue i seguenti passi:

1. individua il numero avente l'esponente $\overline{q}$ più grande;
2. memorizza tutti gli altri numeri utilizzando la loro rappresentazione con esponente $\overline{q}$;
3. esegue la somma.

Infatti, nel precedente esempio, si ha

$$B = fl(b) = 0.2061153622438558 \cdot 10^{-8}$$
$$= 0.00000000000000002061153622438558 \cdot 10^8$$

e, arrotondando a 16 cifre,

$$B = fl(b) = 0$$

Somma di più numeri Sempre a causa dell'errore di incolonnamento, se occorre sommare più numeri, tutti positivi e aventi ordini di grandezza diversi, per ottenere un risultato finale accurato conviene procedere in ordine ascendente (dal più piccolo al più grande), cosicché anche i valori più piccoli diano contributo alla somma.

Esempio 2.12. Dati i numeri in base 10:

$$\begin{array}{lll} 0.8999\text{e+}4 & 0.7889\text{e+}3 & 0.7767\text{e+}3 \\ 0.7555\text{e+}2 & 0.6266\text{e+}2 & 0.4298\text{e+}1 \\ 0.2581\text{e+}1 & 0.2653\text{e+}0 & 0.1580\text{e+}0 \end{array}$$

Supponendo di lavorare in aritmetica floating-point con $t = 4$, sommare i numeri in ordine ascendente e discendente e confrontare i risultati ottenuti con il valore esatto $s = 0.107101123e + 5$. Sommiamo prima in ordine discendente:

$$s1 = fl(0.8999e + 4 + 0.7889e + 3) = 0.9788e + 4$$
$$s1 = fl(s1 + 0.7767e + 3) = 0.1056e + 5$$
$$s1 = fl(s1 + 0.7555e + 2) = 0.1064e + 5$$
$$s1 = fl(s1 + 0.6266e + 2) = 0.1070e + 5$$
$$s1 = fl(s1 + 0.4298e + 1) = 0.1070e + 5$$
$$s1 = fl(s1 + 0.2653e + 0) = 0.1070e + 5$$
$$s1 = fl(s1 + 0.2581e + 1) = 0.1070e + 5$$
$$s1 = fl(s1 + 0.1580e + 0) = 0.1070e + 5$$

Osserviamo che solo i primi 5 numeri danno contributo alla somma.

Sommiamo ora in ordine ascendente:

$$s2 = fl(0.1580e + 0 + 0.2653e + 0) = 0.4233e + 0$$
$$s2 = fl(s2 + 0.2581e + 1) = 0.3004e + 1$$
$$s2 = fl(s2 + 0.4298e + 1) = 0.7302e + 1$$
$$s2 = fl(s2 + 0.6266e + 2) = 0.6996e + 2$$
$$s2 = fl(s2 + 0.7555e + 2) = 0.1455e + 3$$
$$s2 = fl(s2 + 0.7767e + 3) = 0.9222e + 3$$
$$s2 = fl(s2 + 0.7889e + 3) = 0.1711e + 4$$
$$s2 = fl(s2 + 0.8999e + 4) = 0.1071e + 5$$

Osserviamo che tutti i numeri danno contributo alla somma.

Calcolando gli errori relativi:

$$\frac{|s - s1|}{|s|} = 0.944 \cdot 10^{-3} \quad \text{(somma discendente)}$$

$$\frac{|s - s2|}{|s|} = 0.105 \cdot 10^{-4} \quad \text{(somma ascendente)}$$

deduciamo che, sommando in ordine discendente la somma viene calcolata con 3 cifre significative corrette, mentre, sommando in ordine ascendente la somma viene calcolata con 5 cifre significative corrette.

```
% Definiamo una "funzione macchina" che arrotonda a 4 cifre significative
% Ogni volta che facciamo un calcolo, dobbiamo passare di qui.
fl = @(x) round(x, 4, 'significant');

% Valore esatto della somma
su = 0.107101123e+5;

% Dati di input
a1 = 0.8999e+4;
b1 = 0.7889e+3;
c1 = 0.7767e+3;
a2 = 0.7555e+2;
b2 = 0.6266e+2;
c2 = 0.4298e+1;
a3 = 0.2581e+1;
```

```
b3 = 0.2653e+0;
c3 = 0.1580e+0;

% --- CASO 1: Somma dai Grandi ai Piccoli (Discendente) ---
S = a1;            % Primo numero
S = fl(S + b1);    % Sommo e taglio subito a 4 cifre!
S = fl(S + c1);    % Uso il risultato tagliato per la somma successiva...
S = fl(S + a2);
S = fl(S + b2);
S = fl(S + c2);
S = fl(S + a3);
S = fl(S + b3);
S = fl(S + c3);

% --- CASO 2: Somma dai Piccoli ai Grandi (Ascendente) ---
s = c3;            % Ultimo numero
s = fl(s + b3);    % Sommo e taglio...
s = fl(s + a3);
s = fl(s + c2);
s = fl(s + b2);
s = fl(s + a2);
s = fl(s + c1);
s = fl(s + b1);
s = fl(s + a1);

% --- Risultati ---
fprintf('Risultato somma discendente: %.4e\n', S);
fprintf('Risultato somma ascendente: %.4e\n', s);

% Errori relativi tra le due simulazioni di somma
% Somma discendente
diff = errRel(su,S);
fprintf('Errore relativo somma discendente (t=4): %.4e\n', diff);
% Somma ascendente
diff = errRel(su,s);
fprintf('Errore relativo somma ascendente (t=4): %.4e\n', diff);
```

Applicazione in MATLAB

```
Risultato somma discendente: 1.0700e+04
Risultato somma ascendente: 1.0710e+04
Errore relativo somma discendente (t=4): 9.4418e-04
Errore relativo somma ascendente (t=4): 1.0485e-05
```

Cancellazione Numerica È un fenomeno che si verifica durante l'operazione di sottrazione tra due numeri quasi uguali. Siano x_1 e x_2 due numeri reali. Se $x = x_1 - x_2$ è molto piccolo, l'errore relativo

$$\delta x = \left| \frac{fl(x_1 - x_2) - x}{x} \right|$$

può essere molto grande e ciò produce una perdita di cifre significative nel calcolo di $fl(x_1 - x_2)$.

Osservazione È sempre preferibile evitare la sottrazione tra numeri macchina quasi uguali utilizzando, laddove possibile, delle espressioni alternative. Per esempio:

1. $$\sqrt{x+\delta} - \sqrt{x} = \frac{\delta}{\sqrt{x+\delta} + \sqrt{x}}$$

2. $$\cos(x+\delta) - \cos(x) = -2\sin\left(\frac{\delta}{2}\right)\sin\left(x + \frac{\delta}{2}\right)$$

In generale, se si presenta il caso di dover computare

$$f(x+\delta) - f(x),$$

con $|\delta| \ll |x|$, può convenire considerare l'espansione di Taylor

$$f(x+\delta) - f(x) = f'(x)\delta + f(x)\frac{\delta}{2} + \dots$$

Un problema di cancellazione si può presentare anche nella formula risolutiva dell'equazione di secondo grado $ax^2 + bx + c = 0$:

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

per cui conviene usare la formula alternativa:

$$x_1 = \frac{-b - \text{sign}(b)\sqrt{b^2 - 4ac}}{2a}$$

$$x_2 = \frac{c}{ax_1}$$

Esempi in MatLab

Esempio 2.13. Verifichiamo che non sempre una operazione tra numeri macchina produce un numero macchina.

```
a = 1.000345566e+150;
b = 0.15646464e+200;
disp(a*b)

a = 1.000345566e-25;
b = 1.56464646e-300;
disp(a*b)
```

```
   Inf

     0
```

Da quest'ultimo esempio si evince anche che il prodotto tra due numeri macchina può essere nullo anche se entrambi i numeri sono non nulli, cioè non vale la legge di annullamento del prodotto.

Esempio 2.14. Siano $a = 0.23371258 \cdot 10^{-4}$, $b = 0.33678429 \cdot 10^{2}$, $c = -0.33677811 \cdot 10^{2}$, verificare che $(a \oplus b) \oplus c \neq a \oplus (b \oplus c)$.

```
format long e;

% Dati
a = 0.23371258e-4;
b = 0.33678429e+2;
c = -0.33677811e+2;

% Output: [a, b, c]
fprintf('a: %.15e\nb: %.15e\nc: %.15e\n', a, b, c);

% Calcoli
s1 = (a + b) + c;
s2 = a + (b + c);

% Output: [s1, s2, differenza]
fprintf('(a + b) + c: %.15e\na + (b + c): %.15e\nDifferenza tra le due somme: %.15e\n', s1, s2
    , s1-s2);
```

```
a: 2.337125800000000e-05
b: 3.367842900000000e+01
c: -3.367781100000000e+01
(a + b) + c: 6.413712580055630e-04
a + (b + c): 6.413712580028940e-04
Differenza tra le due somme: 2.669088908224815e-15
```

Esempio 2.15. Siano $a = 0.1234567$, $b = 6666.325$ e $c = -6666.325$, verificare che $(a \oplus b) \oplus c \neq a \oplus (b \oplus c)$.

```
format long e;

% Dati
a = 0.1234567;
b = 6666.325;
```

```
c = -6666.325;

% Output: [a, b, c]
fprintf('a: %.15e\nb: %.15e\nc: %.15e\n', a, b, c);

% Calcoli
s1 = (a + b) + c;
s2 = a + (b + c);

% Output: [s1, s2, differenza]
fprintf('(a + b) + c: %.15e\na + (b + c): %.15e\nDifferenza tra le due somme: %.15e\n', s1, s2
    , s1-s2);
```

```
a: 1.234567000000000e-01
b: 6.666325000000000e+03
c: -6.666325000000000e+03
(a + b) + c: 1.234567000001334e-01
a + (b + c): 1.234567000000000e-01
Differenza tra le due somme: 1.334210519843282e-13
```

Esempio 2.16. Siano $a = 0.8e{+}9$, $b = 0.500009e{+}4$ e $c = -0.500008e{+}4$, verificare che $a \otimes (b \oplus c) \neq a \otimes b \oplus a \otimes c$.

```
format long e;

% Dati
a = 0.8e+9
b = 0.500009e+4
c = -0.500008e+4

% Output: [a, b, c]
fprintf('a: %.15e\nb: %.15e\nc: %.15e\n', a, b, c);

% Calcoli
s1 = a * (b + c);
s2 = a * b + a * c;

% Output: [s1, s2, differenza]
fprintf('a * (b + c): %.15e\na * b + a * c: %.15e\nDifferenza tra le due operazioni: %.15e\n',
     s1, s2, s1-s2);
```

```
a: 8.000000000000000e+08
b: 5.000090000000000e+03
c: -5.000080000000000e+03
a * (b + c): 8.000000000174623e+06
a * b + a * c: 8.000000000000000e+06
Differenza tra le due operazioni: 1.746229827404022e-04
```

Infatti:

```
a * b: 4.000072000000000e+12
a * c: -4.000064000000000e+12
a * b + a * c: 8.000000000000000e+06
```

Esempio 2.17. Siano $a = 7.45700244034177e-001$, $b = 3.77400852642836e-001$ e $c = 7.62332001118890e-001$, verificare che $(a \otimes b) \otimes c \neq a \otimes (b \otimes c)$.

```
format long e;

% Dati
a = 7.45700244034177e-001;
b = 3.77400852642836e-001;
c = 7.62332001118890e-001;

% Output: [a, b, c]
fprintf('a: %.15e\nb: %.15e\nc: %.15e\n', a, b, c);

% Calcoli
```

```
p1 = (a * b) * c;
p2 = a * (b * c);

% Output: [s1, s2, differenza]
fprintf('(a * b) * c: %.15e\na * (b * c): %.15e\nDifferenza tra i due prodotti: %.15e\n', p1,
    p2, p1-p2);
```

```
a: 7.457002440341770e-01
b: 3.774008526428360e-01
c: 7.623320011188900e-01
(a * b) * c: 2.145415002111401e-01
a * (b * c): 2.145415002111401e-01
Differenza tra i due prodotti: 2.775557561562891e-17
```

Osserviamo che i valori $p1$ e $p2$ risultano essere uguali anche se i corrispondenti numeri macchina non lo sono (la loro differenza non è nulla). Ciò accade per un errore di visualizzazione o di conversione dal binario al decimale.

Esempio 2.18. Verifichiamo che dati $c = 0.164646415647 \cdot 10^{+9}$ e $d = 0.38731646 \cdot 10^{-7}$ si ha $c \oplus d = fl(c+d) = fl(c)$.

```
>> c = 0.164646415647e+9;
>> d = 0.38731646e-7;
>> c+d
ans =
     1.646464156470000e+008
```

Esempio 2.19. Verificare che l'uguaglianza

$$\sqrt{x+\delta} - \sqrt{x} = \frac{\delta}{\sqrt{x+\delta} + \sqrt{x}}$$

non è vera se si lavora in aritmetica finita. Supponiamo $x = 4$ e $\delta = 0.002$.

```
>> a = sqrt(4.002)
a =
     2.000499937515620e+000
>> b = sqrt(4)
b =
     2
>> a-b
ans =
     4.999375156202746e-004
>> 0.002/(a+b)
ans =
     4.999375156201189e-004
```

Con la prima formula si perdono 4 cifre significative corrette.

Esempio 2.20. Verificare che l'uguaglianza

$$\cos(x+\delta) - \cos(x) = -2\sin\left(\frac{\delta}{2}\right)\sin\left(x + \frac{\delta}{2}\right)$$

non è vera se si lavora in aritmetica finita. Supponiamo $x = \pi$ e $\delta = 0.0001$.

```
>> cos(pi+0.0001)-cos(pi)
ans =
     4.99999969612645e-009
>> -2*sin(0.0001/2)*sin(pi+0.0001/2)
ans =
     4.999999995809435e-009
```

Con la prima formula si perdono 8 cifre significative corrette.

Esempio 2.21. Data l'equazione di secondo grado

$$x^2 - 0.4002x + 0.8e - 4 = 0,$$

calcolare le radici con le formule:

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a},$$

e

$$x_1 = \frac{-b - \text{sign}(b)\sqrt{b^2 - 4ac}}{2a}, \quad x_2 = \frac{c}{ax_1}$$

e confrontare i risultati ottenuti con i valori esatti $x_1 = 0.4$ e $x_2 = 0.0002$. Osserviamo prima di tutto che se calcoliamo il valore del polinomio di secondo grado nelle sue radici non si ottiene 0 ma uno degli infiniti zeri macchina.

```
>> x = 0.4;
>> x^2 - 0.4002*x + 0.8e-4
ans =
     3.102173466024150e-017
>> x = 0.0002;
>> x^2 - 0.4002*x + 0.8e-4
ans =
     1.355252715606881e-020
```

Dunque l'equazione è numericamente verificata a meno di un errore dell'ordine dell'epsilon-macchina.

```
>> a = 1; b = -0.4002; c = 0.8e-4;
>> x1 = (-b + sqrt(b^2 - 4*a*c))/(2*a)
x1 =
     4.000000000000000e-001
>> x2 = (-b - sqrt(b^2 - 4*a*c))/(2*a)
x2 =
     1.999999999999780e-004
>> x22 = c/x1
x22 =
     2.000000000000000e-004
```

È evidente che con la prima formula si perdono 3 cifre significative corrette.

Esempio 2.22. Valutare le espressioni analiticamente equivalenti

$$y_1 = \frac{1 - \cos(t)}{t^2} \quad \text{e} \quad y_2 = \frac{1}{2}\frac{\sin^2(t/2)}{(t/2)^2}$$

nel punto $t = 1.2e - 5$.

```
>> t = 1.2e-5;
>> y1 = (1-cos(t))/t^2
y1 =
     4.999997329749008e-001
>> y2 = 0.5*(sin(t/2)/(t/2))^2
y2 =
     4.999999999940000e-001
```

La diversità dei risultati ottenuti è dovuta al fenomeno della cancellazione numerica verificatosi nel calcolo di $1-\cos(t)$, perché, con la scelta fatta di t, 1 e $\cos(t)$ sono quasi uguali. Poiché l'espressione y_2 è definita mediante operazioni che non introducono alcuna perdita di precisione, possiamo ritenere che il valore $y2$ abbia 15 cifre significative corrette. Pertanto, confrontando $y1$ con $y2$ deduciamo che con l'espressione y_1 si ottengono solo 6 cifre significative corrette.

Esempio 2.23. Valutare le espressioni, analiticamente equivalenti,

$$y_1 = (1-x)^6 \quad \text{e} \quad y_2 = x^6 - 6x^5 + 15x^4 - 20x^3 + 15x^2 - 6x + 1$$

in 100 punti equidistanti nell'intervallo $[1-\delta, 1+\delta]$ per $\delta = 0.1, 0.01, 0.005, 0.0025$. Rappresentare graficamente y_1 e y_2 per ciascun valore di δ assegnato.

```
>> delta = 0.1;
>> x = linspace(1-delta, 1+delta);
>> y1 = (1-x).^6;
>> y2 = x.^6 - 6*x.^5 + 15*x.^4 - 20*x.^3 + 15*x.^2 - 6*x + 1;
>> figure(1)
>> plot(x, y1, x, y2, 'r')
```

Si ottiene il grafico relativo a $\delta = 0.1$.

```
>> delta = 0.01;
>> x = linspace(1-delta, 1+delta);
>> y1 = (1-x).^6;
>> y2 = x.^6 - 6*x.^5 + 15*x.^4 - 20*x.^3 + 15*x.^2 - 6*x + 1;
>> figure(2)
>> plot(x, y1, x, y2, 'r')
```

Si ottiene il grafico relativo a $\delta = 0.01$.

```
>> delta = 0.005;
>> x = linspace(1-delta, 1+delta);
>> y1 = (1-x).^6;
>> y2 = x.^6 - 6*x.^5 + 15*x.^4 - 20*x.^3 + 15*x.^2 - 6*x + 1;
>> figure(3)
>> plot(x, y1, x, y2, 'r')
```

Si ottiene il grafico relativo a $\delta = 0.005$.

```
>> delta = 0.0025;
>> x = linspace(1-delta, 1+delta);
>> y1 = (1-x).^6;
>> y2 = x.^6 - 6*x.^5 + 15*x.^4 - 20*x.^3 + 15*x.^2 - 6*x + 1;
>> figure(4)
>> plot(x, y1, x, y2, 'r')
```

Si ottiene il grafico relativo a $\delta = 0.0025$.

Osservando i precedenti grafici, deduciamo che le rappresentazioni grafiche ottenute utilizzando le espressioni y_1 e y_2 si discostano l'una dall'altra sempre di più al diminuire di δ. Infatti, mentre y_1 rimane fedele al reale andamento del polinomio, y_2 ha un andamento sempre più oscillante (come se avesse tanti zeri!). Tali oscillazioni sono dovute al fenomeno della cancellazione numerica che si verifica quando in y_2 si esegue la somma tra

$$x^6 - 6x^5 + 15x^4 - 20x^3 + 15x^2 - 6x \quad \text{e} \quad 1$$

per valori molto vicini a 1 ($x^6 - 6x^5 + 15x^4 - 20x^3 + 15x^2 - 6x < 0$ per ogni $x \in [-1, 1]$). Osserviamo che il suddetto fenomeno, seppure non evidente graficamente, si verifica anche quando in y_1 si esegue la sottrazione tra 1 e x per valori di x molto vicini a 1. Tuttavia, essendo le quantità

$$(1-x)^6 \ll (1-x)$$

le quantità

$$x^6 - 6x^5 + 15x^4 - 20x^3 + 15x^2 - 6x \quad \text{e} \quad 1$$

hanno più cifre in comune delle quantità 1 e x. Ne consegue che nella valutazione di y_2 si ha una perdita di precisione maggiore rispetto a y_1.

2.3 Propagazione degli Errori Introdotti nei Dati Iniziali

Condizionamento di un problema L'esempio della cancellazione numerica introduce un concetto molto rilevante del Calcolo Scientifico: il condizionamento di un problema. Nella risoluzione di un problema matematico è in molti casi utile avere una misura di quanto i risultati siano sensibili a piccoli cambiamenti dei dati iniziali. In generale se a piccoli cambiamenti nei dati iniziali corrispondono grandi cambiamenti nei dati finali diremo che il problema è **mal condizionato**. In caso contrario diremo che è **ben condizionato**. La quantità responsabile dell'amplificazione degli errori nei dati iniziali si chiama **indice di condizionamento**. Il condizionamento è una caratteristica propria del problema matematico e non ha alcun legame né con gli errori di arrotondamento delle operazioni macchina, né con gli algoritmi eventualmente scelti per risolverlo. Determinare l'indice di condizionamento di un problema non è sempre facile. Consideriamo il caso semplice del problema numerico della somma. Siano x, y e z tre numeri reali con

$$z = x + y$$

Supponiamo di perturbare x con Δx e y con Δy e sia

$$\overline{z} = (x + \Delta x) + (y + \Delta y)$$

Calcoliamo l'errore relativo che commettiamo approssimando z con $\overline{z}$:

$$\begin{aligned}
\delta z = \frac{|z - \overline{z}|}{|z|} &= \frac{|(x+y) - (x + \Delta x + y + \Delta y)|}{|x+y|} = \frac{|-\Delta x - \Delta y|}{|x+y|} = \frac{|\Delta x + \Delta y|}{|x+y|} \\
&\leq \frac{|\Delta x| + |\Delta y|}{|x+y|} = \frac{1}{|x+y|}\left(|\Delta x| \cdot \frac{|x|}{|x|} + |\Delta y| \cdot \frac{|y|}{|y|}\right) \\
&= \frac{1}{|x+y|}\left(|x| \underbrace{\frac{|\Delta x|}{|x|}}_{\delta x} + |y| \underbrace{\frac{|\Delta y|}{|y|}}_{\delta y}\right) = \frac{1}{|x+y|}(|x|\delta x + |y|\delta y) \\
&\leq \frac{1}{|x+y|}\left(\max\{|x|,|y|\}\delta x + \max\{|x|,|y|\}\delta y\right) = \frac{\max\{|x|,|y|\}}{|x+y|}(\delta x + \delta y)
\end{aligned}$$

La quantità

$$\frac{\max\{|x|,|y|\}}{|x+y|}$$

rappresenta l'**indice di condizionamento**. Ciò significa che se facciamo la somma tra due numeri molto vicini e di segno opposto, anche se gli errori relativi δx e δy sono piccoli, il valore $\overline{z}$ approssimerà z con un errore relativo grande, cioè l'operazione somma diventa mal condizionata e si verifica il fenomeno della Cancellazione Numerica. In generale, denotato con $f(x)$ un generico problema numerico avente x come dato iniziale, si cerca di determinare una relazione del tipo

$$\frac{|f(x) - f(\overline{x})|}{|f(x)|} \leq K \frac{|\Delta x|}{|x|}$$

dove K rappresenta l'indice di condizionamento del problema f. Quanto più è piccolo K, tanto più il problema è ben condizionato.

Esempio 2.24. Consideriamo il sistema lineare di due equazioni in due incognite

$$\begin{cases} x_1 - x_2 = 1 \\ x_1 - 1.00001 x_2 = 0 \end{cases}$$

le cui soluzioni esatte sono

$$x_1 = 100001, \quad x_2 = 100000$$

Perturbiamo di una piccola quantità pari a $0.2 \cdot 10^{-4}$ il coefficiente dell'incognita x_2 nella seconda equazione. Il suo nuovo coefficiente sarà $-1.00001 + 0.2 \cdot 10^{-4} = -0.99999$. Il nuovo sistema diventa

$$\begin{cases} y_1 - y_2 = 1 \\ y_1 - 0.99999 y_2 = 0 \end{cases}$$

e le sue soluzioni sono

$$y_1 = -99999, \quad y_2 = -100000$$

Osservazioni

- Provando a risolvere i precedenti sistemi con il metodo di Cramer o il metodo di sostituzione si ottengono soluzioni diverse da quelle indicate. Per verificare che le soluzioni indicate sono corrette, basta sostituirle in ciascuna riga del sistema.
- Poiché le soluzioni sono state ricavate lavorando in aritmetica infinita, l'unico motivo per il quale, in seguito alla piccola perturbazione, si sono ottenuti risultati tanto diversi è che la risoluzione del sistema di partenza è un problema mal condizionato.

Stabilità di un algoritmo

Definizione 2.7. Un algoritmo, per un problema numerico, è una sequenza finita di operazioni non ambigue che trasforma i dati di input e fornisce uno o più valori di output. Diremo che un algoritmo è **instabile** se amplifica gli errori di arrotondamento durante la computazione della soluzione di un problema ben condizionato. Dunque il concetto di stabilità di un algoritmo è strettamente connesso con la precisione di calcolo finita e con le operazioni in cui si risolve l'algoritmo stesso.

Esempio 2.25. Consideriamo ora il seguente sistema lineare di due equazioni in due incognite

$$\begin{cases} \alpha x_1 + x_2 = 1 + \alpha \\ x_1 = 1 \end{cases}$$

la cui soluzione esatta è

$$x_1 = x_2 = 1$$

Si dimostra che il problema della sua risoluzione è ben condizionato. Nonostante l'immediatezza della soluzione, seguiamo lo schema di calcolo del metodo di eliminazione di Gauss. Moltiplicando ambo i membri della prima equazione per $-\frac{1}{\alpha}$, otteniamo

$$-\frac{1}{\alpha}(\alpha x_1 + x_2) = -\frac{1}{\alpha}(1 + \alpha),$$

cioè

$$-x_1 - \frac{1}{\alpha}x_2 = -\frac{1+\alpha}{\alpha}$$

Sommando la precedente equazione alla seconda equazione ($x_1 = 1$), otteniamo

$$-x_1 - \frac{1}{\alpha}x_2 + x_1 = 1 - \frac{1+\alpha}{\alpha}$$

$$-\frac{1}{\alpha}x_2 = 1 - \frac{1+\alpha}{\alpha}$$

Rimpiazzando la seconda equazione con quella ottenuta, otteniamo il seguente sistema equivalente al precedente:

$$\begin{cases} \alpha x_1 + x_2 = 1 + \alpha \\ -\frac{1}{\alpha}x_2 = 1 - \frac{1+\alpha}{\alpha} \end{cases}.$$

Computiamo i coefficienti del sistema per $\alpha = 0.5 \times 10^{-11}$. Si ha

$$0.5 \times 10^{-11} x_1 + x_2 = 1 + 0.5 \times 10^{-11}$$

$$-2 \times 10^{11} x_2 = 1 - \frac{1 + 0.5 \times 10^{-11}}{0.5 \times 10^{-11}}$$

e, dunque, eseguendo la computazione in base alla priorità degli operatori aritmetici:

$$\begin{cases} x_2 = \frac{1}{-2\times 10^{11}} \left(1 - \frac{(1+0.5\times 10^{-11})}{0.5\times 10^{-11}}\right) = 1 \\ x_1 = \frac{(1+0.5\times 10^{-11}-1)}{0.5\times 10^{-11}} = 1.000000082740371 \end{cases}$$

Scambiamo ora le equazioni tra di loro (pivoting):

$$\begin{cases} x_1 = 1 \\ \alpha x_1 + x_2 = 1 + \alpha \end{cases}$$

e ripetiamo la procedura di eliminazione dell'incognita x_1 dalla seconda equazione. Moltiplicando ambo i membri della prima equazione per $-\alpha$, otteniamo

$$-\alpha x_1 = -\alpha$$

Sommando la precedente equazione alla seconda equazione, otteniamo

$$-\alpha x_1 + \alpha x_1 + x_2 = -\alpha + 1 + \alpha$$

$$x_2 = 1$$

Dunque

$$\begin{cases} x_1 = 1 \\ x_2 = 1 \end{cases} .$$

Questa procedura che, sotto opportune condizioni, prevede lo scambio delle righe di un sistema lineare prende il nome di **pivoting**. La causa di questo bizzarro comportamento è da ricercarsi nella sequenza di operazioni che ci conducono dai dati iniziali ai risultati, ovvero nell'algoritmo. Il metodo di eliminazione di Gauss in generale non è stabile, ma se viene combinato con il pivoting è sempre stabile.

Abbiamo detto che il problema della somma tra due numeri molto vicini è mal condizionato e genera il fenomeno della Cancellazione Numerica. Per questo tipo di problema è ancora più importante scegliere un algoritmo stabile.

Se consideriamo il problema della differenza tra due radicali, abbiamo visto che

$$\sqrt{x+\delta} - \sqrt{x} = \frac{\delta}{\sqrt{x+\delta} + \sqrt{x}}$$

Come abbiamo già verificato con dei tests numerici, l'algoritmo che implementa la formula al primo membro è instabile mentre quello che implementa la formula al secondo membro è sempre stabile.

Analogo discorso si può fare per:

- il calcolo della differenza tra due coseni con le formule equivalenti:

  $$\cos(x+\delta) - \cos(x)$$

  e

  $$-2\sin\left(\frac{\delta}{2}\right)\sin\left(x+\frac{\delta}{2}\right)$$

- il calcolo delle radici dell'equazione $ax^2 + bx + c = 0$ con le formule equivalenti:

  $$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

  e

  $$x_1 = \frac{-b - \text{sign}(b)\sqrt{b^2 - 4ac}}{2a}, \quad x_2 = \frac{c}{ax_1}$$

Abbiamo già verificato con dei tests numerici, che per entrambi l'algoritmo che implementa la prima formula è instabile mentre quello che implementa la seconda è sempre stabile.

Dunque il controllo degli errori passa essenzialmente per due fasi:

1. studio del condizionamento del problema matematico;
2. uso di un algoritmo stabile per la soluzione del problema.

Conclusioni

- Qualsiasi algoritmo applicato ad un problema mal condizionato genererà amplificazione degli errori di rappresentazione dei dati.
- Si avrà propagazione degli errori di arrotondamento applicando un algoritmo instabile ad un problema ben condizionato.
- Si raggiunge la massima precisione di calcolo quando un algoritmo stabile viene applicato ad un problema ben condizionato.
- A volte può accadere che un problema sia ben condizionato per certi dati iniziali e mal condizionato per altri (vedi l'operazione di somma).

L'obiettivo dei matematici che si occupano del Calcolo Scientifico è quello di risolvere problemi numerici di cui hanno studiato a priori il condizionamento con algoritmi che siano il più possibile stabili.

Esercizi in MatLab

Esempio 2.26. Risolviamo il sistema

$$\begin{cases} x_1 - x_2 = 1 \\ x_1 - 1.00001x_2 = 0 \end{cases}$$

Definiamo la matrice del sistema:

```
>> A = [1, -1; 1, -1.00001]
A =
   1.000000000000000e+000  -1.000000000000000e+000
   1.000000000000000e+000  -1.000010000000000e+000
```

Definiamo il vettore dei termini noti:

```
>> b = [1; 0]
b =
     1
     0
```

Verifichiamo che il vettore $[100001, 100000]'$ è la soluzione esatta del sistema:

```
>> A * [100001; 100000]
ans =
     1
     0
```

Risolviamo il sistema $Ax = b$ con la function \del MatLab:

```
>> x = A\b
x =
   1.00009999993449e+005
   9.99999999934487e+004
```

Calcoliamo gli errori relativi:

```
>> err1 = abs(100001 - 1.000009999993449e+005)/abs(100001)
err1 =
   6.550915688349306e-12
>> err2 = abs(100000 - 9.999999999934487e+004)/abs(100000)
err2 =
   6.551272235810757e-12
```

Si ha

$$\text{err1} = 0.655\ldots 10^{-11} < \frac{1}{2}10^{-10}$$

$$\text{err2} = 0.655\ldots 10^{-11} < \frac{1}{2}10^{-10}$$

cioè entrambe le soluzioni sono state calcolate con solo 11 cifre significative corrette. La function \del MatLab risolve i sistemi lineari con un algoritmo stabile che implementa il metodo di eliminazione di Gauss con pivoting. Pertanto la perdita di cifre significative è dovuta al mal condizionamento della matrice del sistema. Se A denota la matrice dei coefficienti di un sistema lineare il comando `cond(A,inf)` del MatLab permette di calcolare l'indice di condizionamento del problema della risoluzione del sistema.

```
>> C = cond(A,inf)
C =
   4.000040000073795e+005
```

Poiché le perturbazioni che induciamo sui dati iniziali A e b del problema sono dell'ordine dell'epsilon-macchina, moltiplicando l'indice di condizionamento ottenuto per la variabile `eps` del MatLab, otteniamo una stima dell'errore con cui calcoleremo la soluzione x.

```
>> deltax = C*eps
deltax =
   8.881873015007080e-011
```

cioè

$$\text{deltax} = 0.888\ldots 10^{-10} < \frac{1}{2}10^{-9}$$

La precedente stima ci dice che nel risolvere un sistema lineare avente A come matrice dei coefficienti con un algoritmo stabile possiamo ottenere almeno 10 cifre significative corrette. Tale stima è confermata dai tests numerici, infatti abbiamo 11 cifre.

Esempio 2.27. Risolviamo il sistema

$$\begin{cases} y_1 - y_2 = 1 \\ y_1 - 0.99999y_2 = 0 \end{cases}$$

Definiamo la matrice del sistema:

```
>> A = [1, -1; 1, -0.99999]
A =
   1.000000000000000e+000  -1.000000000000000e+000
   1.000000000000000e+000  -9.999900000000000e-001
```

Definiamo il vettore dei termini noti:

```
>> b = [1; 0]
```

Verifichiamo che il vettore $[-99999, -100000]'$ è la soluzione esatta del sistema:

```
>> A * [-99999; -100000]
ans =
     1
     0
```

Risolviamo il sistema $Ax = b$ con la function \del MatLab:

```
>> x = A\b
x =
  -9.999900000045510e+004
  -1.000000000004551e+005
```

Calcoliamo gli errori relativi:

```
>> err1 = abs(-99999 + 9.999900000045510e+004)/abs(-99999)
err1 =
   4.551011478634235e-012
>> err2 = abs(-100000 + 1.000000000004551e+005)/abs(-100000)
err2 =
   4.550965968519449e-12
```

Si ha

$$\text{err1} < \frac{1}{2}10^{-11}, \quad \text{err2} < \frac{1}{2}10^{-11}$$

cioè entrambe le soluzioni sono state calcolate con solo 12 cifre significative corrette.

Calcoliamo l'indice di condizionamento del problema.

```
>> C = cond(A,inf)
C =
   4.000000000018204e+005
```

Stima dell'errore:

```
>> deltax = C*eps
deltax =
   8.881784197041673e-011
```

cioè deltax $< \frac{1}{2}10^{-9}$. Dunque nel risolvere un sistema lineare avente A come matrice dei coefficienti con un algoritmo stabile possiamo ottenere almeno 10 cifre significative corrette. Ne abbiamo ottenute 12.

Esempio 2.28. Risolviamo il seguente sistema lineare con $\alpha = 0.5 \times 10^{-11}$

$$\begin{cases} \alpha x_1 + x_2 = 1 + \alpha \\ x_1 = 1 \end{cases}$$

La sua soluzione esatta è $x_1 = x_2 = 1$ per ogni scelta di α. Definiamo la matrice del sistema e il vettore dei termini noti:

```
>> al = 0.5e-11;
>> A = [al, 1; 1, 0];
>> b = [1+al; 1];
```

Calcoliamo l'indice di condizionamento del problema:

```
>> C = cond(A,inf)
C =
   1.000000000010000e+000
>> deltax = C*eps
deltax =
   2.220446049272518e-016
```

Da cui deduciamo che il problema della risoluzione del sistema è ben condizionato e se lo risolviamo con un algoritmo stabile possiamo ottenere una soluzione con 16 cifre significative corrette.

Abbiamo visto che se lo risolviamo applicando il metodo di Gauss (instabile in questo caso) otteniamo:

```
>> x1 = (1 + 0.5e-11 - 1)/0.5e-11
x1 =
   1.000000082740371e+000
>> x2 = (1 - (1 + 0.5e-11)/0.5e-11)/(-2.e+11)
x2 =
     1
```

Dunque, se un problema ben condizionato viene risolto con un algoritmo instabile si ottengono soluzioni poco accurate.

Se, invece, lo risolviamo con la function \del MatLab che implementa il metodo di eliminazione di Gauss con pivoting (stabile), otteniamo:

```
>> A\b
ans =
     1
     1
```

Dunque, se un problema ben condizionato viene risolto con un algoritmo stabile si ottengono soluzioni molto accurate.

Esempio 2.29. Siano $a = 1.4e154$ e $b = 1.3e154$. Calcolare le espressioni equivalenti

$$(a^2 - b^2) \quad \text{e} \quad (a-b)(a+b)$$

e confrontare i risultati.

```
>> a = 1.4e154; b = 1.3e154;
>> (a^2 - b^2)
ans =
   Inf
>> (a-b)*(a+b)
ans =
   2.700000000000000e+307
```

Si può vedere che con il primo algoritmo si ottiene un risultato che non è un numero macchina (overflow). Ciò si verifica perché

```
>> a^2
ans =
   Inf
```

Possiamo concludere che il secondo algoritmo è più stabile del primo.

3 Risoluzione di un Sistema Lineare Quadrato

3.1 Il Problema della Risoluione di un Sistema Lineare

Generalità sui Sistemi Lineari Molti problemi dell'ingegneria, della fisica, della chimica, dell'informatica e dell'economia, si modellizzano mediante equazioni lineari, cioè equazioni in cui le incognite appaiono con esponente uguale ad uno e sono legate tra loro da relazioni lineari (prodotto per scalari e somme algebriche). Un sistema di n equazioni lineari in n incognite è definito come segue:

$$\sum_{j=1}^{n} a_{i,j} x_j = b_i \quad i \in \{1, \dots, n\}$$

ovvero:

$$\begin{cases} a_{1,1}x_1 + a_{1,2}x_2 + \cdots + a_{1,n}x_n = b_1 \\ a_{2,1}x_1 + a_{2,2}x_2 + \cdots + a_{2,n}x_n = b_2 \\ \vdots \\ a_{n,1}x_1 + a_{n,2}x_2 + \cdots + a_{n,n}x_n = b_n \end{cases}$$

Assumeremo che tutte le quantità in gioco siano numeri reali.

Forma Matriciale Ponendo A come la matrice dei coefficienti, b come il vettore dei termini noti e x come il vettore delle soluzioni:

$$A = \begin{pmatrix} a_{1,1} & \dots & a_{1,n} \\ \vdots & \ddots & \vdots \\ a_{n,1} & \dots & a_{n,n} \end{pmatrix}, \quad b = \begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix}, \quad x = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}$$

il sistema si può riscrivere come:

$$Ax = b$$

dove $A \in \mathbb{R}^{n \times n}$, $b \in \mathbb{R}^n$ e $x \in \mathbb{R}^n$.

Esistenza e Unicità della Soluzione È noto che il sistema ammette un'unica soluzione se e solo se la matrice A è non singolare, o equivalentemente se $\det(A) \neq 0$. In questo caso, denotata con A^{-1} la matrice inversa di A, si ha:

$$x = A^{-1}b$$

La matrice inversa è infatti l'unica matrice per la quale $A^{-1}A = AA^{-1} = I$, dove I è la matrice identità.

Il Metodo di Cramer La teoria dell'Algebra Lineare propone come metodo risolutivo il metodo di Cramer:

$$x_i = \frac{\det(A_i)}{\det(A)}, \quad i \in \{1, \dots, n\}$$

dove A_i denota la matrice ottenuta da A sostituendo la colonna i-esima con il vettore b. I determinanti possono essere calcolati utilizzando la regola di Laplace:

$$\det(A) = \sum_{j=1}^{n} (-1)^{j+1} a_{1,j} \det(A_{1,j})$$

dove $A_{1,j}$ è la matrice di ordine $n-1$ ottenuta da A eliminando la prima riga e la j-esima colonna.

Costo Computazionale e Necessità dei Metodi Numerici Sfortunatamente il metodo di Cramer ha un costo computazionale molto elevato, pari a circa $[(n+1)(n-1)n]!$ operazioni elementari (flops). Ciò si traduce in tempi effettivi per la computazione inaccettabili. Supponendo per esempio di poter effettuare 10^6 flops al secondo, per risolvere un sistema di 20 equazioni occorrono circa $3 \cdot 10^7$ anni (300.000 secoli). Anche i metodi classici per il calcolo dell'inversa A^{-1} hanno un costo computazionale di tipo fattoriale. Poiché nei problemi reali i sistemi possono essere costituiti da centinaia o migliaia di equazioni, occorrono metodi numerici efficienti che tengano conto della dimensione e delle proprietà della matrice.

Matrice Densa

Definizione 3.1. Una matrice $A \in \mathbb{R}^{n \times n}$ si dice **densa** se la maggior parte dei suoi elementi è non nulla.

Matrice Sparsa

Definizione 3.2. Una matrice $A \in \mathbb{R}^{n \times n}$ si dice **sparsa** se possiede un numero di elementi non nulli dell'ordine di n. Per memorizzare una matrice sparsa è possibile utilizzare solo tre vettori: uno per gli elementi non nulli, uno per gli indici di riga e uno per gli indici di colonna.

Matrici Strutturate

Definizione 3.3. Una matrice $A \in \mathbb{R}^{n \times n}$ si dice **strutturata** se i suoi elementi sono disposti secondo una regola nota.

- **Matrice Diagonale:** $a_{i,j} = 0$ per $i \neq j$.

$$\begin{pmatrix} a_{1,1} & 0 & \dots & 0 \\ 0 & a_{2,2} & \dots & \vdots \\ \vdots & \vdots & \ddots & 0 \\ 0 & \dots & 0 & a_{n,n} \end{pmatrix}$$

- **Matrice Triangolare Inferiore:** $a_{i,j} = 0$ per $i < j$.

$$\begin{pmatrix} a_{1,1} & 0 & \dots & 0 \\ a_{2,1} & a_{2,2} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \dots & a_{n,n} \end{pmatrix}$$

- **Matrice Triangolare Superiore:** $a_{i,j} = 0$ per $i > j$.

$$\begin{pmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,n} \\ 0 & a_{2,2} & \dots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & a_{n,n} \end{pmatrix}$$

- **Matrice Tridiagonale:** elementi non nulli solo sulla diagonale principale, sulla sovra-diagonale e sulla sotto-diagonale.

$$\begin{pmatrix} a_{1,1} & a_{1,2} & 0 & \dots & 0 \\ a_{2,1} & a_{2,2} & a_{2,3} & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & a_{n-1,n-2} & a_{n-1,n-1} & a_{n-1,n} \\ 0 & \dots & 0 & a_{n,n-1} & a_{n,n} \end{pmatrix}$$

- **Hessenberg Inferiore:** $a_{i,j} = 0$ per $j > i + 1$ (elementi nulli sopra la prima sovra-diagonale).

$$\begin{pmatrix} a_{1,1} & a_{1,2} & 0 & \dots & 0 \\ a_{2,1} & a_{2,2} & a_{2,3} & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & 0 \\ a_{n-1,1} & a_{n-1,2} & \dots & a_{n-1,n-1} & a_{n-1,n} \\ a_{n,1} & a_{n,2} & \dots & a_{n,n-1} & a_{n,n} \end{pmatrix}$$

- **Hessenberg Superiore:** $a_{i,j} = 0$ per $i > j + 1$ (elementi nulli sotto la prima sotto-diagonale).

$$\begin{pmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,n-1} & a_{1,n} \\ a_{2,1} & a_{2,2} & \dots & a_{2,n-1} & a_{2,n} \\ 0 & a_{3,2} & \ddots & \vdots & \vdots \\ \vdots & \ddots & \ddots & a_{n-1,n-1} & a_{n-1,n} \\ 0 & \dots & 0 & a_{n,n-1} & a_{n,n} \end{pmatrix}$$

Matrice Simmetrica

Definizione 3.4. Una matrice A si dice **simmetrica** se coincide con la sua trasposta, cioè $A = A^T$.

Matrice Definita Positiva

Definizione 3.5. Una matrice simmetrica A si dice **definita positiva** se per ogni vettore $y \in \mathbb{R}^n, y \neq 0$ si ha $y^T A y > 0$.

Criterio di Sylvester

Proposizione 3.1. *Una matrice simmetrica $A \in \mathbb{R}^{n \times n}$ è definita positiva se e solo se:*

$$\det(A_k) > 0, \quad k \in \{1, \dots, n\}$$

dove $\det(A_k)$ è il determinante della sottomatrice principale di testa di ordine k.

Matrice a Diagonale Dominante

Definizione 3.6. Una matrice A è detta a **diagonale dominante per righe** se:

$$|a_{i,i}| > \sum_{j=1, j\neq i}^{n} |a_{i,j}|, \quad i \in \{1, \dots, n\}.$$

È detta a **diagonale dominante per colonne** se:

$$|a_{j,j}| > \sum_{i=1, i\neq j}^{n} |a_{i,j}|, \quad j \in \{1, \dots, n\}.$$

Norme Vettoriali

Definizione 3.7. Una **norma vettoriale** è una funzione che associa ad un vettore $x \in \mathbb{R}^n$ un numero reale $||x||$ con le seguenti proprietà:

1. $||x|| > 0, \forall x \neq 0$ e $||x|| = 0 \iff x = 0$;
2. $||cx|| = |c|||x||, \forall c \in \mathbb{R}$;
3. $||x + y|| \leq ||x|| + ||y||, \forall y \in \mathbb{R}^n$.

Le norme più utilizzate sono:

- **Norma infinito:**
$$||x||_\infty = \max_{1\leq i\leq n} |x_i|;$$
- **Norma 1:**
$$||x||_1 = \sum_{i=1}^{n} |x_i|;$$
- **Norma Euclidea (o norma 2):**
$$||x||_2 = \sqrt{\sum_{i=1}^{n} |x_i|^2} = \sqrt{x^T x}.$$

Esempio 3.1. Consideriamo $x = (1, -2, 3)^T$.

- $||x||_\infty = \max\{1, 2, 3\} = 3$;
- $||x||_1 = 1 + 2 + 3 = 6$;
- $||x||_2 = \sqrt{1 + 4 + 9} = \sqrt{14}$.

Norme Matriciali

Definizione 3.8. Una **norma matriciale** associa ad una matrice A un numero reale $||A||$ soddisfacendo proprietà analoghe a quelle vettoriali, con l'aggiunta della proprietà di sub-moltiplicatività:

- $||AB|| \leq ||A||||B||, \forall B \in \mathbb{R}^{n\times n}$.

Le norme matriciali più utilizzate sono:

- **Norma infinito:**
$$||A||_\infty = \max_{1\le i\le n} \sum_{j=1}^{n} |a_{i,j}|$$
(massima somma per righe).
- **Norma 1:**
$$||A||_1 = \max_{1\le j\le n} \sum_{i=1}^{n} |a_{i,j}|$$
(massima somma per colonne).
- **Norma Spettrale (o norma 2):**
$$||A||_2 = \sqrt{\rho(A^T A)},$$
dove $\rho(B)$ è il raggio spettrale di B (modulo massimo degli autovalori).
- **Norma di Frobenius:**
$$||A||_F = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n} |a_{i,j}|^2}.$$

Esempio 3.2. Sia $A = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 3 & 4 \\ 0 & 4 & 3 \end{pmatrix}$.

- $||A||_\infty = \max(2, 3+4, 4+3) = 7$ (somma righe);
- $||A||_1 = \max(2, 3+4, 4+3) = 7$ (somma colonne);
- $||A||_2$: Poiché A è simmetrica, $||A||_2 = \max|\lambda(A)|$. Gli autovalori sono $\lambda_1 = 2$ e, dal blocco inferiore, $\lambda_{2,3} = 3 \pm 4 \Rightarrow \{7, -1\}$. Quindi $||A||_2 = \max\{2, 7, |-1|\} = 7$.
- $||A||_F = \sqrt{2^2 + 3^2 + 4^2 + 0 + 4^2 + 3^2} = \sqrt{54} \approx 7.35$;

Norme Compatibili e Indotte

Definizione 3.9. Una norma vettoriale e una matriciale sono **compatibili** se $||Ax|| \le ||A||||x||$. Una norma matriciale si dice **naturale** o **indotta** dalla norma vettoriale se:
$$||A|| = \sup_{x\neq 0} \frac{||Ax||}{||x||} = \sup_{||x||=1} ||Ax||.$$
Per le norme naturali vale $||I|| = 1$. Le norme matriciali $||\cdot||_\infty, ||\cdot||_1$ e $||\cdot||_2$ sono naturali (indotte dalle rispettive norme vettoriali), mentre la norma di Frobenius non lo è (poiché $||I||_F = \sqrt{n}$).

Condizionamento di un sistema lineare Vogliamo ora studiare il condizionamento del sistema lineare $Ax = b$. Ci chiediamo come il problema "risponda" a eventuali perturbazioni nei dati iniziali (matrice A e vettore b). Supponiamo di introdurre una perturbazione $\Delta A \in \mathbb{R}^{n\times n}$ sulla matrice A e una perturbazione $\Delta b \in \mathbb{R}^n$ sul vettore dei termini noti b. La soluzione del problema perturbato non sarà x ma $y = x + \Delta x$, con $\Delta x \in \mathbb{R}^n$. Il sistema perturbato è:
$$(A + \Delta A)(x + \Delta x) = (b + \Delta b).$$

Teorema (Principale)

Teorema 3.2. *Siano $A \in \mathbb{R}^{n\times n}$ una matrice non singolare e $\Delta A \in \mathbb{R}^{n\times n}$ tale che sia soddisfatta la condizione*

$$||A^{-1}||||\Delta A|| \leq \frac{1}{2}$$

per una generica norma matriciale indotta $||\cdot||$. Allora anche la matrice $A + \Delta A$ è non singolare e se $x \in \mathbb{R}^n$ è soluzione del sistema $Ax = b$ con $b \in \mathbb{R}^n$ ($b \neq 0$) e $\Delta x \in \mathbb{R}^n$ verifica il sistema perturbato $(A + \Delta A)(x + \Delta x) = (b + \Delta b)$ per $\Delta b \in \mathbb{R}^n$, si ha che

$$\frac{||\Delta x||}{||x||} \leq 2K(A)\left(\frac{||\Delta A||}{||A||} + \frac{||\Delta b||}{||b||}\right)$$

ove $K(A) = ||A||||A^{-1}||$ è il numero di condizionamento della matrice A.

Per dimostrare il precedente teorema ci occorre il seguente risultato:

Teorema (Ausiliario)

Teorema 3.3. *Sia A una matrice quadrata, se $||A|| < 1$ allora la matrice $I + A$ è invertibile e vale la seguente disuguaglianza*

$$||(I + A)^{-1}|| \leq \frac{1}{1 - ||A||}$$

essendo $||\cdot||$ una norma matriciale indotta tale che $||A|| < 1$.

Proof. Poiché $A + \Delta A = A(I + A^{-1}\Delta A)$ e, per ipotesi,

$$||A^{-1}\Delta A|| \leq ||A^{-1}||||\Delta A|| \leq \frac{1}{2} < 1$$

applicando il teorema ausiliario, si ha che la matrice $I + A^{-1}\Delta A$ e quindi la matrice $A + \Delta A$, è non singolare e risulta

$$||(I + A^{-1}\Delta A)^{-1}|| \leq \frac{1}{1 - ||A^{-1}||||\Delta A||} \leq 2$$

Sottraendo membro a membro le equazioni $Ax = b$ e $(A + \Delta A)(x + \Delta x) = (b + \Delta b)$ si ottiene

$$(A + \Delta A)\Delta x = -\Delta A x + \Delta b$$

moltiplicando ambo i membri per A^{-1} si ha

$$(I + A^{-1}\Delta A)\Delta x = A^{-1}(-\Delta A x + \Delta b)$$

da cui

$$\Delta x = (I + A^{-1}\Delta A)^{-1}A^{-1}(-\Delta A x + \Delta b)$$

e passando alle norme

$$||\Delta x|| \leq 2||A^{-1}||(||\Delta A||||x|| + ||\Delta b||)$$

Dividendo e moltiplicando per $||A||$ a secondo membro, si ha

$$||\Delta x|| \leq 2K(A)\left(\frac{||\Delta A||}{||A||}||x|| + \frac{||\Delta b||}{||A||}\right)$$

Poiché per ipotesi è $b \neq 0$ e A è non singolare, risulta $||x|| > 0$ per cui dividendo ambo i membri per $||x||$ e tenendo conto che $||b|| = ||Ax|| \leq ||A||||x||$, si ha

$$\frac{||\Delta x||}{||x||} \leq 2K(A)\left(\frac{||\Delta A||}{||A||} + \frac{||\Delta b||}{||A||||x||}\right) \leq 2K(A)\left(\frac{||\Delta A||}{||A||} + \frac{||\Delta b||}{||b||}\right)$$

Dunque, otteniamo

$$\frac{||\Delta x||}{||x||} \leq 2K(A)\left(\frac{||\Delta A||}{||A||} + \frac{||\Delta b||}{||b||}\right)$$

Dalla stima si evince che l'eventuale amplificazione delle perturbazioni introdotte su matrice dei coefficienti e vettore dei termini noti si deve alla quantità $K(A) = ||A||||A^{-1}||$. Tale quantità viene chiamata indice o numero di condizionamento della matrice A. □

Osservazioni sulla stima dell'errore Dalla stima ottenuta si evincono alcune proprietà fondamentali:

- Per ogni norma matriciale $||\cdot||$ naturale (che soddisfi la proprietà di sub-moltiplicatività), si ha:

$$K(A) = ||A||||A^{-1}|| \geq ||AA^{-1}|| = ||I|| = 1$$

- Il condizionamento è una caratteristica propria del sistema lineare e non è legato al particolare metodo numerico usato per determinarne la soluzione.
- Ponendo $K_\infty(A) = ||A||_\infty ||A^{-1}||_\infty$, dalla stima dell'errore relativo, se:

$$\frac{||\Delta x||_\infty}{||x||_\infty} \leq 2K_\infty(A)\text{eps} \leq \frac{1}{2}10^{1-p}$$

  allora p cifre significative della soluzione calcolata mediante un algoritmo stabile sono da ritenere corrette.

Calcolo del Condizionamento in Matlab In Matlab esiste una function predefinita per calcolare $K(A)$:

- `cond(A, inf)`: calcola il condizionamento di A in norma infinito ($K_\infty(A)$).
- `cond(A, 1)`: calcola il condizionamento di A in norma 1 ($K_1(A)$).
- `cond(A)`: calcola il condizionamento di A in norma 2 ($K_2(A)$).

Esempi di Matrici Mal Condizionate

Matrici di Hilbert

Esempio 3.3. Sono matrici definite da elementi $h_{i,j} = \frac{1}{i+j-1}$. La struttura della matrice H_n è la seguente:

$$H_n = \begin{pmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \cdots & \frac{1}{n} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \cdots & \frac{1}{n+1} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \cdots & \frac{1}{n+2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{1}{n} & \frac{1}{n+1} & \frac{1}{n+2} & \cdots & \frac{1}{2n-1} \end{pmatrix}$$

Tale matrice risulta essere mal condizionata anche per piccoli valori di n. Ad esempio, per $n = 5$ si ha:

$$\text{cond}(H_5) \simeq 10^5$$

Matrici di Vandermonde

Esempio 3.4. Fissato un vettore $[y_1, \ldots, y_n]$, con $y_i \neq y_j$ per $i \neq j$, sono matrici non singolari definite da:

$$V_n = \begin{pmatrix} 1 & 1 & 1 & \ldots & 1 \\ y_1 & y_2 & y_3 & \cdots & y_n \\ y_1^2 & y_2^2 & y_3^2 & \cdots & y_n^2 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ y_1^{n-1} & y_2^{n-1} & y_3^{n-1} & \ldots & y_n^{n-1} \end{pmatrix}$$

Anche in questo caso l'indice di condizionamento cresce esponenzialmente rispetto alla dimensione della matrice.

Classificazione dei Metodi Numerici I metodi numerici si suddividono in:

- **Metodi diretti:** In assenza di errori di arrotondamento calcolano la soluzione esatta in un numero finito di passi.
- **Metodi iterativi:** Generano una successione infinita di vettori che converge alla soluzione.

La scelta dipende da stabilità, occupazione di memoria e costo computazionale.

Metodi Diretti e Sparsezza I metodi diretti trasformano il problema in problemi equivalenti. Sono efficienti per matrici dense. Tuttavia, per matrici sparse, i metodi diretti causano il fenomeno del *fill-in* (riempimento): il numero di elementi non nulli cresce durante il procedimento, potendo saturare la memoria. In questi casi sono preferibili i metodi iterativi, che lasciano inalterata la matrice A.

3.2 Sistemi Diagonali e Triangolari

Metodi Diretti Si chiamano metodi diretti quei metodi numerici che risolvono sistemi lineari in un numero finito di passi. In altri termini, supponendo di effettuare i calcoli in precisione infinita, tali metodi forniscono la soluzione esatta del sistema mediante un numero finito di operazioni, noto a priori. In precisione di calcolo finita invece, se il metodo proposto è stabile, esso fornisce una soluzione approssimata con la precisione massima ottenibile, cioè in accordo con il condizionamento della matrice dei coefficienti.

Sistemi Diagonali Il più banale dei metodi diretti si ottiene nel caso delle matrici diagonali. Se denotiamo con $x = (x_1, x_2, \dots, x_n)^T$ e $b = (b_1, b_2, \dots, b_n)^T$ rispettivamente il vettore delle incognite e quello dei termini noti, e se la matrice dei coefficienti è del tipo:

$$D = \begin{pmatrix} a_{1,1} & 0 & 0 & \dots & 0 \\ 0 & a_{2,2} & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \dots & \vdots \\ 0 & 0 & 0 & \dots & a_{n,n} \end{pmatrix}$$

il sistema $Dx = b$, nell'ipotesi che D sia non singolare (cioè $a_{i,i} \neq 0, \forall i \in \{1, \dots, n\}$), si risolve banalmente mediante le n divisioni:

$$x_i = \frac{b_i}{a_{i,i}}, \quad i \in \{1, \dots, n\}$$

con un costo computazionale complessivo di n operazioni di divisione. L'algoritmo è ben posto se la matrice è non singolare. Inoltre, poiché l'algoritmo è composto da n divisioni indipendenti l'una dall'altra, esso è banalmente stabile.

Sistemi Triangolari Consideriamo ora il caso dei sistemi con matrice dei coefficienti triangolare. Denotiamo con L una generica matrice triangolare inferiore ("lower triangular"):

$$L = \begin{pmatrix} l_{1,1} & 0 & 0 & \dots & 0 \\ l_{2,1} & l_{2,2} & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \dots & \vdots \\ l_{n-1,1} & l_{n-1,2} & \dots & l_{n-1,n-1} & 0 \\ l_{n,1} & l_{n,2} & \dots & l_{n,n-1} & l_{n,n} \end{pmatrix}$$

e con U una generica matrice triangolare superiore ("upper triangular"):

$$U = \begin{pmatrix} u_{1,1} & u_{1,2} & \dots & u_{1,n-1} & u_{1,n} \\ 0 & u_{2,2} & \dots & u_{2,n-1} & u_{2,n} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & u_{n-1,n-1} & u_{n-1,n} \\ 0 & 0 & \dots & 0 & u_{n,n} \end{pmatrix}$$

Vogliamo fornire degli algoritmi per la risoluzione dei sistemi $Lx = b$ e $Ux = b$. Una matrice triangolare è non singolare se e solo se i suoi elementi diagonali sono tutti diversi da zero. Infatti:

$$\det(L) = \prod_{i=1}^{n} l_{ii} \neq 0 \iff l_{i,i} \neq 0, \quad \forall i \in \{1, \dots, n\};$$

$$\det(U) = \prod_{i=1}^{n} u_{ii} \neq 0 \iff u_{i,i} \neq 0, \quad \forall i \in \{1, \dots, n\}.$$

Gli algoritmi per la risoluzione di sistemi con matrice dei coefficienti triangolare sono essenzialmente basati sulla tecnica di sostituzione.

Algoritmo di Sostituzione in Avanti (Forward Substitution) Esaminiamo il caso della matrice triangolare inferiore $Lx = b$. Esplicitamente il sistema può scriversi come:

$$\begin{cases} l_{1,1}x_1 = b_1 \\ l_{2,1}x_1 + l_{2,2}x_2 = b_2 \\ l_{3,1}x_1 + l_{3,2}x_2 + l_{3,3}x_3 = b_3 \\ \vdots \\ l_{n,1}x_1 + l_{n,2}x_2 + l_{n,3}x_3 + \cdots + l_{n,n}x_n = b_n \end{cases}$$

Si osserva che dalla prima equazione si ricava immediatamente x_1. Tale valore può essere sostituito nella seconda equazione per ricavare x_2, e così via.

$$x_1 = \frac{b_1}{l_{1,1}}$$

$$x_2 = \frac{b_2 - l_{2,1}x_1}{l_{2,2}}$$

$$x_3 = \frac{b_3 - l_{3,1}x_1 - l_{3,2}x_2}{l_{3,3}}$$

$$\cdots$$

$$x_i = \frac{b_i - \sum_{k=1}^{i-1} l_{i,k}x_k}{l_{i,i}}, \quad i \in \{2, \ldots, n\}$$

L'algoritmo è ben posto se la matrice L è non singolare.

Schema Algoritmo e Costo per Iterazione

```
x(1) = b(1) / L(1,1)                  % 1 flop (divisione)
for i = 2 : n
    x(i) = b(i)
    for k = 1 : i-1
        x(i) = x(i) - L(i,k) * x(k)   % 2 flops (1 mult, 1 sub)
    end
    x(i) = x(i) / L(i,i)              % 1 flop (divisione)
end
```

Calcolo del Costo Computazionale Per determinare il costo totale in termini di operazioni floating-point (flops), analizziamo le operazioni svolte per ogni riga i-esima.

- Il ciclo interno (k) esegue 2 operazioni (moltiplicazione e sottrazione) ripetute $i - 1$ volte.
- Al termine del ciclo interno viene eseguita 1 divisione.

Il costo per la riga i è quindi $2(i-1) + 1 = 2i - 1$ flops. Sommando su tutte le righe da 1 a n:

$$C(n) = \sum_{i=1}^{n}(2i - 1) = 2\sum_{i=1}^{n} i - \sum_{i=1}^{n} 1$$

Utilizzando la formula della somma dei primi n interi $\sum i = \frac{n(n+1)}{2}$:

$$C(n) = 2 \cdot \frac{n(n+1)}{2} - n = n^2 + n - n = n^2 \text{ flops.}$$

Implementazione Ottimizzata in MATLAB Nel passaggio dallo schema teorico all'implementazione pratica in ambiente MATLAB (funzione `forwardSub`), sono state introdotte alcune variazioni fondamentali per garantire efficienza e stabilità numerica:

- **Vettorizzazione (Vectorization):** In MATLAB, il ciclo interno (la sommatoria $\sum l_{i,k}x_k$) è sostituito da un prodotto matriciale riga-per-colonna tra la porzione della riga i-esima di L e la parte del vettore x già calcolata:

```
sommatoria = L(i, 1:i-1) * x(1:i-1);
```

Sebbene la complessità computazionale rimanga $O(n^2)$, questo approccio riduce l'overhead dell'interprete e sfrutta le librerie ottimizzate di algebra lineare (BLAS), risultando significativamente più veloce nell'esecuzione rispetto a un ciclo `for` esplicito.

- **Robustezza e Validazione:** A differenza dello pseudo-codice, la funzione reale include il blocco `arguments` per la validazione dei tipi e controlli espliciti sulle dimensioni. Inoltre, la verifica della non-singolarità non avviene controllando l'uguaglianza con zero ($L_{ii} = 0$), ma verificando che $|L_{ii}| <$ `eps` (epsilon macchina), per gestire correttamente gli errori di arrotondamento propri dell'aritmetica floating point.
- **Pre-allocazione:** Il vettore soluzione viene inizializzato staticamente con `zeros(n,1)` prima del ciclo, evitando costose riallocazioni dinamiche della memoria durante l'esecuzione.

```
function x = forwardSub(L, b)
% FORWARDSUB Risolve il sistema lineare triangolare inferiore Lx = b usando la vettorizzazione (più veloce in esecuzione).
%   x = FORWARDSUB(L, b) restituisce il vettore soluzione x.
%
%   INPUT:
%       L - Matrice quadrata triangolare inferiore (n x n)
%       b - Vettore dei termini noti (n x 1)
%
%   OUTPUT:
%       x - Vettore soluzione (n x 1)

    %% 1. Validazione degli Input
    arguments
        L (:,:) double {mustBeNumeric}
        b (:,1) double {mustBeNumeric}
    end

    %% 2. Controllo Dimensioni
    % Ricaviamo le dimensioni
    n = size(L, 1);

    % Verifichiamo che L sia quadrata
    if size(L, 2) ~= n
        error('ForwardSub:NonSquareMatrix', 'La matrice L deve essere quadrata.');
    end

    % Verifichiamo che b sia compatibile con L
    if length(b) ~= n
        error('ForwardSub:DimensionMismatch', 'Le dimensioni di L e b non corrispondono.');
    end

    %% 3. Controllo Singolarità
    % Se c'è uno zero sulla diagonale, non possiamo dividere.
    % Usiamo una piccola tolleranza per i numeri floating point.
    if any(abs(diag(L)) < eps)
        error('ForwardSub:SingularMatrix', 'La matrice è singolare.');
    end

    %% 4. Algoritmo Vettorizzato
    % Pre-allocazione
    x = zeros(n, 1);

    % Calcoliamo il primo elemento manualmente per avviare il processo
    x(1) = b(1) / L(1,1);

    for i = 2 : n
        % Prendiamo la riga i-esima di L (fino alla colonna i-1)
        % e il vettore x (fino all'elemento i-1).
        % Il loro prodotto matriciale (*) coincide esattamente con la sommatoria che ci serve.
        sommatoria = L(i, 1:i-1) * x(1:i-1);

        % Calcolo finale
        x(i) = (b(i) - sommatoria) / L(i,i);
    end
end
```

Funzione forwardSub

Di seguito viene mostrato uno script di esempio che utilizza la funzione `forwardSub` per risolvere un sistema lineare triangolare, riportando il codice e i risultati numerici ottenuti.

```
% 1. Impostiamo il formato numerico
format long e

% 2. Generazione dati (Generalizzato per dimensione n)
n = 4;
L = tril(rand(n));          % Matrice triangolare inferiore
b = randi([-10, 10], n, 1); % Termine noto

disp('Matrice L:'); disp(L);
disp('Vettore b:'); disp(b);

% 3. Algoritmo di Forward Substitution (Sostituzione in avanti)
x = forwardSub(L,b);

% 4. Output e Verifica
disp('Soluzione x calcolata:');
disp(x);

% 5. Calcolo del Residuo
% Calcoliamo la norma della differenza
residuo = norm(L*x - b);
fprintf('Norma del residuo ||Lx - b||: %e\n', residuo);
```

```
Matrice L:
     4.624491592423287e-01                     0                     0
     4.243490398153752e-01     7.847392947607416e-01                     0
     4.609163660289640e-01     4.713571537106116e-01     4.734859929653203e-01
     7.701597286086093e-01     3.576273326911794e-02     1.527212004382318e-01     7.384268399769416e-0

Vettore b:
    -5
     9
    -5
     6

Soluzione x calcolata:
    -1.081199933024409e+01
     1.731538311512285e+01
    -1.727253262241360e+01
     2.213571077577904e+01

Norma del residuo ||Lx - b||: 8.881784e-16
```

Algoritmo di Sostituzione all'Indietro (Backward Substitution) Si vuole risolvere il sistema $Ux = b$ dove U è triangolare superiore:

$$\begin{cases} u_{1,1}x_1 + u_{1,2}x_2 + \cdots + u_{1,n}x_n = b_1 \\ u_{2,2}x_2 + \cdots + u_{2,n}x_n = b_2 \\ \vdots \\ u_{n-1,n-1}x_{n-1} + u_{n-1,n}x_n = b_{n-1} \\ u_{n,n}x_n = b_n \end{cases}$$

Procedendo dall'ultima equazione verso la prima:

$$x_n = \frac{b_n}{u_{n,n}}$$

$$x_{n-1} = \frac{b_{n-1} - u_{n-1,n}x_n}{u_{n-1,n-1}}$$

$$\cdots$$

$$x_i = \frac{b_i - \sum_{k=i+1}^{n} u_{i,k}x_k}{u_{i,i}}, \quad i \in \{n-1, \ldots, 1\}$$

Anche in questo caso l'algoritmo è ben posto se U è non singolare.

Schema Algoritmo e Costo per Iterazione

```
x(n) = b(n) / U(n,n)                    % 1 flop (divisione)
for i = n-1 : -1 : 1
    x(i) = b(i)
    for k = i+1 : n
        x(i) = x(i) - U(i,k) * x(k)     % 2 flops (1 mult, 1 sub)
    end
    x(i) = x(i) / U(i,i)                % 1 flop (divisione)
end
```

Calcolo del Costo Computazionale Analizziamo il costo per una generica riga i-esima.

- Il ciclo interno su k va da $i+1$ a n. Il numero di termini nella sommatoria è quindi $n-(i+1)+1 = n-i$.
- Per ogni termine del ciclo si eseguono 2 operazioni (moltiplicazione e sottrazione).
- Al termine del ciclo si esegue 1 divisione.

Il costo per calcolare x_i è quindi $2(n-i)+1$ flops. Sommando per tutte le righe da 1 a n (notando che per $i = n$ il costo è semplicemente 1):

$$C(n) = \sum_{i=1}^{n} [2(n-i)+1]$$

Ponendo $j = n-i$ (cambio di indice per semplificare la somma), quando $i = 1 \to j = n-1$ e quando $i = n \to j = 0$:

$$C(n) = \sum_{j=0}^{n-1} (2j+1) = 2\sum_{j=0}^{n-1} j + \sum_{j=0}^{n-1} 1$$

$$C(n) = 2\frac{(n-1)n}{2} + n = n^2 - n + n = n^2 \text{ flops.}$$

Implementazione Ottimizzata in MATLAB Analogamente a quanto visto per la sostituzione in avanti, l'implementazione della funzione `backwardSub` introduce ottimizzazioni cruciali rispetto allo schema teorico:

- **Vettorizzazione (Vectorization):** Invece di iterare scalarmente con un ciclo interno, si calcola la sommatoria $\sum_{k=i+1}^{n} u_{i,k}x_k$ mediante un prodotto matriciale (dot product). In MATLAB, questo si traduce nel moltiplicare la parte di riga a destra della diagonale, `U(i, i+1:n)`, per il vettore delle soluzioni già calcolate, `x(i+1:n)`. Questo approccio sfrutta l'efficienza delle operazioni su blocchi di memoria contigui.
- **Robustezza Numerica:** La funzione include la validazione degli argomenti e, soprattutto, un controllo di non-singolarità basato sulla tolleranza macchina (`eps`). Se un elemento diagonale $|u_{ii}| < \varepsilon$, l'algoritmo si arresta controllatamente per evitare divisioni instabili o per zero.

```
function x = backwardSub(U, b)
% FORWARDSUB Risolve il sistema lineare triangolare superiore Ux = b usando la vettorizzazione (più veloce in esecuzione).
%   x = BACKWARDSUB(U, b) restituisce il vettore soluzione x.
%
%   INPUT:
%       U - Matrice quadrata triangolare superiore (n x n)
%       b - Vettore dei termini noti (n x 1)
%
%   OUTPUT:
%       x - Vettore soluzione (n x 1)

    %% 1. Validazione degli Input
    arguments
        U (:,:) double {mustBeNumeric}
        b (:,1) double {mustBeNumeric}
    end

    %% 2. Controllo Dimensioni
    % Ricaviamo le dimensioni
    n = size(U, 1);

    % Verifichiamo che U sia quadrata
```

```
    if size(U, 2) ~= n
        error('BackwardSub:NonSquareMatrix', 'La matrice U deve essere quadrata.');
    end

    % Verifichiamo che b sia compatibile con U
    if length(b) ~= n
        error('BackwardSub:DimensionMismatch', 'Le dimensioni di U e b non corrispondono.');
    end

    %% 3. Controllo Singolarità
    % Se c'è uno zero sulla diagonale, non possiamo dividere.
    % Usiamo una piccola tolleranza per i numeri floating point.
    if any(abs(diag(U)) < eps)
        error('BackwardSub:SingularMatrix', 'La matrice è singolare.');
    end

    %% 4. Algoritmo Vettorizzato
    % Pre-allocazione
    x = zeros(n, 1);

    % Calcoliamo l'ultimo elemento manualmente per avviare il processo
    x(n) = b(n) / U(n,n);

    for i = n-1 : -1 : 1
        % Prendi la porzione di riga i-esima a destra della diagonale: U(i, i+1:n)
        % Prendi la porzione di soluzione x già calcolata (che sta "sotto"): x(i+1:n)
        % Il loro prodotto matriciale (Riga * Colonna) è la sommatoria.
        sommatoria = U(i, i+1:n) * x(i+1:n);

        % Calcolo finale
        x(i) = (b(i) - sommatoria) / U(i,i);
    end
end
```

Funzione backwardSub

Di seguito viene riportato il codice della funzione e un esempio applicativo con i relativi risultati.

```
% 1. Impostiamo il formato numerico
format long e

% 2. Generazione dati (Generalizzato per dimensione n)
n = 4;
U = triu(rand(n));          % Matrice triangolare superiore
b = randi([-10, 10], n, 1); % Termine noto

disp('Matrice U:'); disp(U);
disp('Vettore b:'); disp(b);

% 3. Algoritmo di Backward Substitution (Sostituzione all'indietro)

x = backwardSub(U,b);

% 4. Output e Verifica
disp('Soluzione x calcolata:');
disp(x);

% 5. Calcolo del Residuo
% Calcoliamo la norma della differenza
residuo = norm(U*x - b);
fprintf('Norma del residuo ||Ux - b||: %e\n', residuo);
```

```
Matrice U:
     9.727338850797841e-01     9.541744563795431e-01     2.815015591484906e-01     5.906086529196359e-0
                         0     3.192262950397839e-02     2.303830673174637e-01     6.604379663126019e-0
                         0                         0     7.111285511803251e-01     4.755467311386607e-0
                         0                         0                         0     3.487848085100589e-0

Vettore b:
    -1
    -5
     5
     7
```

```
Soluzione x calcolata:
     5.863474614137475e+02
    -6.129013653804130e+02
     5.688975435765223e+00
     2.006968144599715e+01

Norma del residuo ||Ux - b||: 2.494817e-14
```

Calcolo dell'Inversa di una Matrice Triangolare Il metodo di sostituzione all'indietro può essere adattato per il calcolo esplicito dell'inversa di una matrice triangolare superiore U. I vettori colonna $\mathbf{v}_i$ dell'inversa $V = (\mathbf{v}_1, \ldots, \mathbf{v}_n)$ di U soddisfano i sistemi lineari:

$$U\mathbf{v}_i = \mathbf{e}_i, \quad i \in \{1, \ldots, n\}$$

dove $\mathbf{e}_i$ è l'i-esima colonna della matrice identità. Poiché l'inversa di una matrice triangolare superiore è anch'essa triangolare superiore, le colonne di V hanno tutti gli elementi subdiagonali nulli. Sfruttando questa proprietà, indichiamo con $\mathbf{v}'_j = (\mathbf{v}_{1,j}, \ldots, \mathbf{v}_{j,j})^T$ il vettore di dimensione j tale che:

$$U_j\mathbf{v}'_j = \mathbf{e}'_j, \quad j \in \{1, \ldots, n\}$$

essendo U_j la sottomatrice principale di U di ordine j e $\mathbf{e}'_j$ il vettore di $\mathbb{R}^j$ con tutte le componenti nulle fuorché l'ultima che è 1. I sistemi sono triangolari superiori di ordine j e si risolvono con l'eliminazione all'indietro.

Algoritmo di Inversione Per $j \in \{1, \ldots, n\}$:

$$\mathbf{v}_{j,j} = \frac{1}{u_{j,j}}$$

$$\mathbf{v}_{i,j} = -\frac{1}{u_{i,i}} \sum_{k=i+1}^{j} u_{i,k}\mathbf{v}_{k,j}, \quad \text{per } i \in \{j-1, \ldots, 1\}$$

Schema Algoritmo

```
for j = 1 : n
    v(j,j) = 1 / U(j,j)
    for i = j-1 : -1 : 1
        v(i,j) = 0
        for k = i+1 : j
            v(i,j) = v(i,j) - U(i,k) * v(k,j)
        end
        v(i,j) = v(i,j) / U(i,i)
    end
end
```

Costo Computazionale

$$\sum_{j=1}^{n}\sum_{i=1}^{j}(j-i+1) \simeq \frac{n^3}{6} \text{ operazioni}$$

Una procedura analoga, basata sulla sostituzione in avanti, permette di calcolare l'inversa di una matrice triangolare inferiore. Da fare.

3.3 Metodo di Eliminazione di Gauss

Metodo di eliminazione di Gauss Consideriamo il caso di una generica matrice di ordine n. Sia dunque $A \in \mathbb{R}^{n\times n}$ e come al solito $x, b \in \mathbb{R}^n$. Vogliamo risolvere il generico sistema $Ax = b$. Il metodo di Gauss è basato sulla tecnica di eliminazione delle incognite da fissate equazioni. Obiettivo del metodo è ridurre, attraverso un numero finito di passi, il sistema di partenza ad uno ad esso equivalente che abbia la matrice dei coefficienti triangolare.

Descrizione del metodo (caso $n = 4$) Per semplicità, incominciamo con il descrivere il metodo nel caso di un sistema di dimensione $n = 4$.

$$\begin{cases} a_{1,1}x_1 + a_{1,2}x_2 + a_{1,3}x_3 + a_{1,4}x_4 = b_1 \\ a_{2,1}x_1 + a_{2,2}x_2 + a_{2,3}x_3 + a_{2,4}x_4 = b_2 \\ a_{3,1}x_1 + a_{3,2}x_2 + a_{3,3}x_3 + a_{3,4}x_4 = b_3 \\ a_{4,1}x_1 + a_{4,2}x_2 + a_{4,3}x_3 + a_{4,4}x_4 = b_4 \end{cases}$$

Assumiamo che $a_{1,1} \neq 0$ e consideriamo la quantità $m_{2,1} = -\frac{a_{2,1}}{a_{1,1}}$. Moltiplicando ambo i membri della prima equazione per $m_{2,1}$ e sommando alla seconda equazione, si ottiene:

$$(-a_{2,1} + a_{2,1})x_1 + (a_{2,2} - \frac{a_{2,1}}{a_{1,1}}a_{1,2})x_2 + \cdots + (a_{2,4} - \frac{a_{2,1}}{a_{1,1}}a_{1,4})x_4 = (b_2 - \frac{a_{2,1}}{a_{1,1}}b_1)$$

cioè

$$a^{(2)}_{2,2}x_2 + a^{(2)}_{2,3}x_3 + a^{(2)}_{2,4}x_4 = b^{(2)}_2$$

dunque

$$a^{(2)}_{2,j} = a_{2,j} + m_{2,1}a_{1,j}, \quad j = 2, 3, 4$$

Ripetendo il procedimento per la terza e quarta equazione con i moltiplicatori $m_{3,1} = -\frac{a_{3,1}}{a_{1,1}}$ e $m_{4,1} = -\frac{a_{4,1}}{a_{1,1}}$, otteniamo il sistema equivalente al passo 2:

$$\begin{cases} a_{1,1}x_1 + a_{1,2}x_2 + a_{1,3}x_3 + a_{1,4}x_4 = b_1 \\ a^{(2)}_{2,2}x_2 + a^{(2)}_{2,3}x_3 + a^{(2)}_{2,4}x_4 = b^{(2)}_2 \\ a^{(2)}_{3,2}x_2 + a^{(2)}_{3,3}x_3 + a^{(2)}_{3,4}x_4 = b^{(2)}_3 \\ a^{(2)}_{4,2}x_2 + a^{(2)}_{4,3}x_3 + a^{(2)}_{4,4}x_4 = b^{(2)}_4 \end{cases}$$

dove in generale $a^{(2)}_{i,j} = a_{i,j} + m_{i,1}a_{1,j}$ e $b^{(2)}_i = b_i + m_{i,1}b_1$.

Assumendo ora che $a^{(2)}_{2,2} \neq 0$, eliminiamo l'incognita x_2 dalla terza e quarta equazione usando i moltiplicatori $m_{3,2} = -\frac{a^{(2)}_{3,2}}{a^{(2)}_{2,2}}$ e $m_{4,2} = -\frac{a^{(2)}_{4,2}}{a^{(2)}_{2,2}}$. Si ottiene il sistema al passo 3:

$$\begin{cases} a_{1,1}x_1 + a_{1,2}x_2 + a_{1,3}x_3 + a_{1,4}x_4 = b_1 \\ a^{(2)}_{2,2}x_2 + a^{(2)}_{2,3}x_3 + a^{(2)}_{2,4}x_4 = b^{(2)}_2 \\ a^{(3)}_{3,3}x_3 + a^{(3)}_{3,4}x_4 = b^{(3)}_3 \\ a^{(3)}_{4,3}x_3 + a^{(3)}_{4,4}x_4 = b^{(3)}_4 \end{cases}$$

Infine, assumendo $a^{(3)}_{3,3} \neq 0$, eliminiamo x_3 dall'ultima equazione con $m_{4,3} = -\frac{a^{(3)}_{4,3}}{a^{(3)}_{3,3}}$, ottenendo il sistema triangolare finale:

$$\begin{cases} a_{1,1}x_1 + a_{1,2}x_2 + a_{1,3}x_3 + a_{1,4}x_4 = b_1 \\ a^{(2)}_{2,2}x_2 + a^{(2)}_{2,3}x_3 + a^{(2)}_{2,4}x_4 = b^{(2)}_2 \\ a^{(3)}_{3,3}x_3 + a^{(3)}_{3,4}x_4 = b^{(3)}_3 \\ a^{(4)}_{4,4}x_4 = b^{(4)}_4 \end{cases}$$

Generalizzazione al caso di ordine n Assumiamo che sia $a_{1,1} \neq 0$. Possiamo eliminare l'incognita x_1 dalla $2^a, \ldots, n$-esima equazione sommando all'i-esima equazione $(i = 2, \ldots, n)$ la prima equazione moltiplicata per $m_{i,1} = -\frac{a_{i,1}}{a_{1,1}}$. Le formule di aggiornamento generali al passo k (per eliminare l'incognita x_k dalle equazioni successive) sono:

$$m_{i,k} = -\frac{a^{(k)}_{i,k}}{a^{(k)}_{k,k}}, \quad i \in \{k+1, \ldots, n\}$$

$$a^{(k+1)}_{i,j} = a^{(k)}_{i,j} + m_{i,k}a^{(k)}_{k,j}, \quad j \in \{k+1, \ldots, n\}$$

$$b^{(k+1)}_i = b^{(k)}_i + m_{i,k}b^{(k)}_k$$

Gli elementi $a_{1,1}, a^{(2)}_{2,2}, a^{(3)}_{3,3}, \ldots$ che compaiono al denominatore vengono detti *elementi pivot*, mentre le quantità $m_{i,k}$ sono dette **moltiplicatori**. Dopo $n-1$ passi, se tutti i pivot sono non nulli, si ottiene un sistema triangolare superiore risolvibile con la sostituzione all'indietro.

Esempio 3.5. Consideriamo il sistema:

$$\begin{cases} 2x_1 - x_2 + x_3 - 2x_4 = 0 \\ 2x_2 - x_4 = 1 \\ x_1 - 2x_3 + x_4 = 0 \\ 2x_2 + x_3 + x_4 = 4 \end{cases}.$$

1. **Passo 1:** $m_{2,1} = 0, m_{3,1} = -1/2, m_{4,1} = 0$. Il sistema diventa:

$$\begin{cases} 2x_1 - x_2 + x_3 - 2x_4 = 0 \\ 2x_2 - x_4 = 1 \\ \frac{1}{2}x_2 - \frac{5}{2}x_3 + 2x_4 = 0 \\ 2x_2 + x_3 + x_4 = 4 \end{cases}$$

2. **Passo 2:** $m_{3,2} = -1/4, m_{4,2} = -1$. Il sistema diventa:

$$\begin{cases} 2x_1 - x_2 + x_3 - 2x_4 = 0 \\ 2x_2 - x_4 = 1 \\ -\frac{5}{4}x_3 + \frac{9}{4}x_4 = -\frac{1}{4} \\ x_3 + 2x_4 = 3 \end{cases}$$

3. **Passo 3:** $m_{4,3} = \frac{4}{5}$. Il sistema finale è:

$$\begin{cases} 2x_1 - x_2 + x_3 - 2x_4 = 0 \\ 2x_2 - x_4 = 1 \\ -\frac{5}{4}x_3 + \frac{9}{4}x_4 = -\frac{1}{4} \\ \frac{38}{10}x_4 = \frac{28}{10} \implies x_4 = 1 \dots \end{cases}$$

Sostituendo all'indietro si ottiene la soluzione $x = (1, 1, 1, 1)^T$.

Algoritmo L'algoritmo formale per $k \in \{1, 2, \dots, n-1\}$ è :

$$m_{i,k} = -\frac{a_{i,k}^{(k)}}{a_{k,k}^{(k)}}, \quad i \in \{k+1, \dots, n\};$$

$$a_{i,j}^{(k+1)} = a_{i,j}^{(k)} + m_{i,k} a_{k,j}^{(k)}, \quad j \in \{k+1, \dots, n\};$$
$$b_i^{(k+1)} = b_i^{(k)} + m_{i,k} b_k^{(k)}, \quad i \in \{k+1, \dots, n\}.$$

L'algoritmo è ben posto se e solo se tutti gli elementi pivot sono diversi da zero.

Schema Algoritmo

```
for k = 1 : n-1
    for i = k+1 : n
        mol = -a(i,k) / a(k,k)      % 1 operazione
        for j = k+1 : n
            a(i,j) = a(i,j) + mol * a(k,j)  % n-k operazioni
        end
        b(i) = b(i) + mol * b(k)    % 2 operazione
    end
end
```

Costo Computazionale

$$\sum_{k=1}^{n-1}[2(n-k)+(n-k)^2] \simeq \frac{n^3}{3} \text{ operazioni}$$

```
function [Ag,bg] = gaussElim(A,b)
%GAUSSELIM Applica il metodo dell'eliminazione di Gauss alla matrice dei coefficienti e al
    vettore dei termini noti.
%  [Ag,bg] = gaussElim(A,b)  restituisce A e b risultanti dall'eliminazione.
%
%    INPUT:
%      A - Matrice dei coefficienti (n x n)
%      b - Vettore dei termini noti (n x 1)
%
%    OUTPUT:
%      Ag - Matrice dei coefficienti post-GE (n x n)
%      bg - Vettore dei termini noti post-GE (n x 1)

    %% 1. Validazione degli Input
    arguments
        A (:,:) double {mustBeNumeric}
        b (:,1) double {mustBeNumeric}
    end

    %% 2. Verifica Dimensioni
    [n, m] = size(A);

    % Verifico se la matrice è quadrata
    if n ~= m
        error('gaussElim:nonSquareMatrix', 'La matrice non è quadrata (Size %dx%d).', n, m);
    end

    % Verifico se il prodotto Ab è compatibile
    if n ~= length(b)
        error('gaussElim:nonCompatible', 'Il prodotto non è compatibile (Size (%dx%d)*(%dx1)).
    ', n, m, length(b));
    end


    %% 3. Eliminazione di Gauss
    for k = 1 : n-1
        % Controllo pivot nullo (sicurezza)
        if A(k,k) == 0
             error('gaussElim:zeroPivot', 'Pivot nullo alla riga %d.', k);
        end

        % Eliminazione
        for i = k+1 : n
            mol = -A(i,k) / A(k,k);
            A(i, k+1:n) = A(i, k+1:n) + mol * A(k, k+1:n);
            b(i) = b(i) + mol * b(k);
        end
    end

    Ag = A;
    bg = b;
end
```

Funzione gaussElim

Esistenza dei Pivot

Teorema 3.4. *Sia $A \in \mathbb{R}^{n\times n}$. Gli elementi pivot $a_{k,k}^{(k)}$ sono tutti diversi da zero se e soltanto se tutte le matrici principali di testa di A, denotate con $A_k = (a_{i,j})_{i,j=1,\dots,k}$, sono non singolari, cioè:*

$$\det(A_k) \neq 0 \quad \forall k \in \{1,\dots,n\}.$$

Poiché questa condizione è difficile da verificare, esistono famiglie di matrici per cui è garantita a priori:

- Matrici a **diagonale dominante per righe** o per colonne.

- Matrici **simmetriche definite positive** ($A^T = A$ e $y^T Ay > 0, \forall y \neq 0$).

3.4 Strategia Pivoting e Fattorizzazione LU

Necessità del Pivoting Il metodo di Gauss descritto in precedenza si arresta se, a un certo passo k, l'elemento pivot $a_{k,k}^{(k)}$ risulta nullo. Tuttavia, se la matrice è non singolare, deve esistere almeno un elemento $a_{i,k}^{(k)} \neq 0$ con $i > k$ sulla stessa colonna. Per proseguire è sufficiente scambiare la riga k-esima con la riga i-esima. Anche quando il pivot non è nullo ma è molto piccolo in valore assoluto rispetto agli altri elementi, possono verificarsi errori di arrotondamento disastrosi (amplificazione degli errori o cancellazione numerica). Per garantire la stabilità numerica, è necessario permutare le righe in modo da scegliere un pivot "ottimale". Tale strategia è detta *pivoting*.

Strategie di Pivoting

- **Pivoting Parziale:** Al passo k, si sceglie come pivot l'elemento di modulo massimo nella colonna k-esima, al di sotto o sulla diagonale principale. Si cerca l'indice $r \geq k$ tale che:
$$|a_{r,k}^{(k)}| = \max_{k \leq i \leq n} |a_{i,k}^{(k)}|$$
Se $r \neq k$, si scambia la riga k con la riga r.
- **Pivoting Totale:** Si cerca l'elemento massimo su tutta la sottomatrice attiva. Si scelgono indici $r, s \geq k$ tali che:
$$|a_{r,s}^{(k)}| = \max_{k \leq i,j \leq n} |a_{i,j}^{(k)}|$$
Si scambiano le righe k ed r e le colonne k ed s (quest'ultimo scambio comporta il riordinamento delle incognite).

Il pivoting parziale è la strategia più utilizzata perché meno costosa computazionalmente ($O(n^2)$ confronti) e generalmente sufficiente a garantire stabilità.

Stabilità senza Pivoting Il metodo di eliminazione di Gauss senza pivoting è stabile a priori solo per alcune classi di matrici:

- Matrici a diagonale dominante per colonne.
- Matrici a diagonale dominante per righe.
- Matrici simmetriche e definite positive.

Esempio di Instabilità Consideriamo un sistema di ordine $n = 18$ con matrice definita da $a_{i,j} = \cos((j-1)\frac{2i-1}{2n}\pi)$ e termine noto somma degli elementi della riga (soluzione esatta $x_i = 1$). La matrice è ben condizionata ($K \approx 16.9$). Risolvendo con Gauss senza pivoting, si ottengono errori significativi (es. $x_{15} \approx 1.0000000002...$). Risolvendo con Gauss e pivoting parziale, si ottiene la soluzione corretta con massima precisione (es. $x_{15} \approx 1.000000000000000$).

Gauss (senza pivoting)	Gauss + Pivoting Parziale
9.999999968425205e-001	9.999999999999993e-001
1.000000005848771e+000	1.000000000000000e+000
9.999999953310822e-001	1.000000000000001e+000
1.000000003291504e+000	1.000000000000001e+000
9.999999977773912e-001	9.999999999999997e-001
1.000000001703849e+000	1.000000000000000e+000
9.999999983501708e-001	1.000000000000000e+000
1.000000001767299e+000	1.000000000000001e+000
9.999999982508074e-001	9.999999999999998e-001
1.000000001433063e+000	9.999999999999999e-001
9.999999991425489e-001	9.999999999999993e-001
1.000000000200525e+000	1.000000000000000e+000
1.000000000347843e+000	9.999999999999996e-001
9.999999993163441e-001	9.999999999999990e-001
1.000000000800448e+000	9.999999999999996e-001
9.999999992564082e-001	1.000000000000000e+000
1.000000000564552e+000	1.000000000000000e+000
9.999999996959291e-001	9.999999999999999e-001

Schema Algoritmo: Gauss con Pivoting Parziale

```
for k = 1 : n-1
    % Ricerca del pivot massimo nella colonna k-esima
    trovo p tale che: |a(p,k)| = max(|a(i,k)|) per i >= k

    if k ~= p
        scambio righe k-esima e p-esima di A
        scambio righe k-esima e p-esima di b
    end

    % Eliminazione standard di Gauss
    for i = k+1 : n
        mol = -a(i,k) / a(k,k)
        for j = k+1 : n
            a(i,j) = a(i,j) + mol * a(k,j)
        end
        b(i) = b(i) + mol * b(k)
    end
end
```

Costo Computazionale con Pivoting Il costo computazionale totale dell'algoritmo di Gauss con pivoting è dato dalla somma delle operazioni aritmetiche (flops) necessarie per l'eliminazione e dei confronti necessari per la ricerca del pivot massimo.

- **Operazioni Aritmetiche:** $\approx \frac{n^3}{3}$ (invariato rispetto a Gauss senza pivoting).
- **Confronti:** Ad ogni passo k si effettuano $n-k$ confronti. Il totale è:

$$\sum_{k=1}^{n-1}(n-k) = \frac{n(n-1)}{2} \approx \frac{n^2}{2} \text{ confronti}$$

Poiché n^2 è trascurabile rispetto a n^3 per n grandi, il costo asintotico rimane dominato da $n^3/3$.

```
% function [Ag,bg] = gaussElimP(A,b)
% GAUSSELIMP Applica il metodo dell'eliminazione di Gauss alla matrice dei
% coefficienti e al vettore dei termini noti implementando il pivoting
% parziale.
%   [Ag,bg] = gaussElimP(A,b)  restituisce A e b risultanti dall'eliminazione con pivoting
    parziale.
%
%     INPUT:
%       A - Matrice dei coefficienti (n x n)
%       b - Vettore dei termini noti (n x 1)
%
%     OUTPUT:
%       Ag - Matrice dei coefficienti post-GE (n x n)
%       bg - Vettore dei termini noti post-GE (n x 1)

    %% 1. Validazione degli Input
    arguments
        A (:,:) double {mustBeNumeric}
        b (:,1) double {mustBeNumeric}
    end

    %% 2. Verifica Dimensioni
    [n, m] = size(A);

    % Verifico se la matrice è quadrata
    if n ~= m
        error('gaussElimP:nonSquareMatrix', 'La matrice non è quadrata (Size %dx%d).', n, m);
    end

    % Verifico se il prodotto Ab è compatibile
    if n ~= length(b)
```

```
        error('gaussElimP:nonCompatible', 'Il prodotto non è compatibile (Size (%dx%d)*(%dx1)).', n, m, length(b));
    end


    %% 3. Eliminazione di Gauss
    for k = 1 : n-1

        % Pivoting parziale

        % Trovo il massimo nella colonna k (dal pivot in giù)
        [~, r_rel] = max(abs(A(k:n, k)));

        % Converto in indice globale
        r_glob = r_rel + k - 1;

        % Se il pivot migliore non è quello attuale, scambio
        if r_glob ~= k
            A([k, r_glob], :) = A([r_glob, k], :);
            b([k, r_glob])    = b([r_glob, k]);
        end


        % Controllo pivot nullo (sicurezza)
        if A(k,k) == 0
             error('gaussElimP:zeroPivot', 'Pivot nullo alla riga %d.', k);
        end

        % --- ELIMINAZIONE ---
        for i = k+1 : n
            mol = -A(i,k) / A(k,k);

            % Aggiorno la riga i (Vettorizzazione mista)
            A(i, k+1:n) = A(i, k+1:n) + mol * A(k, k+1:n);

            % Aggiorno il termine noto
            b(i) = b(i) + mol * b(k);

            % Pulizia formale (Facoltativo ma consigliato per l'output)
            % A(i,k) = 0;
        end
    end

    Ag = A;
    bg = b;
end
```

Funzione gaussElimP

La Fattorizzazione LU Il metodo di eliminazione di Gauss, dal punto di vista matriciale, può essere riletto come la costruzione di una successione di matrici:

$$[A|\mathbf{b}] = [A^{(1)}|\mathbf{b}^{(1)}], \dots, [A^{(k)}|\mathbf{b}^{(k)}], \dots, [A^{(n)}|\mathbf{b}^{(n)}]$$

in modo tale che $A^{(n)}$ sia triangolare superiore e $\mathbf{b}^{(n)}$ sia il nuovo termine noto. Le matrici della successione sono tra loro legate da una trasformazione del tipo:

$$[A^{(k+1)}|\mathbf{b}^{(k+1)}] = M^{(k)}[A^{(k)}|\mathbf{b}^{(k)}], \quad k \in \{1, \dots, n-1\}$$

dove $M^{(k)}$ è detta **matrice elementare di Gauss** ed è definita come:

$$M^{(k)} = \begin{pmatrix} 1 & \dots & 0 & 0 & \dots & 0 \\ \vdots & \ddots & \vdots & \vdots & & \vdots \\ 0 & \dots & 1 & 0 & \dots & 0 \\ 0 & \dots & m_{k+1,k} & 1 & \dots & 0 \\ \vdots & & \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & m_{n,k} & 0 & \dots & 1 \end{pmatrix}$$

Si ha quindi che:

$$A = [M^{(1)}]^{-1}A^{(2)} = [M^{(1)}]^{-1}[M^{(2)}]^{-1}A^{(3)} = \dots = [M^{(1)}]^{-1} \dots [M^{(n-1)}]^{-1}A^{(n)}$$

e analogamente per il termine noto:

$$\mathbf{b} = [M^{(1)}]^{-1} \dots [M^{(n-1)}]^{-1} \mathbf{b}^{(n)}$$

Ponendo:

$$L = [M^{(1)}]^{-1} \dots [M^{(n-1)}]^{-1}, \quad U = A^{(n)}, \quad \mathbf{y} = \mathbf{b}^{(n)},$$

si ottiene la fattorizzazione:

$$A = LU \quad \text{e} \quad \mathbf{b} = L\mathbf{y}$$

dove U è la matrice triangolare superiore che si ottiene alla fine del metodo di eliminazione di Gauss e L è una matrice triangolare inferiore. La struttura di L è la seguente:

$$L = \begin{pmatrix} 1 & 0 & \dots & & \dots & 0 & \\ -m_{2,1} & 1 & 0 & & & \vdots & \\ \vdots & \ddots & \ddots & \ddots & & & \\ & \ddots & 1 & & & \vdots & \\ \vdots & & -m_{k+1,k} & 1 & \ddots & & \\ & & -m_{k+2,k} & \ddots & \ddots & & \\ \vdots & & \vdots & & \ddots & 1 & 0 \\ -m_{n,1} & \dots & -m_{n,k} & \dots & \dots & -m_{n,n-1} & 1 \end{pmatrix}$$

Per costruirla basta, ad ogni passo del metodo di Gauss, memorizzare i moltiplicatori cambiati di segno.

Schema Algoritmo

```
for k = 1 : n-1
    for i = k+1 : n
        A(i,k) = -A(i,k) / A(k,k)      % Calcolo moltiplicatori
        for j = k+1 : n
            A(i,j) = A(i,j) + A(i,k) * A(k,j)
        end
    end
end
L = eye(n) - tril(A, -1)   % Parte triangolare inferiore con 1 sulla diagonale
U = triu(A)                % Parte triangolare superiore
```

```
function [L, U] = gaussElimLU(A)
%GAUSSELIMLU Esegue la fattorizzazione LU su una matrice A.
%  [L, U] = gaussElimLU(A) decompone la matrice A in L*U usando
%  l'eliminazione di Gauss senza pivoting.
%
%    INPUT:
%      A - Matrice dei coefficienti (n x n)
%
%    OUTPUT:
%      L - Matrice triangolare inferiore unitaria (n x n)
%      U - Matrice triangolare superiore (n x n)

    %% 1. Validazione degli Input
    arguments
        A (:,:) double {mustBeNumeric}
    end

    %% 2. Verifica Dimensioni
    [n, m] = size(A);

    if n ~= m
        error('gaussElimLU:nonSquareMatrix', 'La matrice dei coefficienti deve essere quadrata (Size: %dx%d).', n, m);
    end

    %% 3. Eliminazione di Gauss
```

```
    for k = 1 : n-1

        % Controllo pivot nullo
        if A(k,k) == 0
             error('gaussElimLU:zeroPivot', 'Pivot nullo incontrato alla riga %d.', k);
        end

        % Ciclo sulle righe sottostanti
        for i = k+1 : n

            % Calcolo e assegnazione del moltiplicatore
            A(i,k) = -A(i,k) / A(k,k);

            % Aggiornamento della riga i-esima (Eliminazione)
            A(i, k+1:n) = A(i, k+1:n) + A(i,k) * A(k, k+1:n);
        end
    end

    % Costruzione di L e U
    L = eye(n) - tril(A, -1);
    U = triu(A);
end
```

Funzione gaussElimLU

Risoluzione di Sistemi con LU Una volta fattorizzata $A = LU$, il sistema $Ax = b$ diventa $LUx = b$. Si risolve in due passi:

1. Risoluzione del sistema triangolare inferiore $Ly = b$ (sostituzione in avanti).
2. Risoluzione del sistema triangolare superiore $Ux = y$ (sostituzione all'indietro).

Costo Computazionale Costo totale: $\frac{n^3}{3}$ (fattorizzazione) + n^2 (soluzione sistemi).

Di seguito un esempio in Matlab.

```
format long e;
n = 5;

% Generazione dati casuali
A = randi([-10,10], n, n);
b = randi([-10,10], n, 1);

% 1. Fattorizzazione A = LU
[L, U] = gaussElimLU(A);

% 2. Risoluzione dei sistemi triangolari
y = forwardSub(L, b);        % Risolve Ly = b
x = backwardSub(U, y);       % Risolve Ux = y

% 3. Verifica dell'errore
sol_ref = A\b;               % Soluzione di riferimento MATLAB
errore = norm(x - sol_ref);

disp("Errore assoluto:");
disp(errore);
```

```
Errore assoluto:
     9.160456941832493e-16
```

Applicazioni della Fattorizzazione LU

1. Calcolo del Determinante di A Il determinante di una matrice A può essere calcolato utilizzando la formula di Binet. Avendo la fattorizzazione $A = LU$, si ha:

$$\det(A) = \det(LU) = \det(L)\det(U)$$

Poiché L è triangolare inferiore con tutti 1 sulla diagonale, $\det(L) = 1$. Poiché U è triangolare superiore, il suo determinante è il prodotto degli elementi diagonali. Dunque:

$$\det(A) = \det(U) = \prod_{i=1}^{n} u_{i,i}$$

Il costo computazionale complessivo è $\frac{n^3}{3} + n$ (dove $\frac{n^3}{3}$ è il costo della fattorizzazione).

2. Risoluzione di p sistemi con la stessa matrice dei coefficienti Supponiamo di dover risolvere i sistemi:

$$A\mathbf{x}_1 = \mathbf{b}_1, \quad A\mathbf{x}_2 = \mathbf{b}_2, \quad \ldots, \quad A\mathbf{x}_p = \mathbf{b}_p$$

o in altri termini il sistema matriciale:

$$AX = B$$

con $A \in \mathbb{R}^{n\times n}$ e $X, B \in \mathbb{R}^{n\times p}$. Calcolate le matrici L e U una sola volta, ogni sistema $A\mathbf{x}_i = \mathbf{b}_i$ viene risolto risolvendo i due sistemi triangolari:

$$\begin{cases} L\mathbf{y} = \mathbf{b}_i \\ U\mathbf{text}_i = \mathbf{y} \end{cases}, \quad i \in \{1, \ldots, p\}$$

mediante gli algoritmi di sostituzione. Il costo computazionale complessivo è:

$$\frac{n^3}{3} + pn^2$$

Se invece si risolvesse ciascun sistema indipendentemente dagli altri con il metodo di Gauss, il costo sarebbe molto più alto: $p(\frac{n^3}{3} + \frac{n^2}{2})$.

3. Calcolo dell'inversa di A Se $p = n$ e $B = I$ (matrice identità), risolvere il sistema $AX = B$ è equivalente a calcolare l'inversa A^{-1}. Si risolvono quindi i sistemi:

$$\begin{cases} L\mathbf{y} = \mathbf{e}_i \\ U\mathbf{x}_i = \mathbf{y} \end{cases}, \quad i \in \{1, \ldots, n\}$$

dove i vettori $\mathbf{e}_i$ rappresentano le colonne della matrice I (vettori della base canonica). Il costo computazionale complessivo "grezzo" sarebbe $\frac{n^3}{3} + n^3$. Tuttavia, sfruttando opportunamente la natura dei vettori $\mathbf{e}_i$ (che contengono molti zeri), il costo si riduce a circa n^3. Nel dettaglio:

- Risolvere il sistema matriciale $LY = I$ ha un costo di $\frac{n^3}{6}$.
- Risolvere il sistema matriciale $UX = Y$ ha un costo di $\frac{n^3}{2}$.

Sommando il costo della fattorizzazione ($\frac{n^3}{3}$), il costo totale asintotico è n^3.

Il metodo di Gauss con la variante del pivoting Il metodo di Gauss con la variante del pivoting esegue ancora una fattorizzazione di matrice nei seguenti termini:

$$PA = \hat{L}U \quad \text{e} \quad Pb = \hat{L}\mathbf{y}$$

dove $P \in \mathbb{R}^{n\times n}$, detta matrice di permutazione, contiene le informazioni relative agli scambi di righe. Vale il seguente:

Teorema 3.5. *Per ogni matrice $A \in \mathbb{R}^{n\times n}$ esiste una matrice di permutazione $P \in \mathbb{R}^{n\times n}$ tale che*

$$PA = \hat{L}U.$$

Le matrici di permutazione Siano $1 \leq r, s \leq n$ e sia $P_{r,s}$ la matrice ottenuta da I scambiando la r-esima e la s-esima riga.

$$P_{r,s} = \begin{pmatrix} 1 & 0 & \dots & \dots & \dots & 0 \\ 0 & 1 & 0 & \dots & \dots & 0 \\ \vdots & & 0 & \dots & 1 & \vdots \\ \vdots & & \vdots & 1 & \vdots & \vdots \\ \vdots & & 1 & \dots & 0 & \vdots \\ 0 & \dots & \dots & \dots & 0 & 1 \end{pmatrix} \begin{matrix} \\ \\ \leftarrow r \\ \vdots \\ \leftarrow s \\ \\ \end{matrix}$$

Proposizione 3.6. *La matrice $P_{r,s}$ è detta matrice di permutazione e gode delle seguenti proprietà:*

1. $det(P_{r,s}) = -1$
2. $P_{r,s} = P_{r,s}^T$
3. $P_{r,s}^2 = I$
4. $P_{r,s}^{-1} = P_{r,s}$

Proof. 1. Banale;

2. Segue dal fatto che l'elemento di posto (r, s) coincide con quello di posto (s, r), mentre tutti gli altri elementi non nulli giacciono sulla diagonale;

3. $$P_{r,s}P_{r,s} = (\mathbf{e}_1, \dots, \mathbf{e}_{r-1}, \mathbf{e}_s, \mathbf{e}_{r+1}, \dots, \mathbf{e}_{s-1}, \mathbf{e}_r, \mathbf{e}_{s+1}, \dots, \mathbf{e}_n)^T(\mathbf{e}_1, \dots, \mathbf{e}_s, \dots, \mathbf{e}_r, \dots, \mathbf{e}_n)$$
$$= \mathbf{e}_1\mathbf{e}_1^T + \dots + \mathbf{e}_{r-1}\mathbf{e}_{r-1}^T + \mathbf{e}_s\mathbf{e}_s^T + \mathbf{e}_{r+1}\mathbf{e}_{r+1}^T + \dots + \mathbf{e}_{s-1}\mathbf{e}_{s-1}^T + \mathbf{e}_r\mathbf{e}_r^T + \mathbf{e}_{s+1}\mathbf{e}_{s+1}^T + \dots + \mathbf{e}_n\mathbf{e}_n^T = I$$
dove $\mathbf{e}_i$ è l'i-esimo vettore colonna della base canonica;

4. Segue dalla 3.

□

Osservazione 1. Si vede inoltre che:

- il prodotto $P_{r,s}A$ produce lo scambio delle righe r-esima ed s-esima in A.
- il prodotto $AP_{r,s}$ produce lo scambio delle colonne r-esima ed s-esima in A.

Fattorizzazione LU con pivoting I passi della fattorizzazione LU senza pivoting della matrice A possono essere sintetizzati come segue:

- Se $a_{1,1} \neq 0$: annullamento elementi prima colonna a partire dalla seconda riga della matrice iniziale, ottenuto dal prodotto $M^{(1)}A$.
- Se $a_{2,2}^{(2)} \neq 0$: annullamento elementi della seconda colonna a partire dalla terza riga della matrice attiva, ottenuto dal prodotto $M^{(2)}M^{(1)}A$.
- ...
- Se $a_{n-1,n-1}^{(n-1)} \neq 0$: annullamento elementi della $(n-1)$-esima colonna a partire dalla n-esima riga della matrice attiva, ottenuto dal prodotto $M^{(n-1)} \dots M^{(2)}M^{(1)}A$.

Al termine si ha $M^{(n-1)} \dots M^{(2)}M^{(1)}A = U$ triangolare superiore. Possiamo descrivere la tecnica del pivoting usando le matrici di permutazione, nel seguente modo:

- Sia $|a_{r_1,1}| = \max_{i\geq 1} |a_{i,1}|$. Se $r_1 \neq 1$ si pone $P_1 = P_{1,r_1}$ e si esegue lo scambio della prima riga con la r_1-esima con il prodotto P_1A. Se $r_1 = 1$ si pone $P_1 = I$ e non si effettuano scambi. Si esegue l'annullamento degli elementi della prima colonna a partire dalla seconda riga della matrice P_1A con il prodotto $M^{(1)}P_1A$.
- Sia $|a^{(2)}_{r_2,2}| = \max_{i\geq 2} |a^{(2)}_{i,2}|$. Se $r_2 \neq 2$ si pone $P_2 = P_{2,r_2}$ e si esegue lo scambio della seconda riga con la r_2-esima della matrice $M^{(1)}P_1A$ con il prodotto $P_2M^{(1)}P_1A$. Se $r_2 = 2$ si pone $P_2 = I$. Si esegue l'annullamento degli elementi della seconda colonna a partire dalla terza riga della matrice $P_2M^{(1)}P_1A$ con il prodotto $M^{(2)}P_2M^{(1)}P_1A$.
- ...

Al termine della procedura si ottiene:

$$M^{(n-1)}P_{n-1}\dots M^{(2)}P_2M^{(1)}P_1A = U$$

ovvero

$$GA = U, \quad G = M^{(n-1)}P_{n-1}\dots M^{(2)}P_2M^{(1)}P_1.$$

Vogliamo ora mostrare come dalla $GA = U$ si perviene alla seguente fattorizzazione della matrice A:

$$PA = \hat{L}U$$

ove P è un'opportuna matrice di permutazione, che tiene conto di tutti gli scambi di righe, $\hat{L}$ è una matrice triangolare inferiore con elementi sulla diagonale uguali a 1 e U è la matrice triangolare superiore del metodo di eliminazione di Gauss. Ovviamente, in assenza di scambi $P = I$. Si possono infatti riordinare le matrici in modo da ottenere:

$$M^{(n-1)}P_{n-1}\dots M^{(2)}P_2M^{(1)}P_1A = (\overline{L}_{n-1}\dots\overline{L}_2\overline{L}_1)^{-1}(P_{n-1}\dots P_2P_1)A$$

e, posto $\hat{L}^{-1} = \overline{L}_{n-1}\dots\overline{L}_2\overline{L}_1$ e $P = P_{n-1}\dots P_2P_1$, possiamo scrivere:

$$\hat{L}^{-1}PA = U$$

da cui

$$PA = \hat{L}U.$$

Esempio 3.6. Esaminiamo il caso particolare $n = 4$. Si ha:

$$M^{(3)}P_3M^{(2)}P_2M^{(1)}P_1A = M^{(3)}P_3M^{(2)}P_3P_3P_2M^{(1)}P_2P_2P_1A$$

Posto $L^{(1)}_1 = P_2M^{(1)}P_2$:

$$= M^{(3)}P_3M^{(2)}P_3P_3L^{(1)}_1P_2P_1A$$

Posto $L^{(1)}_2 = P_3M^{(2)}P_3$:

$$= M^{(3)}L^{(1)}_2P_3L^{(1)}_1P_3P_3P_2P_1A$$

Posto $L^{(2)}_1 = P_3L^{(1)}_1P_3$:

$$= M^{(3)}L^{(1)}_2L^{(2)}_1P_3P_2P_1A$$

In generale (si può provare per induzione):

$$M^{(n-1)}P_{n-1}M^{(n-2)}P_{n-2}\dots M^{(2)}P_2M^{(1)}P_1A = \overline{L}_{n-1}\dots\overline{L}_2\overline{L}_1P_{n-1}\dots P_2P_1A$$

con

$$\overline{L}_j = \begin{cases} L^{(n-1-j)}_j = (P_{n-1}P_{n-2}\dots P_{j+1})M^{(j)}(P_{j+1}\dots P_{n-2}P_{n-1}), & j < n-1 \\ M^{(j)}, & j = n-1 \end{cases}$$

Osservazione 2. Le matrici $M^{(i)}$ e $\overline{L}_i$ (e quindi anche L e $\hat{L}$) differiscono solo per un diverso ordinamento (nell'ambito della stessa colonna e sotto l'elemento diagonale) dei moltiplicatori $m_{i,j}$, in conseguenza degli scambi di riga effettuati in precedenza. Al termine del processo di eliminazione al posto della matrice iniziale A avremo:

$$\begin{pmatrix} u_{11} & u_{12} & u_{13} & \dots & u_{1n} \\ m_{21} & u_{22} & u_{23} & \dots & u_{2n} \\ m_{31} & m_{32} & u_{33} & \dots & u_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ m_{n1} & m_{n2} & m_{n3} & \dots & u_{nn} \end{pmatrix}$$

Osservazione 3. La costruzione della matrice P viene fatta scambiando, ogni volta che viene fatto uno scambio di righe della matrice A, le righe corrispondenti della matrice I.

Schema Algoritmo

```
for k = 1:n-1
    trovo p : |a(p,k)| = max(|a(i,k)|) per i >= k
    if k ~= p
        scambio righe k-esima e p-esima di A
        scambio righe k-esima e p-esima di I
    end
    for i = k+1:n
        a(i,k) = a(i,k) / a(k,k)
        for j = k+1:n
            a(i,j) = a(i,j) - a(i,k) * a(k,j)
        end
    end
end
L = eye(n) + tril(A, -1)
U = triu(A)
```

Costo computazionale

$$\frac{n^3}{3} \text{ operazioni} + \sum_{k=1}^{n-1}(n-k) \text{ confronti} \simeq \frac{n^3}{3} \text{ operazioni} + \frac{n^2}{2} \text{ confronti}$$

Risoluzione di sistemi lineari mediante LU con pivoting Nota la fattorizzazione del tipo $PA = \hat{L}U$, per determinare la soluzione del sistema lineare $Ax = b$ si risolvono due sistemi triangolari:

$$\begin{cases} \hat{L}y = Pb \\ Ux = y \end{cases}$$

che deduciamo dalla relazione $PAx = Pb$, ovvero $\hat{L}Ux = Pb$.

Costo computazionale L'intera procedura (fattorizzazione $PA = \hat{L}U$ e risoluzione dei due sistemi triangolari) consta di $\frac{n^3}{3} + n^2$ operazioni aritmetiche. Ci sono poi $(n-1) + (n-2) + \cdots + 1 = \frac{n(n-1)}{2}$ confronti per determinare il massimo pivot in ogni passo.

Di seguito un esempio in Matlab.

```
format long e;
n = 5;

% Generazione dati casuali
A = randi([-10,10], n, n);
b = randi([-10,10], n, 1);

% 1. Fattorizzazione PA = LU
[L, U, P] = gaussElimLUP(A);

% 2. Permutazione del termine noto
b_perm = P * b;

% 3. Risoluzione dei sistemi triangolari
y = forwardSub(L, b_perm);   % Risolve Ly = Pb
x = backwardSub(U, y);       % Risolve Ux = y

% 4. Verifica dell'errore
sol_ref = A\b;               % Soluzione di riferimento MATLAB
```

```
errore = norm(x - sol_ref);

disp("Errore assoluto:");
disp(errore);
```

```
Errore assoluto:
     2.220446049250313e-16
```

Facciamo ora un confronto tra i due metodi aumentando le dimensioni del problema.

```
format long e;
n = 100;

% Generazione dati casuali
A = randi([-10,10], n, n);
b = randi([-10,10], n, 1);

% 1. Fattorizzazione A = LU e PA = LU
[L0, U0] = gaussElimLU(A);
[L1, U1, P] = gaussElimLUP(A);

% 2. Permutazione del termine noto
b_perm = P * b;

% 3. Risoluzione dei sistemi triangolari
y0 = forwardSub(L0, b);          % Risolve Ly = b
x0 = backwardSub(U0, y0);        % Risolve Ux = y

y1 = forwardSub(L1, b_perm);     % Risolve Ly = Pb
x1 = backwardSub(U1, y1);        % Risolve Ux = y

% 4. Verifica dell'errore
sol_ref = A\b;                   % Soluzione di riferimento MATLAB
errore0 = norm(x0 - sol_ref);
errore1 = norm(x1 - sol_ref);

disp("Errore assoluto LU:");
disp(errore0);

disp("Errore assoluto LUP:");
disp(errore1);
```

```
Errore assoluto LU:
     6.487091132616242e-12

Errore assoluto LUP:
     3.176154547748571e-14
```

Calcolo del determinante Da $PA = \hat{L}U$, essendo $P^{-1} = P^T$,

$$\det(A) = \det(P^T)\det(\hat{L})\det(U) = \det(P)\det(U)$$

Poiché

$$\det(P) = \det(P_{n-1})\dots\det(P_2)\det(P_1),$$

e

$$\det(P_j) = \begin{cases} 1 & \text{se } P_j = I \\ -1 & \text{altrimenti} \end{cases}$$

si ha

$$\det(A) = (-1)^s \prod_{i=1}^{n} a_{i,i}^{(i)}$$

dove s denota il numero complessivo di scambi effettuati.

Calcolo dell'inversa di A Denotate con $\mathbf{x}_1, \ldots, \mathbf{x}_n$ le colonne della matrice inversa di A, quest'ultima può essere calcolata risolvendo i $2n$ sistemi:

$$\begin{cases} \hat{L}\mathbf{y} = P\mathbf{e}_i \\ U\mathbf{x}_i = \mathbf{y} \end{cases} \qquad i \in \{1, \ldots, n\}$$

il primo triangolare inferiore e il secondo triangolare superiore. Il costo computazionale complessivo è $\frac{n^3}{3} + n^3$ ma può essere ridotto a n^3 se si tiene conto dalla natura dei vettori $P\mathbf{e}_i$.

3.5 Metodo di Cholesky

Fattorizzazione LDU Sia $A \in \mathbb{R}^{n\times n}$ una matrice per cui esiste la fattorizzazione LU di A senza pivoting. Questa decomposizione può essere sempre riscritta come:

$$A = LU = LDU_1$$

essendo L, U_1 entrambe matrici triangolari a diagonale unitaria e D matrice diagonale tale che $(D)_{i,i} = (U)_{i,i}$. Se A è simmetrica, allora $U_1 = L^T$, e si ha:

$$A = LDL^T$$

Fattorizzazione LL^T Se $A \in \mathbb{R}^{n\times n}$ è simmetrica definita positiva, essendo gli elementi di D tutti positivi, si ha:

$$D = D^{\frac{1}{2}} D^{\frac{1}{2}}$$

dove gli elementi di $(D^{\frac{1}{2}})_{i,i} = \sqrt{D_{i,i}}$. Da cui segue il teorema:

Teorema 3.7. *Se $A \in \mathbb{R}^{n\times n}$ è una matrice simmetrica definita positiva esiste ed è unica la fattorizzazione:*

$$A = L_1 L_1^T \quad con \quad L_1 = LD^{\frac{1}{2}}$$

dove L_1 è una matrice triangolare inferiore con elementi diagonali non nulli.

Nel seguito si fa riferimento alla fattorizzazione di cui nel teorema precedente come alla fattorizzazione del tipo LL^T. La fattorizzazione LL^T può essere ottenuta attraverso la procedura di fattorizzazione LU senza pivoting con un costo pari a $n^3/6$, cioè circa la metà della fattorizzazione LU. Infatti è possibile organizzare l'algoritmo in modo da sfruttare la simmetria di tutte le matrici attive negli $n-1$ passi. Vedremo ora come sia possibile ottenere la stessa fattorizzazione attraverso una tecnica "compatta" che porta al metodo di Cholesky e che costa circa $n^3/6$ operazioni.

Costruzione del Metodo di Cholesky Posto $A = (a_{i,j})_{i,j\in\{1,\ldots,n\}}$ e $L = (l_{i,j})_{i,j\in\{1,\ldots,n\}}$, si ha:

$$a_{i,j} = \sum_{k=1}^{n} l_{i,k} l_{k,j}^T = \sum_{k=1}^{n} l_{i,k} l_{j,k}$$

Poiché la matrice è simmetrica, possiamo considerare solo gli elementi di A con $j \leq i$. Otteniamo:

$$a_{i,j} = \sum_{k=1}^{j-1} l_{i,k} l_{j,k} + l_{i,j} l_{j,j}, \quad i \in \{1, \ldots, n\}, \quad j \in \{1, \ldots, i-1\}$$

$$a_{i,i} = \sum_{k=1}^{i-1} l_{i,k}^2 + l_{i,i}^2, \quad i \in \{1, \ldots, n\}$$

Da cui si ricavano le formule per gli elementi di L:

$$l_{1,1} = \sqrt{a_{1,1}}$$

$$l_{i,j} = \frac{1}{l_{j,j}} \left(a_{i,j} - \sum_{k=1}^{j-1} l_{i,k} l_{j,k} \right), \quad i \in \{2, \dots, n\}, \quad j \in \{1, \dots, i-1\}$$

$$l_{i,i} = \sqrt{a_{i,i} - \sum_{k=1}^{i-1} l_{i,k}^2}, \quad i \in \{2, \dots, n\}.$$

È lecito assumere che l'operazione di radice quadrata sia equivalente (in termini di tempo) alle operazioni di prodotto e divisione.

Schema Algoritmo

```
l(1,1) = sqrt(a(1,1))                    % 1 operazione
for i = 2:n
    for j = 1:i-1
        l(i,j) = 0;
        for k = 1:j-1
            l(i,j) = l(i,j) + l(i,k) * l(j,k)
        end
        l(i,j) = (a(i,j) - l(i,j)) / l(j,j)    % j-1 operazioni + 1 op.
    end
    l(i,i) = 0;
    for k = 1:i-1
        l(i,i) = l(i,i) + l(i,k)^2
    end
    l(i,i) = sqrt(a(i,i) - l(i,i))             % i-1 operazioni + 1 op.
end
```

Costo Computazionale

$$1 + \sum_{i=2}^{n} \left[i + \sum_{j=1}^{i-1} j \right] = \sum_{i=1}^{n} i + \sum_{i=1}^{n} \frac{i(i-1)}{2}$$

$$= \sum_{i=1}^{n} i + \frac{1}{2} \sum_{i=1}^{n} i^2 - \frac{1}{2} \sum_{i=1}^{n} i = \frac{1}{2} \sum_{i=1}^{n} i + \frac{1}{2} \sum_{i=1}^{n} i^2$$

$$= \frac{n(n+1)}{4} + \frac{n(n+1)(2n+1)}{12} \simeq \frac{n^3}{6}$$

```
function L = cholDec(A)
%CHOLDEC Esegue la fattorizzazione di Cholesky.
%  L = cholDec(A) decompone la matrice A in L*L'
%  La matrice A deve essere Simmetrica Definita Positiva.
%
%    INPUT:
%      A - Matrice dei coefficienti (n x n)
%
%    OUTPUT:
%      L - Matrice fattorizzata (n x n)

    %% 1. Validazione degli Input
    arguments
        A (:,:) double {mustBeNumeric}
    end

    %% 2. Verifica Dimensioni
    [n, m] = size(A);

    if n ~= m
        error('cholDec:nonSquareMatrix', 'La matrice dei coefficienti deve essere quadrata (
    Size: %dx%d).', n, m);
    end

    %% 3. Verifica Simmetria
    if ~isequal(A, A')
        error('cholDec:nonSymmMatrix','La matrice dei coefficienti deve essere simmetrica.')
```

```
    end

    %% 2. Metodo di Cholesky
    L = zeros(n);

    % Primo elemento
    L(1,1) = sqrt(A(1,1));

    for i = 2:n
        % Calcolo elementi fuori diagonale
        for j = 1:i-1
            L(i,j) = 0;     % Inizializza accumulatore

            for k = 1:j-1
                L(i,j) = L(i,j) + L(i,k) * L(j,k);
            end

            L(i,j) = (A(i,j) - L(i,j)) / L(j,j);
        end

        % Calcolo elemento diagonale
        L(i,i) = 0;     % Inizializza accumulatore
        for k = 1:i-1
            L(i,i) = L(i,i) + L(i,k)^2;
        end

        % Radice quadrata finale per la diagonale
        argRad = A(i,i) - L(i,i);

        % Controllo di sicurezza
        if argRad <= 0
            error('cholDec:nonDefPosMatrix','La matrice dei coefficienti non è definita
    positiva (elemento diagonale <= 0).')
        end

        L(i,i) = sqrt(argRad);
    end
end
```

Stabilità e Risoluzione È possibile provare che l'algoritmo di Cholesky è stabile. Ricordiamo che, peraltro, anche l'algoritmo di Gauss, senza pivoting, è stabile per le matrici simmetriche definite positive. Molte routine automatiche calcolano $R = L^T$ anziché L, ma è evidente che l'algoritmo è lo stesso (con un attento uso degli indici), data la simmetria della matrice di partenza A. In tali casi scriveremo:

$$A = R^T R$$

Osserviamo infine che se si vuole risolvere il sistema $Ax = b$ mediante la fattorizzazione di Cholesky basterà, una volta computata la fattorizzazione, e quindi calcolata L (o rispettivamente R), risolvere i seguenti due sistemi:

$$Ly = b, \quad L^T x = y$$

(oppure $R^T y = b, \quad Rx = y$ rispettivamente).

Di seguito un esempio in Matlab.

```
format long e;
n = 5;

% Generazione dati casuali
A = randi([-10,10], n, n);
A = A + A';              % Rendo A simmetrica
A = A * A' + eye(n);     % Rendo A definita positiva
b = randi([-10,10], n, 1);

% 1. Decomposizione Cholesky
L = cholDec(A);

% 2. Risoluzione dei sistemi triangolari
y = forwardSub(L, b);          % Risolve Ly = b
x = backwardSub(L', y);        % Risolve L'x = y

```

```
% 3. Verifica dell'errore
sol_ref = A\b;                % Soluzione di riferimento MATLAB
errore = norm(x - sol_ref);

disp("Errore assoluto:");
disp(errore);
```

```
Errore assoluto:
     2.220446049250313e-16
```

Le function Matlab Chiudiamo questa carrellata dei principali metodi diretti per la risoluzione di un sistema lineare descrivendo le principali function di Matlab che implementano tali metodi. Sia allora A una matrice non singolare di ordine n e b un vettore colonna di ordine n.

- **Risoluzione dei sistemi diagonali:** Se la matrice dei coefficienti A è diagonale, `A\b` risolve il sistema mediante n operazioni di divisione.
- **Risoluzione dei sistemi triangolari:** Se la matrice dei coefficienti A è triangolare (superiore o inferiore), `A\b` risolve il sistema mediante algoritmo di sostituzione (all'indietro o in avanti a seconda della struttura della matrice).
- **Risoluzione di sistemi con matrice qualsiasi:** `A\b` risolve il sistema mediante metodo di eliminazione di Gauss (in realtà con la fattorizzazione LU) con pivoting e gli algoritmi di sostituzione.
- **Risoluzione di sistemi simmetrici def. positivi:** `A\b` risolve il sistema mediante metodo di Cholesky e gli algoritmi di sostituzione.
- **Fattorizzazione LU:** Il comando `[L,U,P] = lu(A)` calcola i fattori L, U e la matrice di permutazione P tali che $PA = LU$.
- **Fattorizzazione di Cholesky:** Se la matrice A è simmetrica e definita positiva, il comando `R = chol(A)` calcola il fattore triangolare superiore R tale che $A = R^T R$.
- **Calcolo del determinante:** `det(A)` calcola il determinante della matrice non singolare A, utilizzando la fattorizzazione LU.
- **Condizionamento:** Il comando `cond(A, p)`, con `p = 1, 2, inf, 'fro'`, calcola il numero di condizionamento rispettivamente in norma 1, 2, infinito e Frobenius.
- **Inversa di una matrice:** `inv(A)` calcola l'inversa della matrice utilizzando la fattorizzazione LU.

3.6 Stabilità dei Metodi di Gauss

Backward Error Analysis Con la "Backward Error Analysis (BEA)" (Analisi dell'errore all'indietro), si misura la perturbazione nei dati di ingresso che porta al risultato finale, ipotizzando che le operazioni siano eseguite con precisione infinita. Si valuta in sostanza quanto è perturbato il dato di input considerando "esatto" il risultato in output. Essa è stata introdotta da Wilkinson nel 1963 con l'obiettivo di studiare gli errori negli algoritmi di Algebra Lineare ed è particolarmente adatta quando il numero di operazioni è elevato. L'algoritmo è detto *stabile* se la perturbazione nel dato di ingresso è dell'ordine della precisione in cui si lavora.

Backward Error Analysis nella fattorizzazione LU

Teorema 3.8. *Sia A una matrice di ordine n i cui elementi sono numeri di macchina. Siano L ed U le matrici della fattorizzazione ottenute senza la tecnica del pivoting. Esiste allora una matrice E per cui:*

$$(A + E) = LU$$

con E tale che:

$$||E|| \leq nu||L||||U||$$

essendo $u = \frac{eps}{2}$.

Poiché $U = A^{(n-1)}$ la stabilità del metodo di eliminazione di Gauss può essere meglio compresa misurando la crescita degli elementi delle matrici $A^{(k)}$ e, quindi, misurando il fattore di crescita.

Fattore di Crescita

Definizione 3.10. Si definisce fattore di crescita la seguente quantità:

$$\rho = \frac{\max(\alpha_1, \alpha_2, \ldots, \alpha_n)}{\alpha_1}$$

con:

$$\alpha_1 = \max_{1 \leq i,j \leq n} |a_{i,j}|$$

$$\alpha_k = \max_{1 \leq i,j \leq n} |a_{i,j}^{(k)}|, \quad k = 2, \ldots, n$$

essendo $A^{(k)} = (a_{i,j}^{(k)})_{i,j=1,\ldots,n}$ la matrice ottenuta al termine del passo k-esimo della fattorizzazione LU.

La definizione vale sia con pivoting che senza pivoting. Osserviamo che, sebbene il pivoting faccia in modo che i moltiplicatori siano tutti più piccoli di 1, gli elementi delle matrici $A^{(k)}$ possono ancora crescere arbitrariamente.

BEA nella fattorizzazione LU con pivoting

Teorema 3.9. *Siano L, U le matrici della fattorizzazione con pivoting parziale. Si ha:*

$$P(A + E) = LU$$

con E tale che:

$$||E|| \leq n^3 u \rho ||A||_\infty$$

essendo $u = \frac{eps}{2}$.

Ci domandiamo: quanto grande può essere ρ?

Esempio senza pivoting Consideriamo il sistema lineare:

$$\begin{cases} \varepsilon x_1 + x_2 = 1 + \varepsilon \\ x_1 = 1 \end{cases}$$

Se scegliamo $\varepsilon = 0.5 \cdot 10^{-5}$ si ha:

$$L = \begin{pmatrix} 1 & 0 \\ 2 \cdot 10^5 & 1 \end{pmatrix}, \quad U = \begin{pmatrix} 0.5 \cdot 10^{-5} & 1 \\ 0 & -2 \cdot 10^5 \end{pmatrix}$$

Poiché $||L||_\infty = (1 + 2 \cdot 10^5)$ e $||U||_\infty = 2 \cdot 10^5$, per il Teorema 1 si ha:

$$||E||_\infty \leq nu||L||_\infty||U||_\infty = \text{eps}(1 + 2 \cdot 10^5)2 \cdot 10^5 \sim 0.9 \cdot 10^{-5} < 0.5 \cdot 10^{-4}$$

Inoltre il fattore di crescita è:

$$\rho = \frac{\max(1, 2 \cdot 10^5)}{1} = 2 \cdot 10^5$$

Esempio con pivoting Applicando il pivoting si ha:

$$L = \begin{pmatrix} 1 & 0 \\ 5.00 \cdot 10^{-6} & 1 \end{pmatrix}, \quad U = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

Poiché il fattore di crescita è $\rho = 1$ e $||A||_\infty = 1 + 0.5 \cdot 10^{-5}$, per il Teorema 2 si ha:

$$||E||_\infty \leq 8 \cdot u \cdot 1 \cdot (1 + 0.5 \cdot 10^{-5}) \sim 0.9 \cdot 10^{-15} < 0.5 \cdot 10^{-14}$$

Crescita del fattore ρ nel pivoting

Teorema 3.10. *Per la fattorizzazione LU con pivoting parziale (e per l'eliminazione di Gauss con pivoting), per ogni matrice A di ordine n si ha:*

$$\rho \leq 2^{n-1}$$

Sfortunatamente esistono matrici che hanno fattore di crescita pari a 2^{n-1}. Una di esse è la matrice di Wilkinson:

$$a_{ij} = \begin{cases} 1, & i = j \\ -1, & i > j \\ 1, & j = n \\ 0 & \text{altrove} \end{cases}$$

Il pivoting totale Si sceglie una coppia (r, s), con $r, s \geq k$ tale che:

$$|a_{rs}^{(k)}| = \max_{k \leq i,j \leq n} |a_{ij}^{(k)}|$$

e si scambia l'equazione k-esima con la r-esima e l'incognita k-esima (con il suo coefficiente) con la s-esima. Nella fattorizzazione LU con pivoting totale si costruiscono due matrici di permutazione P e Q tali che:

$$PAQ = LU$$

Il sistema lineare $Ax = b$ diventa equivalente a ($Q^{-1} = Q^T$):

$$PAQQ^T x = Pb$$

Posto $Q^T x = y$ si ha $PAQy = Pb$ ed usando la fattorizzazione LU si ha $LUy = Pb$. Posto $Uy = z$ si calcola il vettore soluzione di $Lz = Pb$ e poi il vettore soluzione di $Uy = z$. Infine, si calcola la soluzione x del sistema iniziale $x = Qy$. Nella fattorizzazione LU con pivoting totale il fattore di crescita ρ cresce molto lentamente, si prova infatti che:

$$\rho \leq (n \cdot 2^1 \cdot 3^{\frac{1}{2}} \cdot 4^{\frac{1}{3}} \cdots n^{\frac{1}{n-1}})^{\frac{1}{2}}$$

Conclusioni sulla stabilità Il metodo di eliminazione di Gauss senza pivoting ha fattore di crescita pari a 1 per matrici simmetriche definite positive. Il metodo di eliminazione di Gauss senza pivoting ha fattore di crescita ≤ 2 per matrici a diagonale dominante per righe o per colonne. Da cui segue la stabilità del metodo di Gauss senza pivoting quando viene applicato a matrici simmetriche definite positive o a diagonale dominante per righe o per colonne.

Tabella riassuntiva:

- **GE** (o $A = LU$): non stabile in generale (stabile se la matrice è simmetrica definita positiva oppure a diagonale dominante).
- **GEPP** (o $PA = LU$): quasi sempre stabile.
- **GETP** (o $PAQ = LU$): stabile.

(GE: Gaussian Elimination, GEPP: Partial Pivoting, GETP: Total Pivoting)

Residuo e sua valutazione In riferimento ai sistemi lineari quadrati, sia x la soluzione esatta ed x^* la soluzione calcolata. Dimostriamo che vale la seguente stima dell'errore relativo:

$$\frac{||x - x^*||}{||x||} \leq K(A) \frac{||Ax^* - b||}{||b||}$$

Dimostrazione. Si ha:

$$||x - x^*|| = ||A^{-1}(Ax - Ax^*)|| \leq ||A^{-1}||||b - Ax^*|| = ||A^{-1}||||b|| \frac{||b - Ax^*||}{||b||}$$

Inoltre, poiché $||b|| \leq ||A||||x||$, si ottiene:

$$\frac{||x - x^*||}{||x||} \leq ||A||||A^{-1}|| \frac{||Ax^* - b||}{||b||}$$

Dalla stima precedente si deduce che la norma del residuo $||Ax^* - b||$, pur essendo piccola (magari dell'ordine di eps), non garantisce un errore relativo piccolo, in particolare per matrici con elevato indice di condizionamento. Quindi la quantità $||Ax^* - b||$ deve essere usata con cautela come stimatore a posteriori dell'errore.

4 Metodi Numerici per la Risoluzione di un Sistema Lineare Rettangolare nel Senso dei Minimi Quadrati

Il problema dei minimi quadrati lineari Consideriamo un sistema lineare del tipo

$$Ax = b$$

dove A è una matrice di ordine $m \times n$, con $m \neq n$, x è un vettore di lunghezza n e b un vettore di lunghezza m. Molte situazioni reali, per esempio nelle applicazioni statistiche, nei problemi di teoria dei segnali, in astronomia, possono essere modellizzate mediante un sistema lineare nel quale la matrice dei coefficienti è rettangolare (o quadrata ma singolare).

In tali casi la soluzione può non esistere affatto oppure possono esserci infinite soluzioni.

- Se $m > n$, se cioè il numero delle equazioni è superiore a quello delle incognite, il sistema è detto *sovradeterminato* e non ammette soluzioni.
- Se invece $m < n$ il sistema è detto *sottodeterminato* e ammette infinite soluzioni.

In questi casi, invece di cercare di determinare il vettore x che verifichi l'identità $Ax = b$, si cerca, se esiste, un vettore x tale che la distanza di Ax da b sia la minima possibile. In altri termini il problema viene ricondotto ad un problema di minimo: determinare un $\overline{x}$ di lunghezza n tale che, posto $r(x) = Ax - b$ (detto residuo) risulti

$$||r(\overline{x})|| = \min_x ||r(x)|| = \min_x ||Ax - b||$$

Se la norma utilizzata è quella euclidea, ovvero la norma 2,

$$||x||_2 = \sqrt{\sum_{i=1}^{n} x_i^2} = \sqrt{x^T x}$$

il problema è detto *problema dei minimi quadrati lineari* (PMQ).

Il nome è dovuto al fatto che si cerca il minimo della quantità

$$\min_{x=(x_1,\ldots,x_n)} ||Ax - b||_2 = \min_{x=(x_1,\ldots,x_n)} \sqrt{\sum_{i=1}^{m} \left[b_i - \sum_{j=1}^{n} a_{ij} x_j \right]^2}$$

cioè si cerca di minimizzare una somma di quadrati.

Anche il problema dei minimi quadrati può avere infinite soluzioni. In questo caso si definisce *soluzione di minima norma* quella per cui accade che

$$||x_{min}||_2 \leq ||x||_2$$

se x è una qualsiasi soluzione.

D'ora in avanti ci occuperemo solo del caso dei sistemi sovradeterminati, cioè con $m \geq n$, che è anche il più significativo dal punto di vista delle applicazioni. Una delle più significative applicazioni del problema dei minimi quadrati sovradeterminati è quella relativa all'approssimazione di dati, detta anche *best fitting*.

Risolubilità del problema dei minimi quadrati Vediamo ora in quali ipotesi il problema dei minimi quadrati ammette soluzione.

Teorema 4.1. *Se A è una matrice $m \times n$ con $m > n$ allora esiste sempre una soluzione del PMQ. Inoltre tale soluzione è unica se e solo se il rango di A è massimo, cioè se rank$(A) = n$.*

Omettiamo la dimostrazione di tale teorema mettendo però in evidenza che tale dimostrazione è basata sulla seguente equivalenza:

$$x \text{ è soluzione del PMQ} \iff x \text{ è soluzione del sistema } A^T Ax = A^T b$$

Il sistema delle equazioni normali Il sistema

$$A^T Ax = A^T b$$

è detto *sistema delle equazioni normali*. Osserviamo innanzitutto che la matrice $A^T A$ è una matrice quadrata di ordine n. Inoltre è simmetrica in quanto

$$(A^T A)^T = A^T (A^T)^T = A^T A$$

cioè coincide con la sua trasposta. Infine, se A ha rango massimo, è definita positiva poiché se y è un qualunque vettore di lunghezza n non nullo si ha

$$y^T (A^T A) y = (y^T A^T)(Ay) = (Ay)^T (Ay) = ||Ay||_2^2 > 0$$

(se il rango non è massimo può accadere che $Ay = 0$, con $y \neq 0$).

Interpretazione geometrica Vogliamo ora dare un'interpretazione geometrica del problema dei minimi quadrati. Se A è di ordine $m \times n$ allora può essere pensata come un'applicazione lineare di $\mathbb{R}^n \to R(A) \subseteq \mathbb{R}^m$. Il sottospazio $R(A)$ delle immagini è l'insieme dei vettori u tali che esiste un $y \in \mathbb{R}^n$ tale che $u = Ay$. Sia ora $b \in \mathbb{R}^m$. Poiché la norma 2 è quella euclidea, $||b - Ax||_2$ rappresenta la distanza tra i vettori b e Ax.

È evidente allora che affinché tale distanza sia minima deve accadere che $b - Ax$ sia ortogonale a $R(A)$. Ora, ogni vettore di $R(A)$ può essere visto come combinazione lineare delle colonne di A. Infatti se $u \in R(A)$ è tale che $u = Ay$, ciò significa che

$$u = \begin{pmatrix} u_1 \\ \vdots \\ u_m \end{pmatrix} = \begin{pmatrix} a_{11}y_1 + \cdots + a_{1n}y_n \\ \vdots \\ a_{m1}y_1 + \cdots + a_{mn}y_n \end{pmatrix}$$

cioè

$$u = y_1 a_1 + y_2 a_2 + \cdots + y_n a_n$$

dove a_j denota la colonna j-esima di A.

Dunque, affinché $b - Ax$ risulti ortogonale a $R(A)$, basta che sia ortogonale a tutte le colonne di A, cioè

$$a_j^T (b - Ax) = 0, \quad j = 1, \ldots, n \iff A^T (b - Ax) = 0 \iff A^T Ax = A^T b$$

Si perviene così, anche per via geometrica, all'equivalenza tra la soluzione del problema dei minimi quadrati e quello del sistema delle equazioni normali.

La pseudoinversa

Definizione 4.1. Sia A una matrice $m \times n$, $m \geq n$ e con rango massimo ($\text{rank}(A) = n$). Si definisce *pseudoinversa* o inversa generalizzata di Moore-Penrose di A la matrice

$$A^+ = (A^T A)^{-1} A^T.$$

La pseudoinversa generalizza il concetto di inversa di una matrice nel caso delle matrici rettangolari. Infatti

$$A^+ A = (A^T A)^{-1} A^T A = I$$

D'altra parte se A è quadrata e non singolare si ha

$$A^+ = (A^T A)^{-1} A^T = A^{-1} (A^T)^{-1} A^T = A^{-1}$$

cioè inversa e pseudoinversa coincidono.

Il legame tra pseudoinversa e problema dei minimi quadrati è fornito dalla seguente

Proposizione 4.2. *Sia A di ordine $m \times n$, $m \geq n$ e con rango massimo ($\text{rank}(A) = n$) e b un vettore di lunghezza m. L'unica soluzione del problema dei minimi quadrati relativo a tali dati è data da*

$$x = A^+ b \equiv (A^T A)^{-1} A^T b.$$

La Proposizione è immediata conseguenza del teorema di esistenza e unicità. Infatti abbiamo già detto che la soluzione del PMQ è quella del sistema delle equazioni normali

$$A^TAx = A^Tb \implies x = (A^TA)^{-1}A^Tb$$

La matrice pseudoinversa ha diverse e utili applicazioni, non solo nella soluzione del problema dei minimi quadrati. Un'importante informazione che si può dedurre dalla conoscenza di A^+ è il condizionamento del problema dei minimi quadrati. È infatti possibile dimostrare che, se si introducono delle perturbazioni su b ed A, allora la corrispondente perturbazione sul vettore x, soluzione del problema dei minimi quadrati, è proporzionale a quelle introdotte sui dati mediante la quantità

$$pcond(A) = ||A||||A^+||$$

Risoluzione del sistema delle equazioni normali Abbiamo visto come la soluzione del problema dei minimi quadrati coincida, nel caso sovradeterminato e di rango massimo, con la soluzione del sistema delle equazioni normali:

$$A^TAx = A^Tb$$

Dunque per calcolare numericamente la soluzione del PMQ è possibile risolvere numericamente il sistema delle equazioni normali. Poiché la matrice dei coefficienti è simmetrica e definita positiva, possiamo usare il metodo di eliminazione di Gauss senza pivoting o, con un costo computazionale dimezzato, il metodo di Cholesky.

Dunque il problema dei minimi quadrati può essere risolto nel seguente modo:

- si forma la matrice A^TA;
- si fattorizza A^TA mediante fattorizzazione di Cholesky, ovvero si computa L tale che $A^TA = LL^T$;
- si forma il vettore $c = A^Tb$;
- si risolve la coppia di sistemi lineari triangolari:

$$\begin{cases} Ly = c \\ L^Tx = y \end{cases}$$

Costo computazionale e stabilità Il costo computazionale complessivo della procedura è di circa

$$m\frac{n^2}{2} + \frac{n^3}{6}$$

operazioni, in quanto ne occorrono circa $m\frac{n^2}{2}$ per formare A^TA (che si costruisce per simmetria) e $\frac{n^3}{6}$ per la fattorizzazione di Cholesky. Le sostituzioni all'indietro e in avanti costano complessivamente n^2 e sono trascurate. Dunque la procedura è relativamente economica, in quanto l'operazione più costosa (la fattorizzazione di Cholesky) viene effettuata sulla matrice di ordine n (che in genere è molto minore di m).

Sfortunatamente la procedura appena descritta è complessivamente instabile. Infatti il primo problema è la costruzione della matrice A^TA. Può accadere che vi sia una perdita di cifre significative nella rappresentazione di macchina di A^TA. Nei casi più patologici può verificarsi che la matrice "di macchina" non sia più definita positiva o addirittura che risulti singolare. È stato infatti provato che se $pcond(A) > 10^{\frac{t}{2}}$, dove t è la precisione della rappresentazione floating point, non è più assicurato che $fl(A^TA)$ sia non singolare.

Esempio di instabilità Consideriamo la matrice

$$A = \begin{pmatrix} 1 & 1 \\ 10^{-4} & 0 \\ 0 & 10^{-4} \end{pmatrix}$$

e assumiamo di lavorare in precisione $t = 8$. Le colonne di A sono linearmente indipendenti e quindi il rango è 2. Computiamo ora A^TA.

$$A^TA = \begin{pmatrix} 1 + 10^{-8} & 1 \\ 1 & 1 + 10^{-8} \end{pmatrix}$$

Poiché $t = 8$, allora $fl(A^TA) = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$ che è singolare! Si noti che $pcond(A) = 1.4142... \cdot 10^4 > 10^4$.

Anche se non si incorre in questi casi "patologici" tuttavia la procedura è generalmente poco stabile in quanto, mentre il PMQ ha un condizionamento che è regolato da $pcond(A)$, il sistema delle equazioni normali ha un condizionamento che dipende da $cond(A^TA)$. Si può provare che (in norma 2):

$$cond(A^TA) = pcond(A)^2$$

Concludendo possiamo dire che alla risoluzione mediante equazioni normali, benché economica, si deve ricorrere solo quando si hanno informazioni sul buon condizionamento della matrice A^TA. Nei casi in cui si è incerti sul condizionamento del problema conviene utilizzare altri metodi, più costosi, ma stabili.

Metodo mediante fattorizzazione QR

La fattorizzazione QR È noto che le fattorizzazioni di matrici costituiscono un efficace strumento in algebra lineare. Le più note sono la fattorizzazione LU e la fattorizzazione LL^T di Cholesky (per matrici simmetriche e definite positive). Entrambe queste fattorizzazioni sono per matrici quadrate. Il costo computazionale per la costruzione di L e U è di circa $\frac{m^3}{3}$ operazioni, mentre la fattorizzazione di Cholesky costa $\frac{m^3}{6}$ operazioni.

Illustriamo ora un'altra fattorizzazione che prende il nome di fattorizzazione QR e che può essere costruita anche per matrici rettangolari. Data una matrice A di ordine $m \times n$ esistono due matrici, Q quadrata di ordine m e ortogonale (cioè tale che $Q^{-1} = Q^T$) e R di ordine $m \times n$, del tipo:

$$R = \begin{pmatrix} R_1 \\ 0 \end{pmatrix}$$

con R_1 triangolare superiore di ordine n, se $m > n$, oppure del tipo $(R_1 \quad 0)$ con R_1 triangolare superiore di ordine m, se $m < n$.

Nel caso A sia quadrata i fattori Q ed R sono entrambi quadrati e R è triangolare superiore. Esistono vari algoritmi per la costruzione della fattorizzazione QR, il più utilizzato dei quali (quello che adopera le matrici elementari di Householder) ha un costo computazionale di $\frac{2}{3}m^3$, nel caso della matrice quadrata e piena. Nel caso rettangolare con $m > n$ invece il costo è dell'ordine di:

$$n^2(m - \frac{n}{3})$$

Confrontando dunque su matrici quadrate, la fattorizzazione QR "costa di più" della fattorizzazione LU. Il valore aggiunto però sta nel fatto che il fattore Q è ortogonale. Le matrici ortogonali rivestono un'importanza rilevante per le loro numerose proprietà:

- Non è necessario costruire le inverse, che sono disponibili con la sola trasposizione ($Q^{-1} \equiv Q^T$), il che implica $Q^TQ = QQ^T = I$.
- Se y è un qualsiasi vettore di lunghezza m si ha:

  $$||Qy||_2 = \sqrt{(Qy)^T(Qy)} = \sqrt{y^TQ^TQy} = \sqrt{y^Ty} = ||y||_2$$

  cioè la moltiplicazione per matrici ortogonali lascia invariata la norma del vettore (lo stesso vale per Q^T).
- Da quanto sopra si ha $||Q||_2 = 1$, $||Q^T||_2 = 1$, da cui $cond(Q) = 1$, cioè le matrici ortogonali sono perfettamente condizionate.

Il metodo per il PMQ basato sulla fattorizzazione QR Ricordiamo che il problema dei minimi quadrati lineari consiste nel cercare il minimo, su tutti i vettori di lunghezza n, della quantità $||Ax - b||_2$ dove A è una matrice di ordine $m \times n$, con $m \geq n$, $rank(A) = n$ e b è di dimensione m. Un metodo numerico alternativo è basato sull'utilizzo della fattorizzazione QR di A.

Siano infatti Q e R i fattori di A, si ha:

$$Q^TA = \begin{pmatrix} R_1 \\ 0 \end{pmatrix}$$

dove R_1 è quadrata, di ordine n e triangolare superiore. Poiché Q^T è ortogonale, e quindi preserva la norma 2 del vettore, si ha:

$$||Ax - b||_2^2 = ||Q^T(Ax - b)||_2^2 = ||Q^TAx - Q^Tb||_2^2$$

Ponendo ora $Q^Tb = \begin{pmatrix} c \\ d \end{pmatrix}$ con c vettore di lunghezza n, si ha:

$$||Ax - b||_2^2 = || \begin{pmatrix} R_1x \\ 0 \end{pmatrix} - \begin{pmatrix} c \\ d \end{pmatrix} ||_2^2 = || \begin{pmatrix} R_1x - c \\ -d \end{pmatrix} ||_2^2$$

Osserviamo ora che se un vettore y di lunghezza m è decomposto in due sottovettori, per esempio $y = (z, t)^T$ con z di lunghezza n e t di lunghezza $m - n$, si ha che $||y||_2^2 = ||z||_2^2 + ||t||_2^2$. Dunque:

$$||Ax - b||_2^2 = || \begin{pmatrix} R_1 x - c \\ -d \end{pmatrix} ||_2^2 = ||R_1 x - c||_2^2 + ||d||_2^2$$

Poiché il secondo termine della somma non dipende da x ed è sempre positivo, il minimo del primo membro sarà assunto se $||R_1 x - c||_2 = 0$, se cioè $R_1 x = c$, ovvero se x è soluzione del sistema triangolare con R_1 come matrice dei coefficienti e c come termine noto.

Procedura e Costo Dunque la procedura proposta può essere riassunta come segue:

- si effettua la fattorizzazione QR della matrice A e quindi, in particolare, si determina la matrice R_1;
- si forma il vettore c costituito dai primi n elementi del vettore $Q^T b$;
- si risolve il sistema triangolare $R_1 x = c$ (mediante sostituzione all'indietro).

La quantità $||d||_2$ rappresenta la norma 2 del residuo. L'algoritmo ha un costo computazionale che è essenzialmente quello della fattorizzazione QR e pertanto è $n^2(m - \frac{n}{3})$. La procedura descritta risulta inoltre stabile.

Conclusioni Abbiamo proposto due possibili strategie numeriche per la risoluzione del problema dei minimi quadrati lineari, nel caso sia A di dimensioni $m \times n$, con $m \geq n$ e $rank(A) = n$:

1. la risoluzione del sistema delle equazioni normali $A^T Ax = A^T b$;
2. il metodo che utilizza la fattorizzazione QR della matrice A.

Confronto Schemi **Metodo 1 (Equazioni Normali):**

- formare la matrice $A^T A$;
- calcolare la fattorizzazione di Cholesky $A^T A = LL^T$;
- risolvere la coppia di sistemi triangolari $\begin{cases} Ly = A^T b \\ L^T x = y \end{cases}$;
- costo computazionale: $m\frac{n^2}{2} + \frac{n^3}{6}$;
- stabilità: assicurata solo se $cond(A^T A) = pcond(A)^2$ non è grande.

Metodo 2 (Fattorizzazione QR):

- calcolare la fattorizzazione QR di A;
- calcolare $Q^T b = \begin{pmatrix} c \\ d \end{pmatrix}$, dove c ha lunghezza n;
- posto $R = \begin{pmatrix} R_1 \\ 0 \end{pmatrix}$, risolvere il sistema $R_1 x = c$;
- la norma del residuo è data da $||d||_2$;
- costo computazionale: $n^2(m - \frac{n}{3})$;
- stabilità: assicurata sempre.

Functions del Matlab Le funzioni Matlab coinvolte nella risoluzione del PMQ sono le seguenti:

- `A\b`: implementa, nel caso A sia rettangolare, il metodo che utilizza la fattorizzazione QR, cioè il metodo 2;
- `R = chol(B)`: calcola il fattore di Cholesky della matrice simmetrica e definita positiva B ($B = R^T R$);
- `D = pinv(A)`: calcola la pseudoinversa (sinistra) della matrice A;
- `[Q,R] = qr(A)`: calcola la fattorizzazione QR della matrice A (quadrata o rettangolare) mediante trasformazioni di Householder.

5 Metodi Numerici per la Risoluzione di una Equazione non Lineare

5.1 Risoluzione di Equazioni non Lineari

Introduzione Sia $F \in C^0([a,b])$, cioè F è una funzione continua in un intervallo $[a,b] \subset \mathbb{R}$, tale che $F(a)F(b) < 0$. Vogliamo trovare le radici dell'equazione non lineare

$$F(x) = 0$$

La condizione $F(a)F(b) < 0$ è una condizione sufficiente per l'esistenza di almeno una radice di F in $[a,b]$.

Metodo di Bisezione Questo metodo consiste nel costruire, a partire dall'intervallo $[a,b]$, una successione di intervalli incapsulati

$$[a,b] = [a_0,b_0] \supset [a_1,b_1] \supset \cdots \supset [a_n,b_n]$$

tutti contenenti la radice $\overline{x}$ di F, tale che

$$b_n - a_n \to 0 \quad \text{per } n \to +\infty$$

Gli intervalli $[a_k,b_k]$, $k = 1,\ldots,n$, della successione vengono determinati come segue: dato $[a_{k-1},b_{k-1}]$, determiniamo il punto medio

$$m_k = \frac{a_{k-1} + b_{k-1}}{2}$$

- se $F(m_k) = 0$ allora $\overline{x} = m_k$ e abbiamo terminato;
- se $F(m_k) \neq 0$ poniamo

$$[a_k,b_k] = \begin{cases} [a_{k-1},m_k] & \text{se } F(a_{k-1})F(m_k) < 0 \\ [m_k,b_{k-1}] & \text{se } F(b_{k-1})F(m_k) < 0 \end{cases}$$

Dopo n passi si giunge all'intervallo $[a_n,b_n]$ contenente la radice $\overline{x}$ e di ampiezza

$$b_n - a_n = \frac{b_{n-1} - a_{n-1}}{2} = \frac{b_{n-2} - a_{n-2}}{2^2} = \cdots = \frac{b-a}{2^n}$$

Come stima di $\overline{x}$ scegliamo

$$x_{n+1} = m_{n+1} = \frac{a_n + b_n}{2}$$

così che

$$\overline{x} = x_{n+1} \pm \varepsilon_{n+1}$$

dove

$$\varepsilon_{n+1} = |\overline{x} - x_{n+1}| < \frac{b-a}{2^{n+1}}$$

è l'errore assoluto di approssimazione della radice $\overline{x}$ di F.

Fissata una tolleranza TOLL, è possibile determinare il numero di iterazioni k necessarie per approssimare la radice $\overline{x}$ con precisione TOLL imponendo che

$$|\overline{x} - x_n| < \frac{b-a}{2^n} < \text{TOLL}$$

Infatti, possiamo scrivere

$$\frac{b-a}{\text{TOLL}} < 2^n$$

$$\log\left(\frac{b-a}{\text{TOLL}}\right) < \log(2^n) = n\log 2$$

$$n > \frac{\log\left(\frac{b-a}{\text{TOLL}}\right)}{\log 2}$$

Dunque prendiamo

$$n = \left\lfloor \frac{\log\left(\frac{b-a}{\text{TOLL}}\right)}{\log 2} \right\rfloor + 1$$

dove $\lfloor a \rfloor$ denota la parte intera inferiore di a.

Ordine di convergenza Introduciamo ora una misura della velocità di convergenza di una successione numerica.

Definizione 5.1. Sia $\{x_n\}_n$ una successione convergente al valore $\overline{x}$. Poniamo $\varepsilon_n = |\overline{x} - x_n|$. Se esiste un numero reale $p \geq 1$ e una costante reale positiva C tale che

$$\lim_{n \to +\infty} \frac{\varepsilon_{n+1}}{\varepsilon_n^p} = C$$

allora diciamo che la successione $\{x_n\}_n$ ha ordine di convergenza p.

Poiché per il metodo di bisezione si ha

$$\frac{\varepsilon_{n+1}}{\varepsilon_n} \simeq \frac{1}{2}$$

ovvero l'errore si dimezza (più o meno) ad ogni passo, il suo ordine di convergenza è 1, cioè converge molto lentamente. Va però osservato che il metodo richiede solo la continuità della funzione F e la conoscenza del segno di F negli estremi dell'intervallo $[a, b]$.

Osservazione: Per ogni $n \geq 0$ supponiamo che l'approssimazione x_n di $\overline{x}$ abbia $t_n \in \mathbb{R}$ cifre decimali corrette, cioè

$$\varepsilon_n = \frac{1}{2} 10^{-t_n}$$

Allora, poiché ad ogni passo l'errore si dimezza,

$$\frac{1}{2} 10^{-t_{n+1}} = \varepsilon_{n+1} = \frac{1}{2} \varepsilon_n = \frac{1}{4} 10^{-t_n}$$

ovvero

$$10^{-t_{n+1}} = \frac{1}{2} 10^{-t_n}$$

$$t_{n+1} = t_n + \log_{10} 2 \approx t_n + 0.3010, \quad n \geq 0$$

Dunque t_n aumenta di una unità dopo almeno 4 iterazioni. 3 iterazioni non sono sufficienti per ottenere un'approssimazione di $\overline{x}$ con una cifra decimale corretta in più.

Metodi Iterativi Partendo dal punto iniziale x_0, generiamo i valori $x_1, x_2, \ldots, x_n$ nel seguente modo:

- Conduciamo dal punto $(x_0, F(x_0))$ una retta con pendenza m_0 e scegliamo come approssimazione x_1 l'intersezione di tale retta con l'asse delle x;
- Conduciamo dal punto $(x_1, F(x_1))$ una retta con pendenza m_1 e scegliamo come approssimazione x_2 l'intersezione di tale retta con l'asse delle x;

Dunque ad ogni passo scegliamo come approssimazione della radice $\overline{x}$ di F, la radice dell'equazione lineare

$$F(x_n) + m_n(x - x_n) = 0$$

cioè

$$x_{n+1} = x_n - \frac{F(x_n)}{m_n} \quad (1)$$

Una formula del tipo (1) viene detta *formula iterativa* e si dice che la successione $x_1, x_2, \ldots, x_n, \ldots$ viene costruita mediante un procedimento iterativo.

La funzione

$$g(x) = x - \frac{F(x)}{m_n}$$

tale che $x_{n+1} = g(x_n)$ viene detta *funzione di iterazione semplice* perché per la costruzione della successione $\{x_n\}_n$ utilizza solo il punto x_n. Si parla di procedimenti iterativi multipli quando

$$x_{n+k} = g(x_{n+k-1}, x_{n+k-2}, \ldots, x_n), \quad k > 1$$

cioè la successione $\{x_n\}_n$ è costruita a partire dai punti $x_0, x_1, \ldots, x_{k-1}$.

Criteri di Arresto Quando si implementa in maniera automatica un procedimento iterativo del tipo $x_{n+1} = g(x_n)$, $n = 0, 1, \ldots$, occorrono uno o più criteri per arrestare tale procedimento. Fissata una tolleranza TOLL arbitrariamente piccola, i criteri di arresto usualmente utilizzati sono: 1. $|F(x_n)| <$ TOLL (tanto più è piccolo TOLL tanto più x_n è vicino ad $\overline{x}$); 2. $|x_{n+1} - x_n| <$ TOLL oppure $\frac{|x_{n+1}-x_n|}{|x_{n+1}|} <$ TOLL (tanto più è piccolo TOLL tanto più x_n è vicino al limite della successione $\overline{x}$); 3. numero delle iterazioni ¡ ITMAX, dove ITMAX è una variabile intera fissata.

Il criterio 3 entra in gioco quando i primi due falliscono. Per questo motivo la variabile ITMAX viene anche detta variabile tappo. In generale è preferibile utilizzare tutti e tre i criteri, perché per particolari funzioni si può verificare che la convergenza sia troppo lenta oppure che i criteri 1 e/o 2 non vengano mai soddisfatti numericamente.

Metodo di Newton o delle tangenti Nella formula

$$x_{n+1} = x_n - \frac{F(x_n)}{m_n}$$

le direzioni m_n possono essere scelte in vari modi. In questo metodo la direzione scelta ad ogni passo è quella della tangente alla curva $y = F(x)$ nel punto x_n, cioè

$$m_n = F'(x_n)$$

Dunque il metodo è dato da

$$x_{n+1} = x_n - \frac{F(x_n)}{F'(x_n)}, \quad n = 0, 1, \ldots$$

La funzione di iterazione è quindi

$$g(x) = x - \frac{F(x)}{F'(x)}$$

Appare evidente che il metodo perde di significato se per qualche n risulta $F'(x_n) = 0$. Dunque in generale si suppone che $F'(x) \neq 0$, $\forall x \in [a, b]$.

La formula iterativa ci permette di calcolare la successione $x_1, x_2, \ldots$ a partire dal punto x_0. *Domanda:* Come scegliamo il punto x_0?

Teorema di Convergenza Il seguente teorema stabilisce delle condizioni per la determinazione di un intervallo $[a, b]$ per il quale il metodo di Newton converge per ogni scelta di $x_0 \in [a, b]$.

Teorema 5.1. *Sia $F(x) \in C^2([a, b])$ con $[a, b]$ chiuso e limitato. Se:*

- $F(a)F(b) < 0$;
- $F'(x) \neq 0$, $\forall x \in [a, b]$;
- $F(x) \geq 0$ *oppure* $F(x) \leq 0$, $\forall x \in [a, b]$;
- $|\frac{F(a)}{F'(a)}| < (b - a)$ *e* $|\frac{F(b)}{F'(b)}| < (b - a)$;

allora il Metodo di Newton converge all'unica soluzione $\overline{x} \in [a, b]$ per ogni scelta di $x_0 \in [a, b]$.

Proof. Dalle condizioni 1-3 segue l'unicità della soluzione dell'equazione $F(x) = 0$ nell'intervallo $[a, b]$. Facciamo prima vedere che la condizione 4 implica che la tangente alla curva di equazione $y = F(x)$ negli estremi dell'intervallo interseca l'asse x all'interno di $[a, b]$. Conseguentemente, se prendiamo come punto iniziale x_0 un punto appartenente all'intervallo $[a, b]$ (a e b inclusi) tutti i punti della successione $\{x_k\}_{k=1,2,\ldots}$ appartengono all'intervallo $[a, b]$.

La retta tangente in $(a, F(a))$ ha equazione

$$y = F'(a)(x - a) + F(a)$$

ed ha dunque intersezione con l'asse x in

$$x = a - \frac{F(a)}{F'(a)}$$

Poiché $\frac{F(a)}{F'(a)}$ sotto le condizioni 1-3 è sempre negativa e, usando l'ipotesi 4, risulta

$$|x - a| = \left|\frac{F(a)}{F'(a)}\right| < b - a$$

otteniamo $a < x < b$. Analogamente si ragiona per l'altro estremo.

Facciamo ora vedere che la successione $\{x_k\}_{k=1,2,\dots}$ che si costruisce a partire da $x_0 \in [a,b]$ è monotona e limitata. Conduciamo la dimostrazione in una delle possibili quattro situazioni che si possono verificare nelle ipotesi 1-3 (la dimostrazione negli altri casi è simile). Supponiamo allora che sia $F' > 0$, $F \leq 0$, $F(a) < 0$, $F(b) > 0$. Dimostriamo dunque che $x_k \leq \overline{x}$ e $x_{k+1} \geq x_k$. Incominciamo per $k = 0$ e poi procederemo per induzione. Senza ledere la generalità, prendiamo x_0 nell'intervallo $[a, \overline{x}]$ e quindi tale che $F(x_0) \leq 0$. Si ha allora

$$x_1 = x_0 - \frac{F(x_0)}{F'(x_0)} \geq x_0$$

Dunque $x_0 \leq \overline{x}$ e $x_1 \geq x_0$.

Poiché la proprietà è vera per $k = 0$, dimostriamo che, supposta vera per k, essa è verificata per $k+1$. Si ha

$$x_{k+1} = x_k - \frac{F(x_k)}{F'(x_k)}$$

e, per il Teorema di Lagrange,

$$-F(x_k) = F(\overline{x}) - F(x_k) = F'(\xi_k)(\overline{x} - x_k), \quad x_k \leq \xi_k \leq \overline{x}$$

Dal momento che per ipotesi $F(x) \leq 0$, si ha che F' è decrescente e quindi $F'(\xi_k) \leq F'(x_k)$, da cui

$$-F(x_k) \leq F'(x_k)(\overline{x} - x_k)$$

Di conseguenza

$$x_{k+1} = x_k - \frac{F(x_k)}{F'(x_k)} \leq x_k + (\overline{x} - x_k) = \overline{x}$$

$$F(x_{k+1}) \leq 0 \implies x_{k+2} = x_{k+1} - \frac{F(x_{k+1})}{F'(x_{k+1})} \geq x_{k+1}$$

Per completare la dimostrazione, basta osservare che la successione $\{x_k\}_{k=1,2,\dots}$ essendo monotona e limitata, è convergente. Inoltre passando al limite nella formula iterativa si deduce immediatamente che essa converge ad $\overline{x}$. Infatti, detto $\beta = \lim_{k\to\infty} x_k$, si ha

$$\beta = \lim_{k\to\infty} x_{k+1} = \lim_{k\to\infty} x_k - \lim_{k\to\infty} \frac{F(x_k)}{F'(x_k)} = \beta - \frac{F(\beta)}{F'(\beta)}$$

da cui $F(\beta) = 0$ e quindi $\beta = \overline{x}$ essendo, date le nostre ipotesi, $\overline{x}$ l'unico zero di F nell'intervallo $[a,b]$. □

Ordine di convergenza del Metodo di Newton Per determinare l'ordine di convergenza utilizziamo il seguente

Teorema 5.2. *Condizione necessaria e sufficiente affinché un procedimento iterativo semplice e convergente ad $\overline{x}$ abbia ordine di convergenza p è che, se la funzione di iterazione g è dotata di derivata p-esima continua per $x = \overline{x}$ risulti*

$$g'(\overline{x}) = g(\overline{x}) = \cdots = g^{(p-1)}(\overline{x}) = 0 \quad e \quad g^{(p)}(\overline{x}) \neq 0$$

Per il Metodo di Newton si ha:

$$g'(x) = 1 - \frac{(F'(x))^2 - F(x)F(x)}{(F'(x))^2} = \frac{F(x)F(x)}{(F'(x))^2}$$

ma, poiché assumiamo che $F(\overline{x}) = 0$ si ha

$$g'(\overline{x}) = 0$$

e quindi

$$g(x) = \frac{(F'(x)F(x) + F(x)F'(x))(F'(x))^2 - 2F'(x)F(x)(F(x))^2}{(F'(x))^4}$$

$$g(\overline{x}) = \frac{F(\overline{x})}{F'(\overline{x})}$$

che, in generale, è diverso da 0. Dunque, in generale, il metodo ha ordine di convergenza 2.

Domanda: Cosa succede se $\overline{x}$ è uno zero doppio, cioè $F(\overline{x}) = F'(\overline{x}) = 0$ e $F(\overline{x}) \neq 0$? In questo caso non è possibile calcolare né g né g' in $\overline{x}$, ma è possibile definirli per continuità:

$$g(\overline{x}) = \overline{x} - \lim_{x\to\overline{x}} \frac{F(x)}{F'(x)} = \overline{x} - \lim_{x\to\overline{x}} \frac{F'(x)}{F(x)} = \overline{x}$$

$$g'(\overline{x}) = F(\overline{x}) \lim_{x\to\overline{x}} \frac{F(x)}{(F'(x))^2} = F(\overline{x}) \lim_{x\to\overline{x}} \frac{F'(x)}{2F'(x)F(x)} = \frac{1}{2}$$

Dunque se $\overline{x}$ è uno zero doppio, essendo $g'(\overline{x}) \neq 0$, il metodo di Newton ha ordine di convergenza 1. Più in generale, se $\overline{x}$ è uno zero d'ordine r, cioè

$$F(\overline{x}) = F'(\overline{x}) = \cdots = F^{(r-1)}(\overline{x}) = 0 \quad \text{e} \quad F^{(r)}(\overline{x}) \neq 0$$

allora

$$g'(\overline{x}) = 1 - \frac{1}{r} \neq 0$$

dunque il metodo di Newton ha ancora ordine di convergenza 1.

Metodo di Newton per radici multiple Quando $\overline{x}$ è uno zero multiplo è possibile modificare leggermente il metodo di Newton per "recuperare" l'ordine 2. Si considera la seguente formula iterativa:

$$x_{n+1} = x_n - 2\frac{F(x_n)}{F'(x_n)}$$

la cui funzione di iterazione è

$$g(x) = x - 2\frac{F(x)}{F'(x)}$$

La precedente formula iterativa ha ordine di convergenza 2, infatti:

$$g'(x) = 1 - 2\frac{(F'(x))^2 - F(x)F(x)}{(F'(x))^2} = 2\frac{F(x)F(x)}{(F'(x))^2} - 1$$

otteniamo

$$g'(\overline{x}) = 2F(\overline{x}) \lim_{x\to\overline{x}} \frac{F(x)}{(F'(x))^2} - 1 = 1 - 1 = 0$$

In generale se $\overline{x}$ è una radice multipla di ordine r si utilizza la seguente formula iterativa

$$x_{n+1} = x_n - r\frac{F(x_n)}{F'(x_n)}, \quad n = 0, 1, \ldots$$

il cui ordine di convergenza è ancora 2. Esistono opportune modifiche anche nei casi in cui la molteplicità della radice è > 1 ma non è nota e nei casi in cui la molteplicità è infinita. Esistono delle varianti del Metodo di Newton che hanno ordine di convergenza > 2.

Osservazione: Quando la funzione F ha radici multiple, abbiamo visto che da un punto di vista teorico il Metodo di Newton converge. Tuttavia, spesso si verifica che esso numericamente non converge. La stessa cosa certe volte accade quando F ha radici "quasi multiple".

Esempi *Esempio 1:* Data l'equazione

$$\cos x - 4x = 0 \quad (2)$$

determinare un intervallo dell'asse reale che contenga l'unico zero di tale equazione e sia tale che il Metodo di Newton risulti convergente. *Svolgimento:* Localizziamo graficamente lo zero dell'equazione (2) trovando il punto di intersezione delle curve $y = \cos x$ e $y = 4x$. Si vede che lo zero $\overline{x}$ cade nell'intervallo $[0, \frac{1}{2}]$. Vediamo se tale intervallo è il giusto candidato per la convergenza del Metodo di Newton. Prima verifichiamo le condizioni agli estremi. Si ha

$$F(0)F(1/2) = 1 \cdot (-1.12241) < 0$$

Ponendo $F(x) = \cos x - 4x$ si ha

$$F'(x) = -\sin x - 4 \Rightarrow F'(x) < 0 \forall x \in \mathbb{R}$$

$$F(x) = -\cos x \Rightarrow F(x) \leq 0 \text{ se } \cos x \geq 0$$

$$F(x) \leq 0 \quad \forall x \in [0, 1/2]$$

Inoltre

$$\left|\frac{F(0)}{F'(0)}\right| = \left|\frac{1}{-4}\right| = \frac{1}{4} < \frac{1}{2}$$

$$\left|\frac{F(1/2)}{F'(1/2)}\right| = \left|\frac{-1.12241}{-4.47942}\right| = 0.250571\cdots < \frac{1}{2}$$

Dunque il Metodo di Newton converge $\forall x_0 \in [0, 1/2]$.

Esempio 2: Data l'equazione

$$e^x + \frac{x}{10} = 0 \quad (3)$$

determinare un intervallo dell'asse reale che contenga l'unico zero di tale equazione e sia tale che il Metodo di Newton risulti convergente. *Svolgimento:* Localizziamo graficamente lo zero dell'equazione (3) trovando il punto di intersezione delle curve $y = e^x$ e $y = -x/10$. Si vede che lo zero $\overline{x}$ cade nell'intervallo $[-2, -3/2]$. Vediamo se tale intervallo è il giusto candidato per la convergenza del Metodo di Newton. Prima verifichiamo le condizioni agli estremi. Si ha

$$F(-2)F(-3/2) = (-0.64665e-003)\cdot(0.73130e-003) < 0$$

Ponendo $F(x) = e^x + x/10$ si ha

$$F'(x) = e^x + 1/10 \Rightarrow F'(x) > 0 \forall x \in \mathbb{R}$$

$$F(x) = e^x \Rightarrow F(x) > 0 \forall x \in \mathbb{R}$$

Inoltre

$$\left|\frac{F(-2)}{F'(-2)}\right| = \left|\frac{-0.64665e-003}{0.23534}\right| = 0.27478 < \frac{1}{2}$$

$$\left|\frac{F(-3/2)}{F'(-3/2)}\right| = \left|\frac{0.73130e-003}{0.32313e-002}\right| = 0.22632e-001\cdots < \frac{1}{2}$$

Dunque il Metodo di Newton converge $\forall x_0 \in [-2, -3/2]$.

Esempio 3: Data l'equazione

$$x + \log x^3 = 0 \quad (4)$$

determinare un intervallo dell'asse reale che contenga l'unico zero di tale equazione e sia tale che il Metodo di Newton risulti convergente. *Svolgimento:* Localizziamo graficamente lo zero dell'equazione (4) trovando il punto di intersezione delle curve $y = \log x$ e $y = -x/3$. Si vede che lo zero $\overline{x}$ cade nell'intervallo $[1/2, 1]$. Vediamo se tale intervallo è il giusto candidato per la convergenza del Metodo di Newton. Prima verifichiamo le condizioni agli estremi. Si ha

$$F(1/2)F(1) = (-1.57944)\cdot 1 < 0$$

Ponendo $F(x) = x + 3\log x$ si ha

$$F'(x) = 1 + \frac{3}{x} \Rightarrow F'(x) > 0 \forall x > 0$$

$$F(x) = -\frac{3}{x^2} \Rightarrow F(x) < 0 \forall x \in [1/2, 1]$$

Inoltre

$$\left|\frac{F(1/2)}{F'(1/2)}\right| = \left|\frac{-1.57944}{7}\right| = 0.22563 < \frac{1}{2}$$

$$\left|\frac{F(1)}{F'(1)}\right| = \left|\frac{1}{4}\right| < \frac{1}{2}$$

Dunque il Metodo di Newton converge $\forall x_0 \in [1/2, 1]$.

Osservazione La convergenza del Metodo di Newton è garantita quando, supposte soddisfatte le ipotesi del teorema, l'approssimazione iniziale x_0 è "sufficientemente" vicina alla radice $\overline{x}$. Pertanto tale metodo si rivela spesso efficiente soprattutto per migliorare un'approssimazione sufficientemente buona ottenuta con un metodo di ordine più basso la cui convergenza è assicurata. In generale, anziché verificare le ipotesi del teorema, è preferibile utilizzare il Metodo di bisezione per determinare un'approssimazione della radice con 1 o 2 cifre decimali corrette, e poi applicare il Metodo di Newton per ottenere con pochissime iterazioni la precisione desiderata.

5.2 Risoluzione di Equazioni Algebriche

Valutazione di un polinomio in un punto Le equazioni algebriche sono equazioni del tipo

$$P(x) = 0$$

dove P è un polinomio di grado n, cioè

$$P(x) = a_1 x^n + a_2 x^{n-1} + \cdots + a_n x + a_{n+1}, \quad a_1 \neq 0, \quad a_k \in \mathbb{R}$$

Le seguenti proprietà sono ben note:

- Un polinomio di grado n ha esattamente n radici, che possono essere reali o complesse, ciascuna contata con la sua moltiplicità (Teorema Fondamentale dell'Algebra);
- Se un polinomio P ha una radice complessa $\alpha = a + ib$ allora la sua coniugata $\overline{\alpha} = a - ib$ è anche radice di P;
- Un polinomio di grado dispari ha almeno una radice reale;
- Per $n \geq 5$ non esiste alcuna forma esplicita per le radici di P.

Per determinare un'approssimazione delle radici reali di un'equazione algebrica è possibile utilizzare i metodi visti per le equazioni non lineari, dopo aver individuato per ciascuna radice reale un intervallo che la contenga. Quando n è molto grande il problema di individuare tali intervalli può risultare non immediato. In generale occorre effettuare delle operazioni preliminari:

- *localizzazione delle radici*: serve a determinare un cerchio del piano complesso che contenga tutte le radici (sia reali che complesse);
- *numerazione delle radici*: consiste nel determinare il numero delle radici reali.

A tale scopo sono utili i seguenti risultati.

Teorema di Cauchy

Teorema 5.3. *Tutti gli zeri di*

$$P(x) = a_1 x^n + a_2 x^{n-1} + \cdots + a_n x + a_{n+1}$$

sono inclusi nel cerchio del piano di centro l'origine e raggio

$$r = 1 + \max_{2 \leq k \leq n+1} \left| \frac{a_k}{a_1} \right|$$

Dunque, in particolare le radici reali si trovano nell'intervallo $[-r, r]$.

Regola dei segni di Cartesio Sia

$$P(x) = a_1 x^n + a_2 x^{n-1} + \cdots + a_n x + a_{n+1}$$

Consideriamo l'insieme ordinato dei coefficienti

$$A = \{a_1, a_2, \ldots, a_n, a_{n+1}\}$$

Se denotiamo con ν il numero di variazioni di segno nell'insieme A (gli eventuali zeri non si contano) si ha che il numero k delle radici reali positive di P soddisfa le seguenti proprietà:

$$k \leq \nu \quad \text{e} \quad \nu - k \text{ è un numero pari.}$$

Se applichiamo la regola di Cartesio al polinomio

$$Q(x) = P(-x)$$

otteniamo il numero delle radici reali positive di Q e quindi il numero delle radici reali negative di P.

Applicando i precedenti risultati e disegnando il grafico di P sull'intervallo $[-r, r]$ si stabilisce con certezza il numero delle radici reali di P e si determinano tanti intervallini quante sono le radici di P. Si può così procedere al calcolo approssimato delle radici reali di P utilizzando i metodi visti per le equazioni non lineari.

Osservazioni È possibile utilizzare il metodo di Newton anche per approssimare le radici complesse di un polinomio scegliendo opportunamente il punto x_0. Esistono delle varianti del Metodo di Newton che permettono di approssimare contemporaneamente tutte le radici del polinomio, sia quelle reali che quelle complesse.

Esempio 1 Sia

$$P(x) = x^6 - x - 1$$

Poiché

$$\max_{2\leq k\leq n+1} \left|\frac{a_k}{a_1}\right| = 1$$

applicando il Teorema di Cauchy, tutte le radici reali di P appartengono all'intervallo $[-2, 2]$. Poiché

$$A = \{1, 0, 0, 0, 0, -1, -1\}$$

applicando la regola di Cartesio si ha $\nu = 1$. Pertanto k deve essere necessariamente 1, cioè P ha una sola radice positiva. Infine, posto

$$Q(x) := P(-x) = x^6 + x - 1$$

si ha

$$A = \{1, 0, 0, 0, 0, 1, -1\}$$

e, applicando ancora la regola di Cartesio, otteniamo anche in questo caso $\nu = 1$, cioè il numero delle radici negative di P è $k = 1$.

Possiamo concludere che $P(x) = x^6 - x - 1$ ha due radici reali e due coppie di radici complesse coniugate appartenenti alla circonferenza del piano complesso di centro 0 e raggio 2. In particolare:

- la radice positiva appartiene all'intervallo $[0, 2]$;
- la radice negativa appartiene all'intervallo $[-2, 0]$.

Esempio 2 Sia

$$P(x) = x^9 + 2x^8 - 3x^7 + x^6 + x^4 - 2x^2 + x - 1$$

Poiché

$$\max_{2\leq k\leq n+1} \left|\frac{a_k}{a_1}\right| = 3$$

applicando il Teorema di Cauchy, tutte le radici reali di P appartengono all'intervallo $[-4, 4]$.

Poiché l'insieme dei coefficienti è

$$A = \{1, 2, -3, 1, 0, 1, 0, -2, 1, -1\}$$

applicando la regola di Cartesio si ha $\nu = 5$. Pertanto k, il numero di radici positive, può essere 1, 3 o 5, cioè P ha 1, 3 o 5 radici positive.

Infine, posto

$$Q(x) := P(-x) = -x^9 + 2x^8 + 3x^7 + x^6 + x^4 - 2x^2 - x - 1$$

si ha l'insieme dei coefficienti di Q:

$$A = \{-1, 2, 3, 1, 0, 1, 0, -2, -1, -1\}$$

e, applicando ancora la regola di Cartesio, otteniamo $\nu = 2$. Cioè il numero delle radici negative di P è $k = 0$ o $k = 2$.

Possiamo concludere che tutte le radici di $P(x)$ appartengono alla circonferenza del piano complesso di centro 0 e raggio 4. In particolare, i seguenti casi sono possibili:

- P ha 5 radici positive appartenenti all'intervallo $[0, 4]$ e due coppie di radici complesse coniugate;
- P ha 3 radici positive appartenenti all'intervallo $[0, 4]$ e tre coppie di radici complesse coniugate;
- P ha 1 radice positiva appartenenti all'intervallo $[0, 4]$ e quattro coppie di radici complesse coniugate;
- P ha 2 radici negative appartenenti all'intervallo $[-4, 0]$, 5 radici positive appartenenti all'intervallo $[0, 4]$ e una coppia di radici complesse coniugate;
- P ha 2 radici negative appartenenti all'intervallo $[-4, 0]$, 3 radici positive appartenenti all'intervallo $[0, 4]$ e due coppie di radici complesse coniugate;

- P ha 2 radici negative appartenenti all'intervallo $[-4, 0]$, 1 radice positiva appartenenti all'intervallo $[0, 4]$ e tre coppie di radici complesse coniugate.

Zoommando si può vedere che P ha 3 radici reali di cui:

- 1 radice negativa appartenente all'intervallo $[-\frac{7}{2}, -3]$;
- 1 radice negativa appartenente all'intervallo $[-1, -\frac{1}{2}]$;
- 1 radice positiva appartenente all'intervallo $[\frac{1}{2}, \frac{3}{2}]$.

Valutazione di un polinomio in un punto Consideriamo un polinomio di grado n a coefficienti reali:

$$P(x) = a_1 x^n + a_2 x^{n-1} + \cdots + a_n x + a_{n+1}$$

Il nostro obiettivo è costruire l'algoritmo più efficiente per valutare P in un punto fissato $\overline{x}$.

Algoritmo 1: Metodo Immediato

L'approccio più diretto consiste nel calcolare ogni termine della sommatoria singolarmente:

$$P(x) = \sum_{i=0}^{n} a_{i+1} x^{n-i}$$

In questo metodo, ogni potenza x^k viene calcolata da zero moltiplicando x per se stesso $k-1$ volte. Questo comporta un costo computazionale elevato per gradi n grandi.

- *Operazioni moltiplicative:* $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$
- *Operazioni additive:* n

Algoritmo 2: Metodo con Accumulazione delle Potenze

È possibile migliorare l'efficienza evitando di ricalcolare le potenze ogni volta. Riscriviamo il polinomio come:

$$P(x) = a_{n+1} + \sum_{i=0}^{n-1} a_{n-i} x^{i+1}$$

L'idea chiave è quella di accumulare la potenza di x: invece di calcolare x^{i+1} da capo, si utilizza il valore della potenza precedente x^i e lo si moltiplica per x (cioè $x^{i+1} = x \cdot x^i$). In questo modo, il numero di moltiplicazioni necessarie diminuisce drasticamente, passando da una crescita quadratica a una lineare.

- *Operazioni moltiplicative:* $2n$ (una per aggiornare la potenza e una per il coefficiente ad ogni passo)
- *Operazioni additive:* n

Esiste un algoritmo, detto **Algoritmo di Horner**, che permette di valutare P in un punto con un costo computazionale pari a n operazioni moltiplicative e n operazioni additive.

Algoritmo di Horner Per semplificare, supponiamo che P sia di grado 3:

$$P(x) = a_4 + a_3 x + a_2 x^2 + a_1 x^3$$

Mettendo in evidenza la x otteniamo:

$$P(x) = a_4 + x(a_3 + x(a_2 + x a_1))$$

Riepilogando, ponendo

$$b_1 := a_1$$
$$b_2 := a_2 + \overline{x} b_1$$
$$b_3 := a_3 + \overline{x} b_2$$
$$b_4 := a_4 + \overline{x} b_3$$

si ottiene $P(\overline{x}) = b_4$. In generale, per valutare il polinomio $P(x)$ in un punto $\overline{x}$, ponendo:

$$b_1 := a_1$$
$$b_i := a_i + \overline{x} b_{i-1}, \quad i = 2, 3, \ldots, n+1$$

si ha

$$P(\overline{x}) = b_{n+1}$$

Osservazione I numeri $b_i, i = 1, \ldots, n$ non sono altro che i coefficienti del polinomio $Q(x)$ ottenuto come quoziente della divisione del polinomio $P(x)$ per il binomio $(x - \overline{x})$ e $P(\overline{x})$ non è altro che il resto, cioè

$$P(x) = Q(x)(x - \overline{x}) + P(\overline{x})$$

con

$$Q(x) = b_1 x^{n-1} + b_2 x^{n-2} + \cdots + b_{n-1} x + b_n$$

Dunque le formule dell'algoritmo di Horner permettono di calcolare, senza eseguire la divisione, i coefficienti di $Q(x)$ nonché il resto $P(\overline{x})$ secondo lo schema di Ruffini:

	a_1	a_2	a_3	$\ldots$	a_{n-1}	a_n	a_{n+1}
$\overline{x}$		$\overline{x}b_1$	$\overline{x}b_2$	$\ldots$	$\overline{x}b_{n-2}$	$\overline{x}b_{n-1}$	$\overline{x}b_n$
	b_1	b_2	b_3	$\ldots$	b_{n-1}	b_n	b_{n+1}

Schema Algoritmo di Horner

```
START main
in n, a, x
b(1) = a(1)
for i = 2; i <= n; i++
    b(i) = a(i) + x * b(i-1)
px = a(n+1) + x * b(n)
out px, b
END
```

Costo: n operazioni moltiplicative e n operazioni additive.

Esempio Valutare il polinomio

$$P(x) = 2x^9 + 8x^4 - x^3 - 1$$

nel punto $\overline{x} = 1$. Usando lo schema di Ruffini:

	2	0	0	0	0	8	-1	0	0	-1
1		2	2	2	2	2	10	9	9	9
	2	2	2	2	2	10	9	9	9	8

Dunque $P(1) = 8$ e il quoziente è:

$$Q(x) = 2x^8 + 2x^7 + 2x^6 + 2x^5 + 2x^4 + 10x^3 + 9x^2 + 9x + 9$$

Esercizio in Octave

```
>> a = [2 0 0 0 0 8 -1 0 0 -1];
>> [px, b] = horner(a, 1)
px = 8
b = 2 2 2 2 2 10 9 9 9
```

Calcolo della derivata Vediamo ora che, contestualmente al calcolo di $P(\xi)$ è possibile calcolare anche $P'(\xi)$. Si può dimostrare facilmente che

$$P(x) = Q(x)(x - \xi) + P(\xi)$$

dove

$$Q(x) = b_1 x^{n-1} + b_2 x^{n-2} + \cdots + b_{n-1} x + b_n$$

Ricordando la formula di Taylor:

$$P(x) = P(\xi) + P'(\xi)(x - \xi) + \frac{(x - \xi)^2}{2!} P(\xi) + \cdots + \frac{(x - \xi)^n}{n!} P^{(n)}(\xi)$$

possiamo scrivere

$$Q(x) = \frac{P(x) - P(\xi)}{x - \xi} = P'(\xi) + \frac{(x - \xi)}{2!} P(\xi) + \ldots$$

Facendo il limite per $x \to \xi$ otteniamo

$$Q(\xi) = \lim_{x \to \xi} \frac{P(x) - P(\xi)}{x - \xi} = P'(\xi)$$

Dunque per calcolare $P'(\xi)$ basta calcolare $Q(\xi)$. Lo possiamo fare utilizzando ancora l'algoritmo di Horner sui coefficienti b_i. Infatti, ricordando la definizione di $Q(x)$ e ponendo:

$$\begin{cases} c_1 := b_1 \\ c_i = c_{i-1}\xi + b_i, \quad i = 2, \ldots, n \end{cases}$$

si ha

$$P'(\xi) = Q(\xi) = c_n$$

L'algoritmo per calcolare $P(\xi)$ e $P'(\xi)$ è dunque:

```
p = a(1);
dp = p;
for i = 2:n
    p = p * x + a(i);
    dp = dp * x + p;
end
p = p * x + a(n+1);
```

Il suo costo computazionale complessivo è $2n$.

Condizionamento delle radici di un polinomio Sia

$$P(x) = a_1 x^n + a_2 x^{n-1} + \cdots + a_n x + a_{n+1}$$

un polinomio di grado n e siano $\alpha_1, \ldots, \alpha_n$ le sue radici. Per effetto della rappresentazione finita, i dati in ingresso del problema (ovvero i coefficienti) sono affetti da un errore che al massimo è l'epsilon macchina e di conseguenza le radici effettivamente calcolate saranno altre.

Obiettivo dello studio del condizionamento è misurare, a posteriori, la sensibilità del problema posto. Nello stesso polinomio ci possono essere radici ben condizionate ed altre mal condizionate, cioè lo studio non è globale (tutte le radici) ma puntuale (una sola radice alla volta).

Condizionamento di una radice semplice Sia α una generica radice semplice del polinomio $P(x)$ e sia β la corrispondente radice del polinomio che si ottiene perturbando il k-esimo coefficiente. Si dimostra che:

$$\frac{|\alpha - \beta|}{|\alpha|} \leq \frac{\varepsilon}{|\alpha|} \max_{k=1,\ldots,n+1} \left| \frac{a_k \alpha^{n-k+1}}{P'(\alpha)} \right|$$

La quantità

$$\frac{1}{|\alpha|} \max_{k=1,\ldots,n+1} \left| \frac{a_k \alpha^{n-k+1}}{P'(\alpha)} \right|$$

rappresenta l'indice di condizionamento della radice. Si evince che l'errore si amplifica o perché la derivata prima del polinomio computato nella radice è molto piccola (radici quasi doppie) o perché i coefficienti del polinomio sono molto grandi.

Condizionamento di una radice multipla Sia α una generica radice di molteplicità $r > 1$. Si dimostra che:

$$\frac{|\alpha - \beta|}{|\alpha|} \leq \frac{\varepsilon^{1/r}}{|\alpha|} \max_{k=1,\ldots,n+1} \left| \frac{r! a_k \alpha^{n-k+1}}{P^{(r)}(\alpha)} \right|^{1/r}$$

Quindi il problema è in generale mal condizionato.

Problema del calcolo simultaneo di tutti gli zeri Esistono diversi approcci:

1. **Deflazione:** Si calcola una radice x_1 e si costruisce il quoziente $P_1(x) = P(x)/(x - x_1)$. Si itera il procedimento. *Difetti:* L'errore di arrotondamento si accumula in modo inaccettabile fornendo risultati fasulli dopo le prime radici.

2. **Matrice Companion:** Si riduce il calcolo degli zeri di P al calcolo degli autovalori della matrice companion di P, che ha P come polinomio caratteristico.

$$P_n(x) = a_1 x^n + \cdots + a_n x + a_{n+1}$$

$$A = \begin{pmatrix} -\frac{a_2}{a_1} & -\frac{a_3}{a_1} & \dots & -\frac{a_n}{a_1} & -\frac{a_{n+1}}{a_1} \\ 1 & 0 & \dots & 0 & 0 \\ 0 & 1 & \dots & 0 & 0 \\ \vdots & & \ddots & & \vdots \\ 0 & 0 & \dots & 1 & 0 \end{pmatrix}$$

È quello che fa la function `roots` del Matlab. *Difetti:* Il metodo calcola con precisione gli autovalori della matrice, ma ciò non implica una soddisfacente approssimazione degli zeri di P, poiché i due problemi hanno un differente condizionamento.